

# **NA200Pro Programming Software User Manual**

**V4.3**



**Nanda Automation Technology Jiangsu Co., Ltd.**

**September in 2012**

# Content

<b>Chapter 1</b>	<b>Overview</b>	<b>1</b>
1.1	Software features	1
1.1.1	Windows style	1
1.1.2	International standard—IIEC61131-3	2
1.1.3	Project management—Tree management structure	2
1.1.4	Programming language—5 programming languages	2
1.1.5	Programming mode—Cross call	2
1.1.6	Computing capability—Rich computing control function	3
1.1.7	Viewing function—Intuitive online viewing function	3
1.1.8	Modifying function—Complete online modifying function	3
1.1.9	Debugging function—Powerful online debugging function	3
1.1.10	Monitoring function—Real-time online monitoring function	3
1.1.11	Simulating function—Perfect simulating function without hardware	4
1.1.12	Diagnostic tool—Effective diagnostic tools	4
1.1.13	Transfer mode—Standard file transfer mode	4
1.1.14	Chinese programming—Comprehensive support for Chinese	4
1.1.15	Print mode—WYSIWYG print mode	4
1.1.16	User interface—Friendly user interface	4
1.2	System requirements	5
1.2.1	Operation system	5
1.2.2	Hardware	5
1.3	Software installation	5
<b>Chapter 2</b>	<b>Development Environment Operation</b>	<b>6</b>
2.1	Work window	6
2.1.1	Work interface	6
2.1.2	Main functions of each window	7
2.2	Menu introduction	7
2.2.1	Main menu or drop-down menu	7
2.2.2	Sub-menu	8
2.2.3	Shortcut menu or pop-up menu	9
2.3	Menu function	10
2.3.1	File	10
2.3.2	Edit	16
2.3.3	View	21
2.3.4	LD	22
2.3.5	Online	26
2.3.6	Download	28
2.3.7	Window	28
2.3.8	Help	29

2.4	System toolbar	30
2.5	LD toolbar	32
2.6	FBD toolbar	34
2.7	IL toolbar	35
2.8	ST toolbar	36
2.9	Output window	37
<b>Chapter 3 Project Management</b>		<b>39</b>
3.1	Project browser	39
3.2	Create new project	40
3.2.1	Create project	40
3.2.2	PLC hardware configuration	41
3.3	Program	47
3.3.1	Add program	48
3.3.2	Delete program	49
3.3.3	Rename program	49
3.3.4	Describe program	50
3.4	Protect project	50
3.5	Connect and disconnect	51
3.6	Download and upload project	51
3.7	Download and upload program	52
<b>Chapter 4 Data Management</b>		<b>53</b>
4.1	Data type	53
4.2	Data management	54
4.2.1	Data tab	54
4.2.2	Point table	54
4.2.3	Optional point table	58
4.3	Addressing mode	61
<b>Chapter 5 Basic Function Block</b>		<b>63</b>
5.1	Introduction	63
5.1.1	Property modification	63
5.1.2	EN/ENO	64
5.2	Mathematics	65
5.2.1	ADD: 16-bit addition	66
5.2.2	DADD: 32-bit addition	67
5.2.3	EADD: Floating-point addition	68
5.2.4	SUB: 16-bit subtraction	69
5.2.5	DSUB: 32-bit subtraction	71

5.2.6	ESUB: Floating-point subtraction	72
5.2.7	MUL: 16-bit multiplication	73
5.2.8	DMUL: 32-bit multiplication	74
5.2.9	EMUL: Floating-point multiplication	75
5.2.10	DIV: 16-bit division	76
5.2.11	DDIV: 32-bit division	77
5.2.12	EDIV: Floating-point division	78
5.2.13	MOD: 16-bit modulo	79
5.2.14	DMOD: 32-bit modulo	80
5.2.15	DIVMOD: 16-bit division and modulo	81
5.2.16	DDIVMOD: 32-bit division and modulo	82
5.2.17	INC: 16-bit increment	83
5.2.18	DINC: 32-bit increment	84
5.2.19	DEC: 16-bit decrement	85
5.2.20	DDEC: 32-bit decrement	86
5.2.21	NEG: 16-bit negation	87
5.2.22	DNEG: 32-bit negation	88
5.2.23	ENEG: Floating-point negation	89
5.2.24	SIGN: 16-bit sign evaluation	90
5.2.25	DSIGN: 32-bit sign evaluation	91
5.2.26	ESIGN: Floating-point sign evaluation	92
5.2.27	ABS: 16-bit absolute value	93
5.2.28	DABS: 32-bit absolute value	94
5.2.29	EABS: Floating-point absolute value	95
5.2.30	SQRT: Floating-point square root	96
5.2.31	LOG: Floating-point decimal logarithm	97
5.2.32	LN: Floating-point natural logarithm	98
5.2.33	EXP: Floating-point natural exponentiation	99
5.2.34	EXPT: Floating-point exponentiation	100
5.2.35	SIN: Floating-point sine	101
5.2.36	COS: Floating-point cosine	101
5.2.37	TAN: Floating-point tangent	102
5.2.38	ASIN: Floating-point arc sine	103
5.2.39	ACOS: Floating-point arc cosine	104
5.2.40	ATAN: Floating-point arc tangent	105
<b>5.3</b>	<b>Statistics</b>	<b>106</b>
5.3.1	MIN: 16-bit minimum value	107
5.3.2	DMIN: 32-bit minimum value	108
5.3.3	EMIN: Floating-point minimum value	109
5.3.4	MAX: 16-bit maximum value	110
5.3.5	DMAX: 32-bit maximum value	111
5.3.6	EMAX: Floating-point maximum value	112
5.3.7	AVE: 16-bit averaging	113
5.3.8	DAVE: 32-bit averaging	114

5.3.9	EAVE: Floating-point averaging	115
5.3.10	LIMIT: 16-bit limit	116
5.3.11	DLIMIT: 32-bit limit	118
5.3.12	ELIMIT: Floating-point limit	119
5.3.13	SEL: 16-bit 0/1 selection	121
5.3.14	DSEL: 32-bit 0/1 selection	122
5.3.15	ESEL: Floating-point 0/1 selection	123
5.3.16	MUX: 16-bit multiplexer	124
5.3.17	DMUX: 32-bit multiplexer	126
5.3.18	EMUX: Floating-point multiplexer	127
5.3.19	LMT: 16-bit limit detection	129
5.3.20	DLMT: 32-bit limit detection	130
5.3.21	ELMT: Floating-point limit detection	131
<b>5.4</b>	<b>Logic</b>	<b>133</b>
5.4.1	AND: 16-bit AND	134
5.4.2	DAND: 32-bit AND	135
5.4.3	OR: 16-bit OR	136
5.4.4	DOR: 32-bit OR	137
5.4.5	NOT: 16-bit negation	138
5.4.6	DNOT: 32-bit negation	139
5.4.7	XOR: 16-bit exclusive OR	140
5.4.8	DXOR: 32-bit exclusive OR	141
5.4.9	SHL: 16-bit shift left	142
5.4.10	DSHL: 32-bit shift left	143
5.4.11	SHR: 16-bit shift right	144
5.4.12	DSHR: 32-bit shift right	145
5.4.13	ROL: 16-bit rotate left	146
5.4.14	DROL: 32-bit rotate left	147
5.4.15	ROR: 16-bit rotate right	148
5.4.16	DROR: 32-bit rotate right	149
5.4.17	BSET: Bit set	150
5.4.18	BCLR: Bit clear	151
5.4.19	BTST: Bit test	152
5.4.20	R_TRIG: Rising edge detection	153
5.4.21	F_TRIG: Falling edge detection	154
5.4.22	SET: Set	155
5.4.23	RESET: Reset	156
5.4.24	SR: Bistable (set dominant)	157
5.4.25	RS: Bistable (reset dominant)	158
<b>5.5</b>	<b>Comparison</b>	<b>159</b>
5.5.1	EQ: 16-bit equal to	160
5.5.2	DEQ: 32-bit equal to	161
5.5.3	EEQ: Floating-point equal to	162
5.5.4	NE: 16-bit not equal to	163

5.5.5	DNE: 32-bit not equal to _____	164
5.5.6	ENE: Floating-point not equal to _____	165
5.5.7	GT: 16-bit greater than _____	166
5.5.8	DGT: 32-bit greater than _____	167
5.5.9	EGT: Floating-point greater than _____	168
5.5.10	GE: 16-bit greater than or equal to _____	169
5.5.11	DGE: 32-bit greater than or equal to _____	170
5.5.12	EGE: Floating-point greater than or equal to _____	171
5.5.13	LT: 16-bit less than _____	172
5.5.14	DLT: 32-bit less than _____	173
5.5.15	ELT: Floating-point less than _____	174
5.5.16	LE: 16-bit less than or equal to _____	175
5.5.17	DLE: 32-bit less than or equal to _____	177
5.5.18	ELE: Floating-point less than or equal to _____	178
<b>5.6</b>	<b>Conversion _____</b>	<b>179</b>
5.6.1	INT_TO_ASC: Integer to ASCII code _____	179
5.6.2	ASC_TO_INT: ASCII code to integer _____	180
5.6.3	INT_TO_BCD: Integer to BCD code _____	181
5.6.4	BCD_TO_INT: BCD code to integer _____	182
5.6.5	INT_TO_GRY: 16-bit integer to Gray code _____	183
5.6.6	GRY_TO_INT: Gray code to 16-bit integer _____	184
5.6.7	DIT_TO_GRY: 32-bit integer to Gray code _____	185
5.6.8	GRY_TO_DIT: Gray code to 32-bit integer _____	186
5.6.9	INT_TO_DIT: 16-bit integer to 32-bit integer _____	187
5.6.10	INT_TO_RAL: 16-bit integer to floating-point _____	188
5.6.11	RAL_TO_INT: Floating-point to 16-bit integer _____	188
5.6.12	DIT_TO_RAL: 32-bit integer to floating-point _____	189
5.6.13	RAL_TO_DIT: Floating-point to 32-bit integer _____	190
<b>5.7</b>	<b>Data move _____</b>	<b>191</b>
5.7.1	MOVE: Assignment _____	191
5.7.2	DMOVE: 32-bit assignment _____	192
5.7.3	EMOVE: Floating-point assignment _____	193
5.7.4	BLKMOV: Block data move _____	194
5.7.5	BLKCLR: Block data clear _____	195
5.7.6	SWAP: High/low byte swap _____	196
5.7.7	SFL: Bit shift left _____	197
5.7.8	SFR: Bit shift right _____	199
5.7.9	BSFL: Byte shift left _____	200
5.7.10	BSFR: Byte shift right _____	201
5.7.11	WSFL: Word shift left _____	202
5.7.12	WSFR: Word shift right _____	203
5.7.13	XCH: 16-bit data exchange _____	205
5.7.14	DXCH: 32-bit data exchange _____	206
5.7.15	EXCH: Floating-point data exchange _____	207

<b>5.8</b>	<b>Timer</b>	<b>208</b>
5.8.1	TON: On delay timer	208
5.8.2	TOF: Off delay timer	210
5.8.3	TP: Pulse timer	212
<b>5.9</b>	<b>Counter</b>	<b>213</b>
5.9.1	CTU: Up counter	214
5.9.2	CTD: Down counter	215
5.9.3	CTUD: Up/down counter	217
<b>5.10</b>	<b>Control</b>	<b>219</b>
5.10.1	CALL: Call program	219
<b>5.11</b>	<b>PLC</b>	<b>220</b>
5.11.1	PULSE: Pulse digital output	221
5.11.2	AOUT: Analog output	222
5.11.3	FORCE: I/O force	223
5.11.4	UNFORCE: I/O unforce	224
5.11.5	RESH: I/O refresh	225
5.11.6	ENI: Interrupt enable	226
5.11.7	DISI: Interrupt disable	226
5.11.8	ATCH: Interrupt attach	227
5.11.9	DTCH: Interrupt disattach	228
5.11.10	PWM: Pulse width modulation output	229
5.11.11	PWM1: Pulse width modulation output 1	230
5.11.12	PLSY: High speed pulse output	231
5.11.13	PLSY1: High speed pulse output 1	233
5.11.14	PLSR: Acceleration-deceleration high speed pulse output	234
5.11.15	PLSR1: Acceleration-deceleration high speed pulse output 1	236
5.11.16	PLSR2: Acceleration-deceleration high speed pulse output 2	237
5.11.17	ORG: Origin search	239
5.11.18	READ: Special data read	241
5.11.19	WRITE: Special data write	242
5.11.20	XMT: Free port transmit	243
5.11.21	RCV: Free port receive	245
5.11.22	FLASH: Flash data read/write	246
5.11.23	RTC: Real-time clock	247
5.11.24	MODRW: Modbus data read/write	248
<b>5.12</b>	<b>Others</b>	<b>250</b>
5.12.1	PID: PID control	251
5.12.2	LRC: LRC check	253
5.12.3	CRC: CRC check	254
5.12.4	LC: 16-bit linear change	255
5.12.5	DLC: 32-bit linear change	256
5.12.6	ELC: Floating-point linear change	258
5.12.7	SORT: 16-bit data sort	259

5.12.8	DSORT: 32-bit data sort	260
5.12.9	ESORT: Floating-point data sort	262
<b>Chapter 6 LD Programming</b>		<b>264</b>
6.1	Contact, coil and function block	264
6.1.1	Contact	264
6.1.2	Coil	265
6.1.3	Function block	266
6.2	Operation	267
6.2.1	Link	268
6.2.2	Negate	268
6.2.3	Label	269
6.2.4	Return	270
6.2.5	Comment	271
6.2.6	Zoom	271
6.2.7	Insert	271
6.2.8	Remove	271
<b>Chapter 7 FBD Programming</b>		<b>273</b>
7.1	Using FBD programming language	273
7.1.1	Properties of FBD program	273
7.1.2	New FBD program	273
7.2	Edit FBD program	274
7.2.1	Function block	274
7.2.2	Property modification of function block	274
7.3	Operation	277
7.3.1	Link	277
7.3.2	Negate	277
7.3.3	Label	278
7.3.4	Return	279
7.3.5	Comment	279
7.3.6	Zoom	279
7.3.7	Insert	280
7.3.8	Remove	280
<b>Chapter 8 IL Programming</b>		<b>281</b>
8.1	The structure of programming language	281
8.2	Execution order	282
8.3	Instruction interpretation	282
8.3.1	Operand	282
8.3.2	Modifier	282
8.3.3	Operator	284
8.3.4	Label	285

8.3.5	Comment	285
<b>8.4</b>	<b>Operator</b>	<b>286</b>
8.4.1	Load (LD and LDN)	286
8.4.2	Store (ST and STN)	286
8.4.3	Set (S), Reset (R)	287
8.4.4	Logic operation	288
8.4.5	Mathematics operation	292
8.4.6	Comparison operation	295
8.4.7	Jump (JMP, JMPC and JMPCN)	299
8.4.8	Call (CAL, CALC and CALCN)	300
8.4.9	Return (RET, RETC and RETCN)	301
<b>8.5</b>	<b>Function block</b>	<b>302</b>
<b>Chapter 9 ST Programming</b>		<b>303</b>
<b>9.1</b>	<b>Expression</b>	<b>303</b>
9.1.1	Operand	303
9.1.2	Operator table	303
<b>9.2</b>	<b>Operator</b>	<b>304</b>
9.2.1	Parentheses (( ))	304
9.2.2	Negation (NOT)	304
9.2.3	Multiplication (*)	305
9.2.4	Division (/)	305
9.2.5	Modulo (MOD)	305
9.2.6	Addition (+)	305
9.2.7	Subtraction (-)	306
9.2.8	Greater than (>)	306
9.2.9	Greater than or equal to (>=)	306
9.2.10	Equal to (=)	306
9.2.11	Not equal to (<>)	307
9.2.12	Less than (<)	307
9.2.13	Less than or equal to (<=)	308
9.2.14	And (AND)	308
9.2.15	Or (OR)	308
9.2.16	Exclusive or (XOR)	308
9.2.17	Assignment (:=)	309
<b>9.3</b>	<b>Statement</b>	<b>310</b>
9.3.1	Instruction	310
9.3.2	IF...THEN...ELSE...END_IF	310
9.3.3	CASE...OF... ELSE... END_CASE	310
9.3.4	FOR...TO...BY...DO...END_FOR	311
9.3.5	WHILE...DO...END_WHILE	312
9.3.6	REPEAT...UNTIL...END_REPEAT	312
9.3.7	EXIT	313

9.3.8	RETURN	313
9.3.9	Comment	313
9.3.10	GOTO	313
9.4	Function block	314
<b>Chapter 10 Program Debugging</b>		<b>315</b>
10.1	LD / FBD debugging	315
10.1.1	Online modifying	315
10.1.2	Online debugging	317
10.2	IL debugging	318
10.3	ST debugging	319
10.4	Simulator	320
<b>Chapter 11 Communication Protocol</b>		<b>322</b>
11.1	General instructions	322
11.1.1	Exchange properties	322
11.1.2	Message format	323
11.2	Addressing mode	324
11.2.1	Addressing mode 1	325
11.2.2	Addressing mode 2	325
11.2.3	Difference of two addressing modes	325
11.2.4	Special instructions for addressing mode 2	325
11.3	Error response	326
11.4	MODBUS protocol	327
11.4.1	Function code descriptions	327
11.4.2	Function code and data classification	328
11.4.3	Details for function code	329
<b>Chapter 12 Connection with Intelligent Touch Screen</b>		<b>342</b>
12.1	Touch screen application	342
12.2	Relative setup	342
12.3	Data access of touch screen	343
12.3.1	Serial port setup	344
12.3.2	Communication protocol	344
12.3.3	Register access	345
12.3.4	LS access	345

# Chapter 1 Overview

This manual is used to help you create, edit and debug the user programs by NA200Pro programming software.

NA200 PLC is a new generation of programmable logic controller of international advanced level developed by Nanda Automation Technology Co., Ltd. It applies a series of the newest achievements in industrial control field, chooses completely new software and hardware platforms. It has a rapid processing capacity, powerful anti-jamming ability, and flexible expanded capability. It is used easily and smoothly for any complex environments and processing requirements.

NA200Pro programming software is the important part of NA200 PLC, and is the integrated development environment of NA200 PLC, which includes editor, compiler, debugger, emulator and graphic user interface tool, mainly used to complete the hardware configuration, point configuration, software programming, simulating, debugging and downloading. This programming software provides a simple and practical software programming and online debugging tool for the engineering and technical personnel.

NA200Pro provides a series of complete functions that can help to achieve higher productivity and better software interoperability. NA200Pro software can optimize the customer's software investment, reduce training cost, and provides an unmatched potential for development and compatibility by reducing the development cost and optimizing the operation.

NA200Pro provides ladder diagram (LD), function block diagram (FBD), instruction list (IL), and structured text (ST) programming languages according to the IEC61131-3 standard.

## 1.1 Software features

### 1.1.1 Windows style

At present, for certain types of tasks, using the graphic user interface has become a basic need. For this reason, NA200Pro is designed as MS Windows application. NA200Pro can be operated in Windows 2000, Windows XP, Windows NT and Windows Vista which are widely used in the world. PC customers all have the basic knowledge on Windows and mouse operations. NA200Pro is developed by VC++ 6.0 in the Windows environment, so it completely has the same design style with Windows, which has the standard menu operation, shortcut operation, toolbar operation, and mouse operation. It is easy to use and can reduce the training time of programming staff and the programming cost.

## **1.1.2 International standard——IEC61131-3**

Due to the instruction system differences among PLC manufacturers and the different user demands on programming methods, recently IEC has developed Windows-based programming language standard IEC61131-3 (in 1993, IEC promulgated the PLC international standard IEC61131), which specifies the five programming languages of instruction list (IL), ladder diagram (LD), sequential function chart (SFC), function block diagram (FBD) and structured text (ST). It includes textual programming (IL, ST) and graphic programming (LD, FBD), and the SFC can be used in both of the above programming languages. It is a standardized file that the digital technology-based programmable logic control device opens at a high level, which is a major trend of PLC development.

NA200Pro programming software provides a unified and effective system configuration environment according to the international standard IEC61131-3, so the engineer can "learn it once, use it everywhere".

## **1.1.3 Project management——Tree management structure**

NA200Pro programming software applies the project management concepts, displays in a tree structure in the integrated development environment, and visually displays the program contents by the multi-document, in this way, the relevant contents are at a glance, and the program development or maintenance can be implemented intuitively.

## **1.1.4 Programming language——5 programming languages**

As the industrial automation solutions, NA200Pro provides the IEC61131-3 standard compatible programming languages: ladder diagram (LD), instruction list (IL), structured text (ST) and function block diagram (FBD). The programs edited by the above languages can be cross called, so as to make programming more flexible and meet the requirements of various complex conditions. Wherein, the LD and FBD apply graphic editor, are flexible, convenient and fast. All the languages support the shortcut functions of Cut, Copy, Paste, Delete, Undo, Redo, Find and Replace, etc.

## **1.1.5 Programming mode——Cross call**

The control programs are composed of programs with logic structure. In one program, there is only one kind of programming language. All of the programs are combined to establish a complete control project to control the processes. The different IEC language programs (LD, FBD, IL, and ST) all can be cross called in the programs.

### **1.1.6 Computing capability—— Rich computing control function**

NA200Pro programming software is embedded with various standard operators, control function blocks, and standard functions. Moreover, it provides practical function blocks such as the pulse output, interrupt enable/disable, and serial communication etc., so that the engineering technical personnel is easy to solve the complex process control requirements and shorten the project development cycle.

### **1.1.7 Viewing function——Intuitive online viewing function**

In the online condition, it can monitor the LD operating status; the red link indicates ON, the green link indicates OFF. It is very intuitive and at a glance. Also, it can implement the operations of set time, reset, force and unforce etc., so that the engineering staff could easily implement the various functions.

### **1.1.8 Modifying function—— Complete online modifying function**

In the online condition, it can directly modify the parameters of function blocks, add and delete the function blocks, move the function blocks. It can directly transfer the modifications into PLC during operating, keep the program continuity at the same time.

### **1.1.9 Debugging function—— Powerful online debugging function**

All of the ladder diagram (LD), instruction list (IL), structured text (ST), and function block diagram (FBD) support the online debugging functions such as breakpoint and single step execution etc. The engineering staff can easily debug the programs and find the errors.

### **1.1.10 Monitoring function——Real-time online monitoring function**

In the online condition, all of the points can be operated (forced, assigned, and observed) by point table; all of the error messages can be checked by debug tab in output window. The data can be displayed in three ways: decimal, binary or hexadecimal.

### **1.1.11 Simulating function——Perfect simulating function without hardware**

In the simulator condition, programs can be developed and debugged without an actual PLC. NA200Pro can perfectly simulate the hardware functions, exactly reproduce the target program behaviors and effectively reduce the program development period.

### **1.1.12 Diagnostic tool——Effective diagnostic tools**

NA200Pro has comprehensive application diagnostic functions. The output window can display all of the system and application failures. In this window, you only need to double click the mouse button, and then it will access to the editor so as to modify the wrong program.

### **1.1.13 Transfer mode——Standard file transfer mode**

The save, upload and download operations of programming results all apply file mode, in this way, they can keep all the program configurations in conformance.

### **1.1.14 Chinese programming——Comprehensive support for Chinese**

In NA200Pro, it comprehensively supports the Chinese language. Not only the point names, comments and descriptions in programs can be used in Chinese, but also it has Chinese work window, menu, tab, online help and user manual. Moreover, NA200Pro also has convenient Chinese/English comment function; it can conveniently display the desired comment contents in the programs, so as to easily annotate, read, mark, and modify the programs.

### **1.1.15 Print mode——WYSIWYG print mode**

NA200Pro supports the WYSIWYG print mode, to print all produced PLC configuration, point information, ladder diagram (LD) program, function block diagram (FBD) program, instruction list (IL) program and structured text (ST) program, so as to archive.

### **1.1.16 User interface——Friendly user interface**

NA200Pro fully uses the advantages of the Windows graphic and context-sensitive interfaces. Optimizing the use of screen space, direct access to the tools and information, as well as Chinese and English comment etc. all maximize the user friendliness.

## **1.2 System requirements**

### **1.2.1 Operation system**

Windows NT, Windows 2000, Windows XP and Windows Vista.

### **1.2.2 Hardware**

CPU: Pentium 300 and above

Memory: 256M and above

Hard disk: 20G and above

Monitor: Resolution 1024 × 768 and above

If the users use a higher machine configuration, it is recommended to use 1024 × 768 resolution.

## **1.3 Software installation**

Execute the installation software “setup.exe”, and follow the prompts to complete the installation.

# Chapter 2 Development Environment Operation

NA200Pro includes a complete PLC configuration and application development system which conforms to the Windows system operating style and is easy to use.

## 2.1 Work window

### 2.1.1 Work interface

The interface after NA200Pro programming software starts is shown below. The development environment includes the following parts: menu, toolbar, project browser, output window, status bar, and program window, the position of each part is as shown in Fig.2.1.

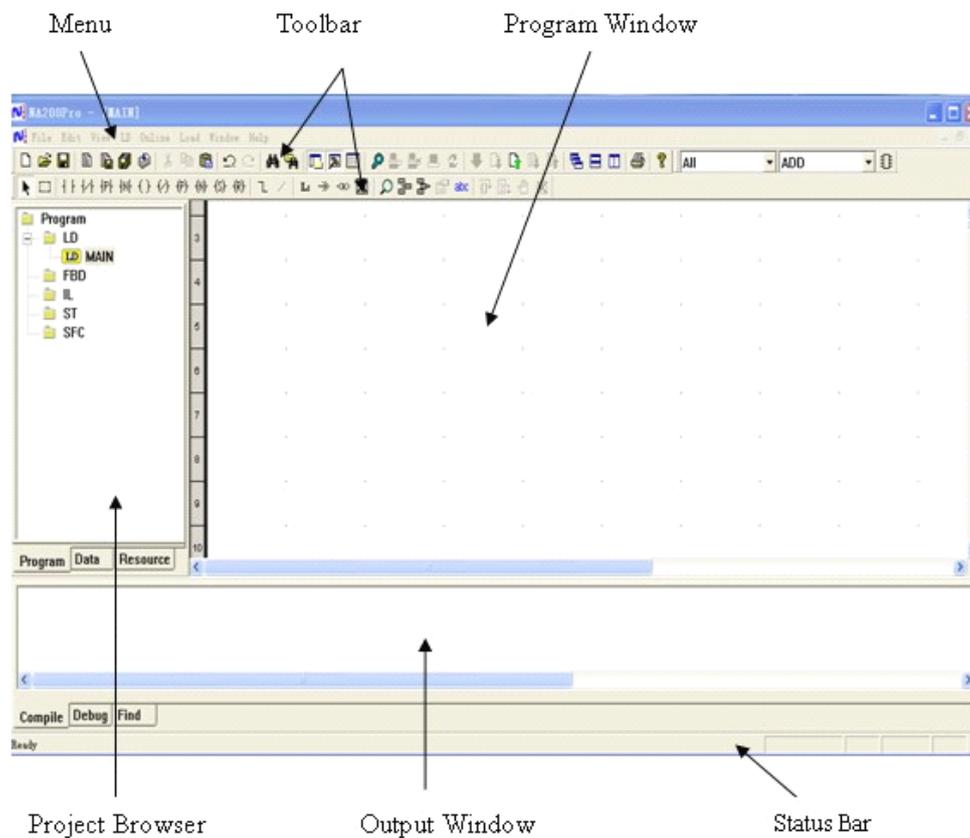


Fig.2.1 Work interface

## 2.1.2 Main functions of each window

**Menu:** Achieve the main functions of programming software.

**Toolbar:** Achieve the file operations of new, open, and save, the online operations of login, upload and download, and the toolbars of each programming language.

**Status bar:** The status bar is at the bottom of the screen. On the right of status bar, there are status information such as program coordinate, online/offline, emulation, and force mark etc. Each of the operation content is shown on the left of status bar.

**Program window:** Achieve system configuration, program editing, and program debugging, etc.

**Project browser:** Achieve management of project.

**Output window:** Display the results of program find, compile, and debug operations.

## 2.2 Menu introduction

### 2.2.1 Main menu or drop-down menu

The main menu can achieve the main functions of programming software, which mainly includes the parts of **File**, **Edit**, **View**, **LD (FBD, IL, ST, etc.)**, **Online**, **Load**, **Window**, and **Help** etc., as shown in Fig.2.2:



Fig.2.2 Main menu

For the drop-down menu, click any item on the main menu by mouse, the item icon will dent, at the same time display a drop-down menu, place the mouse on any menu item of the drop-down menu, the item will turn blue, it is shown that this operation has been chosen, click any place outside the menu or press the **ESC** key to turn off the menu, as shown in Fig.2.3:

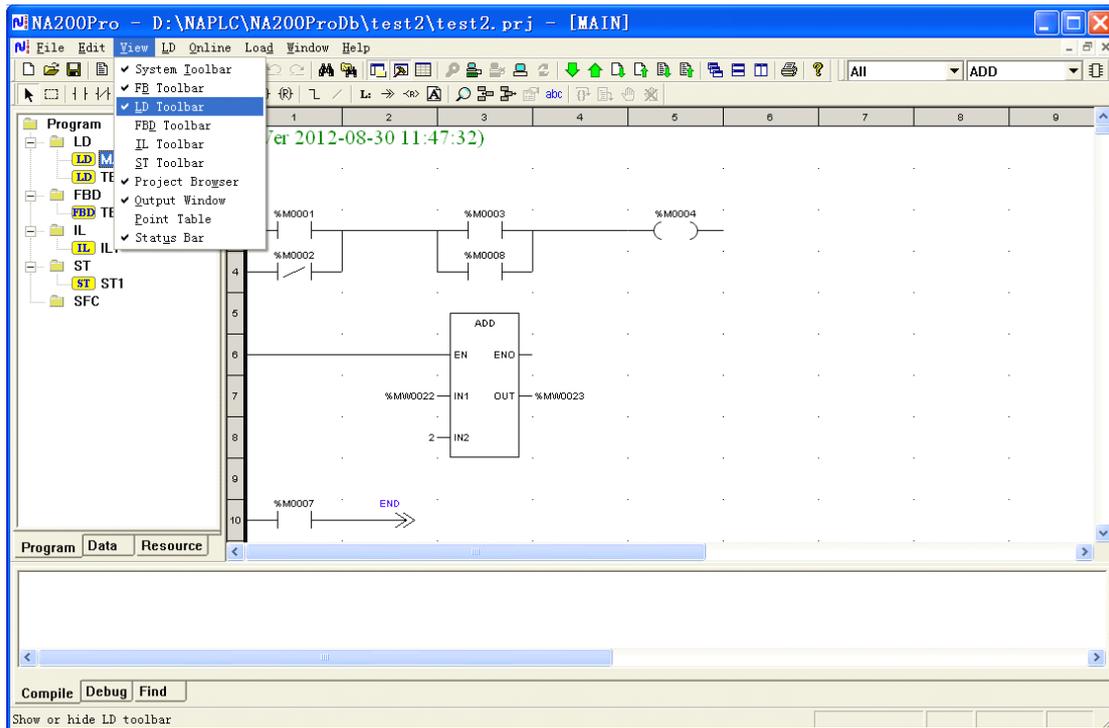


Fig.2.3 Drop-down menu

## 2.2.2 Sub-menu

Each menu item of sub-menu is listed on the drop-down menu. Place the mouse on any of the menu item of sub-menu, this menu item turns blue, it is shown that this operation has been chosen, click any place outside the menu or press the **ESC** key to turn off the menu, as shown in Fig.2.4:

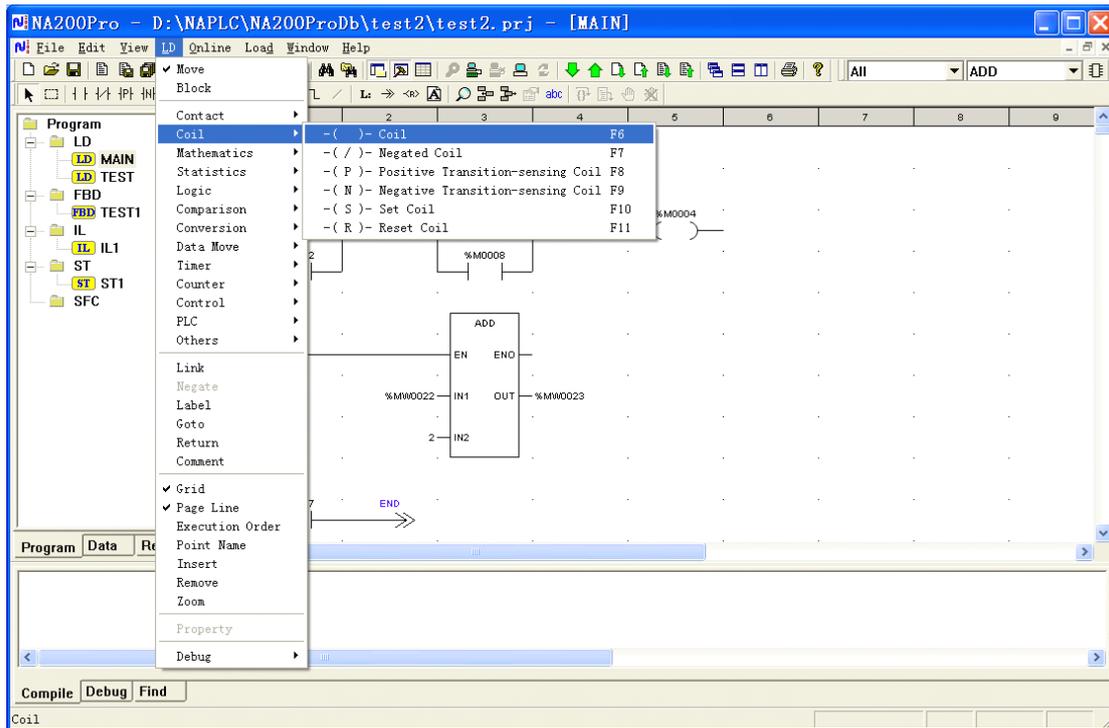


Fig.2.4 Sub-menu

### 2.2.3 Shortcut menu or pop-up menu

Click the object by right mouse button to open the shortcut menu; if selecting multiple objects, the shortcut menu also can be called, and in this case, the menu only contains the effective menu items suitable for all objects. Click any place outside the menu or press the **ESC** key to turn off the menu, as shown in Fig.2.5:

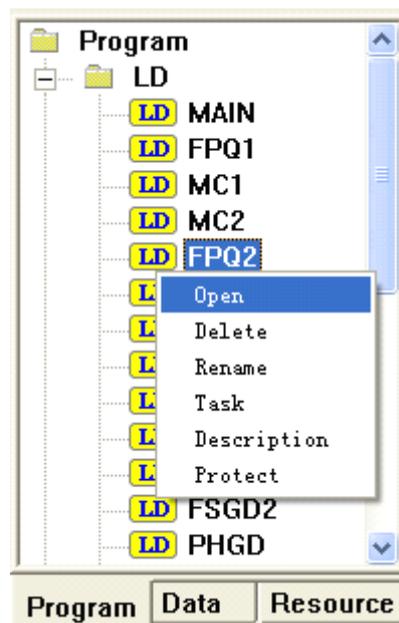


Fig.2.5 Shortcut menu or pop-up menu

## 2.3 Menu function

### 2.3.1 File

#### Composition of File menu

The **File** menu is used to manage the files, whose drop-down menu mainly includes **New**, **Open**, **Save**, **New Program**, **Save Program**, **Save All Programs**, **Compile Program**, **Compile All Programs**, **Password**, **Used Times**, **Export Point**, **Import Point**, **Print**, **Print Preview**, **Print Setup**, and **Exit**, etc., as shown in Fig.2.6:

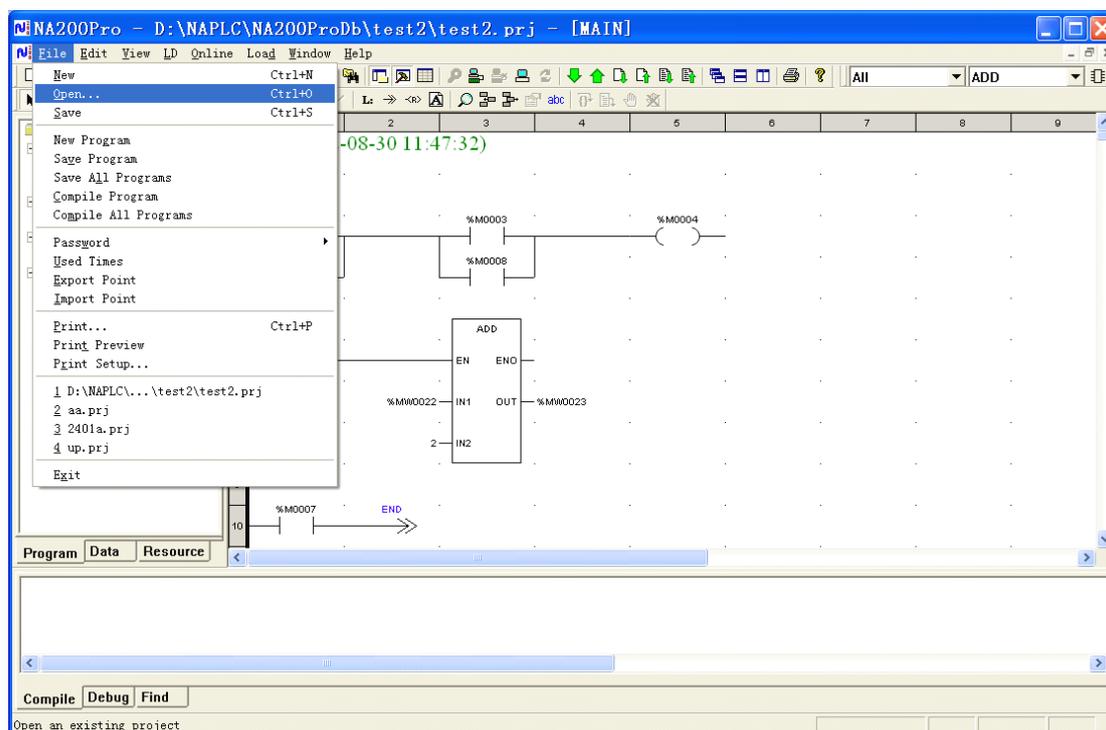


Fig.2.6 **File** menu

#### Functions of File menu

**【 New 】** : Create a new project file, which contains database, LD, FBD, IL and ST programs, etc.

**【 Open 】** : Open an existed project file. Choose **Open**, the programming software will pop up the **Open** dialog box, the file type is “NA200Pro Files”, the extension name is “prj”. If the file password is set, the **Password** dialog box will pop up, as shown in Fig.2.7.

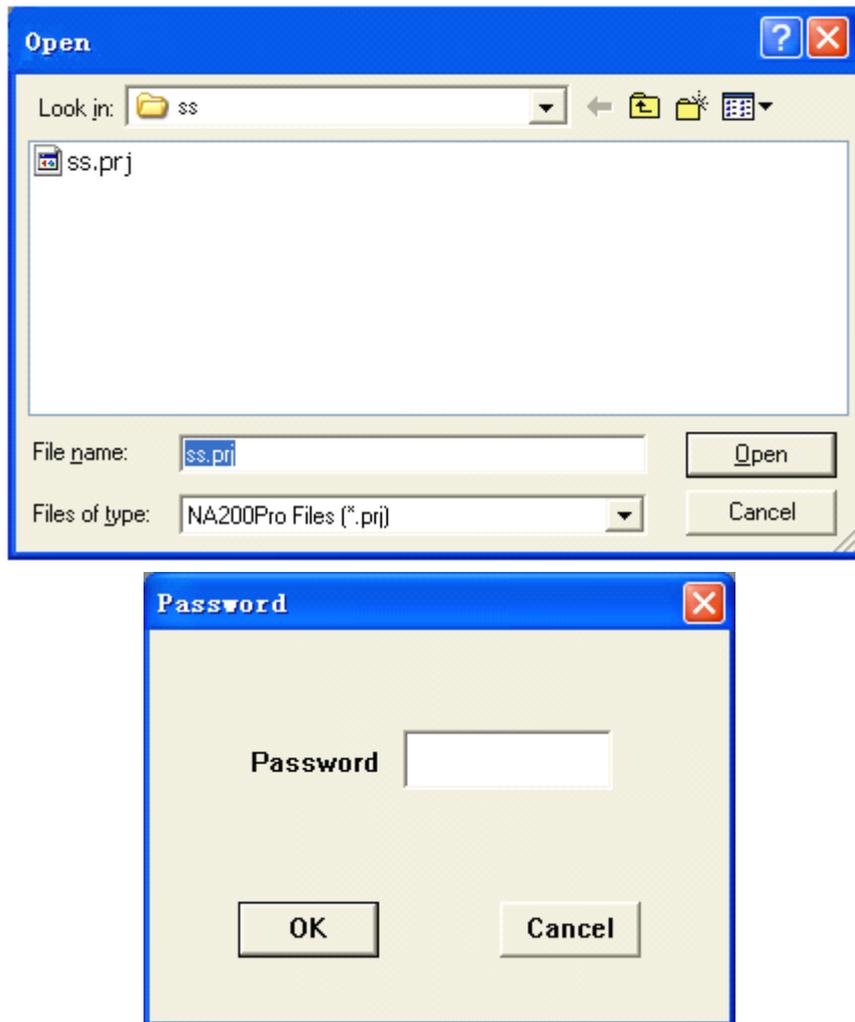


Fig.2.7 Open project

**【Save】** : Save the being edited project file. If the being edited file is an existed file, then directly overwrite the original file; if the being edited file is a new file, the programming software will pop up the **Save As** dialog box requiring to input the file name, as shown in Fig.2.8:

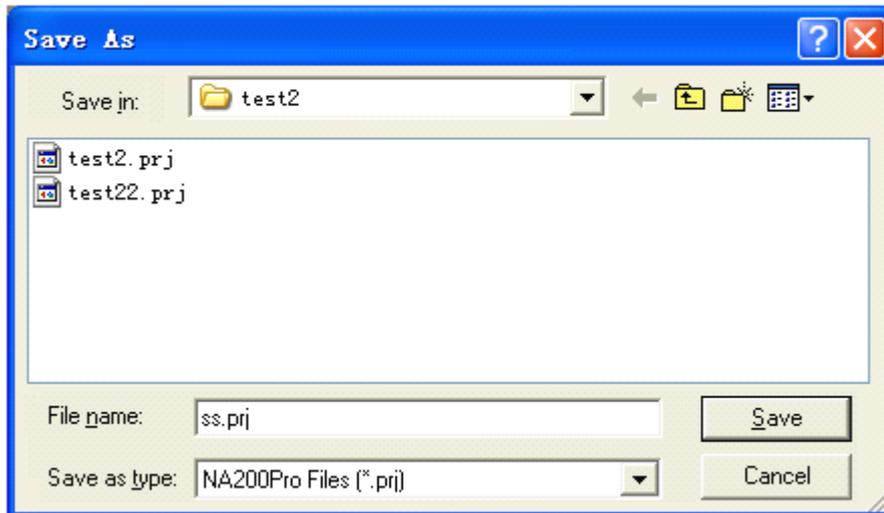


Fig.2.8 Save project

※ **Modification of project file will affect each program, please do Compile All Programs before download.**

**【New Program】** : Create a new program. The program can be LD, FBD, IL, or ST etc. The new program shall be named, and can be described, the description content will be displayed in program edit area, as shown in Fig.2.9.



Fig.2.9 New program

**【 Save Program 】** : Save the being edited program. Choose **Save Program**, the programming software will save the program automatically, whether the being edited file is a new file or an existed file.

**【 Save All Programs 】** : Save all of the programs having been opened.

**【 Compile Program 】** : Compile the current program. If the program is changed, the programming software will prompt to save or not. Then automatically check the error in the current program, if there is wrong, it can't be compiled, at the same time, the **Compile** tab of output window points out the error place, error type and the number of errors, as shown in Fig.2.10:

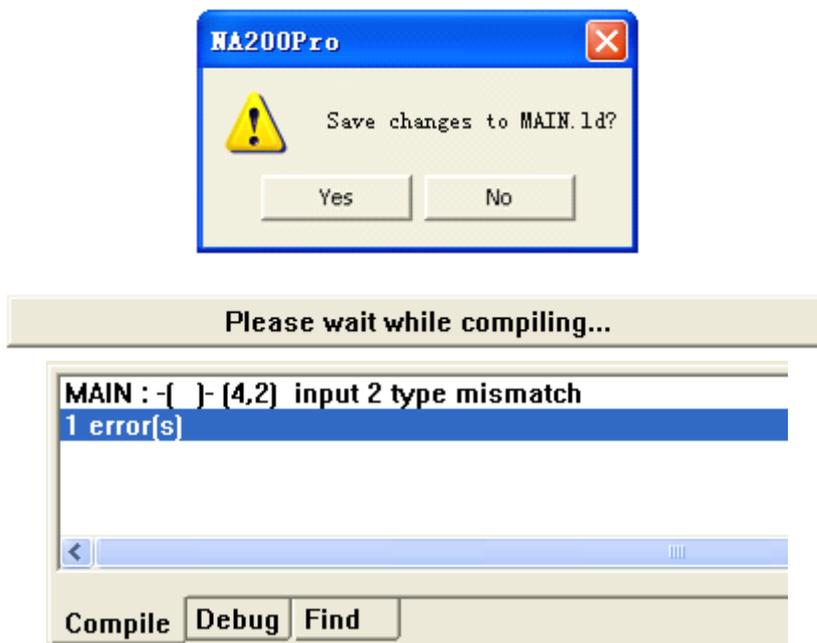


Fig.2.10 Compile program

**【 Compile All Programs 】** : Compile all of the programs of current project. It is same as **Compile Program**, the programming software will prompt to save or not (if any change). Then automatically check the error in all of the programs, if there is wrong, it can't be compiled, at the same time, the error place, error type and the number of errors can be pointed out.

**【 Password 】** : Change the file password and login password. The new project has no password, as shown in Fig.2.11:



Fig.2.11 Password

**【 Used Times 】** : Statistically calculate the used times of each point in programs. As shown in Fig.2.12:

Number	Name	Description	Used Times	Value				
SM0001			1					
SM0002			2					
SM0003			2					
SM0004			1					
SM0005			1					
SM0006			1					

Fig.2.12 Used times

**【 Print 】** : Print the PLC configuration, point information, LD, FBD, IL, and ST. When choose **Print**, pop up the standard **Print** dialog box where the printer, print range and copies etc. can be chosen. As shown in Fig.2.13.

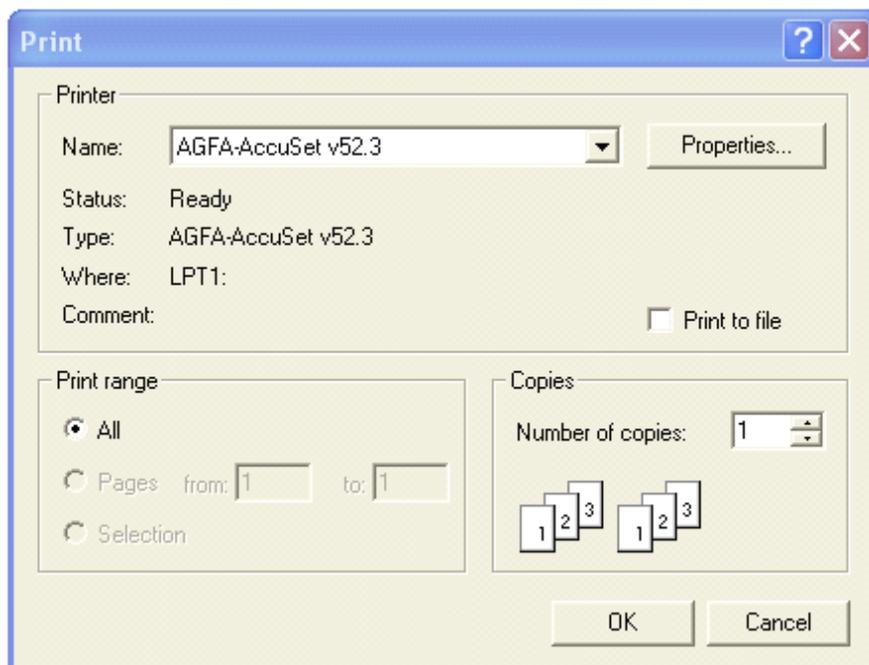


Fig.2.13 Print

**【 Print Preview 】**: Preview the printing results of PLC configuration, point information, LD, FBD, IL, and ST.

**【 Print Setup 】** : Reset the print options. Choose **Print Setup**, then pop up the **Print Setup** dialog box. For beautiful looking, the orientation of printing paper for LD and FBD is generally set to be landscape, however, for other programs, it shall be set to be portrait. As shown in Fig.2.14:

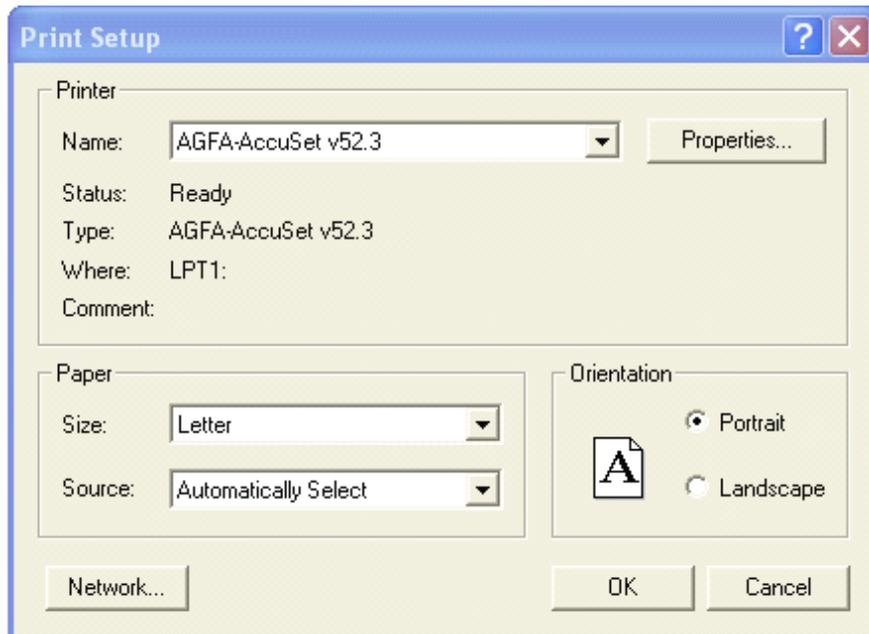


Fig.2.14 Print setup

**【 Exit 】** : Exit the NA200Pro programming software.

Moreover, the **File** menu also has the several project files recently been accessed. The files can be directly opened by the left mouse button click. As shown in Fig.2.15:

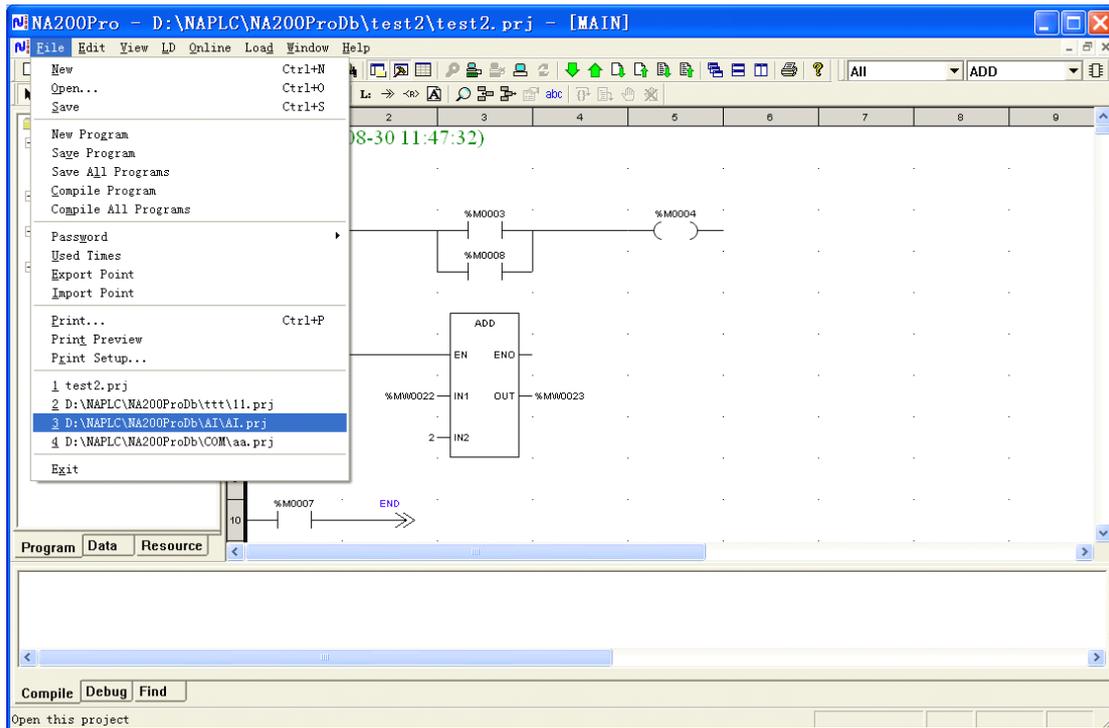


Fig.2.15 Open the recently accessed file

## 2.3.2 Edit

### Composition of Edit menu

In the **Edit** menu, there are some functions often used while editing program, it mainly includes the menu item of **Undo**, **Redo**, **Cut**, **Copy**, **Paste**, **Delete**, **Select All**, **Find**, **Replace**, and **Global Find** etc., as shown in Fig.2.16:

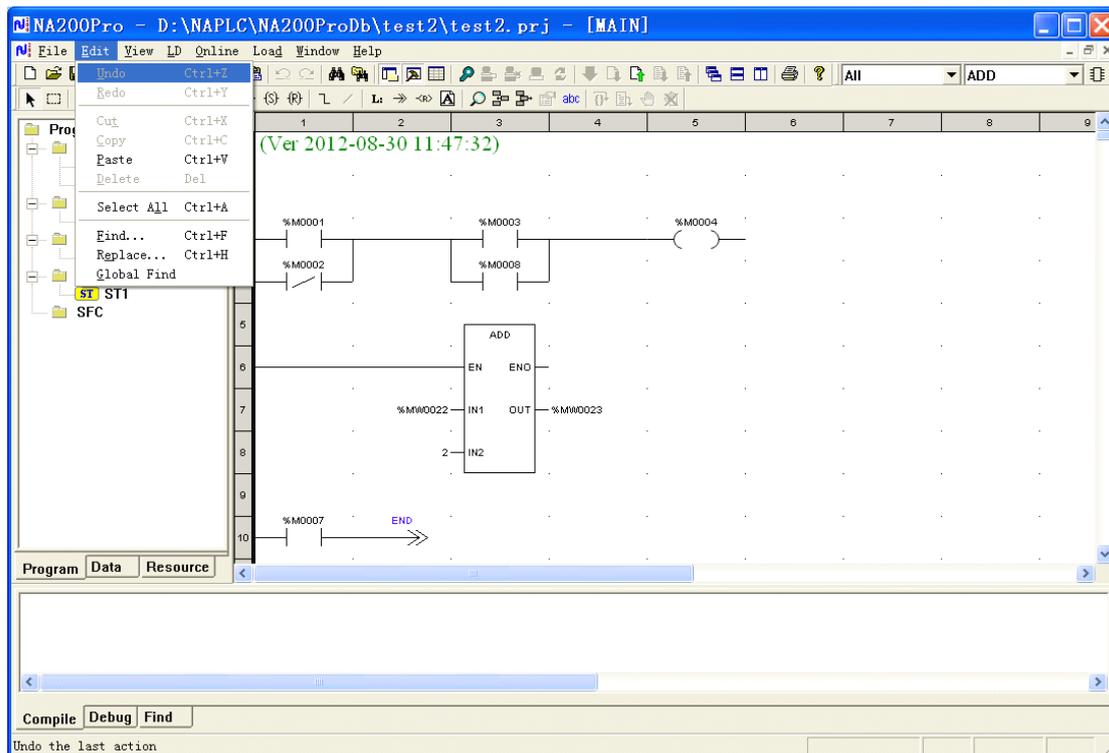


Fig.2.16 Edit menu

### Functions of Edit menu

**【Undo】** : Cancel the last operation. No matter what the last operation is to place/delete function block, paste/cut a section of program, move the place of function block etc., by **Undo** operation, it can be canceled. Changing the parameters of function block is not deemed as an operation in programming software, so it can't be canceled.

**【Redo】** : Redo the last operation. Redo is only used for just undo operation, which means no canceling.

**【Cut】** : Delete the current selected content, and place it into clipboard, the content in the clipboard can be pasted. The selected content can be a function block (contact, coil and special function block) in LD and FBD, an instruction in IL or a statement in ST, also can be the content of an area selected by block operation.

**【Copy】** : Place the current selected content into clipboard but do not delete it.

**【Paste】** : Place the content in the clipboard at the mouse clicking place.

**【Delete】** : Delete the current selected content but do not place it into clipboard.

Note: The operations of **Cut**, **Copy**, **Paste** and **Delete** also can be achieved by clicking the editing area with the right mouse button. Select the function block, the **Cut**, **Copy**, and **Delete** operations can be executed, and the **Paste** operation can be executed at the blank. As shown in Fig.2.17:

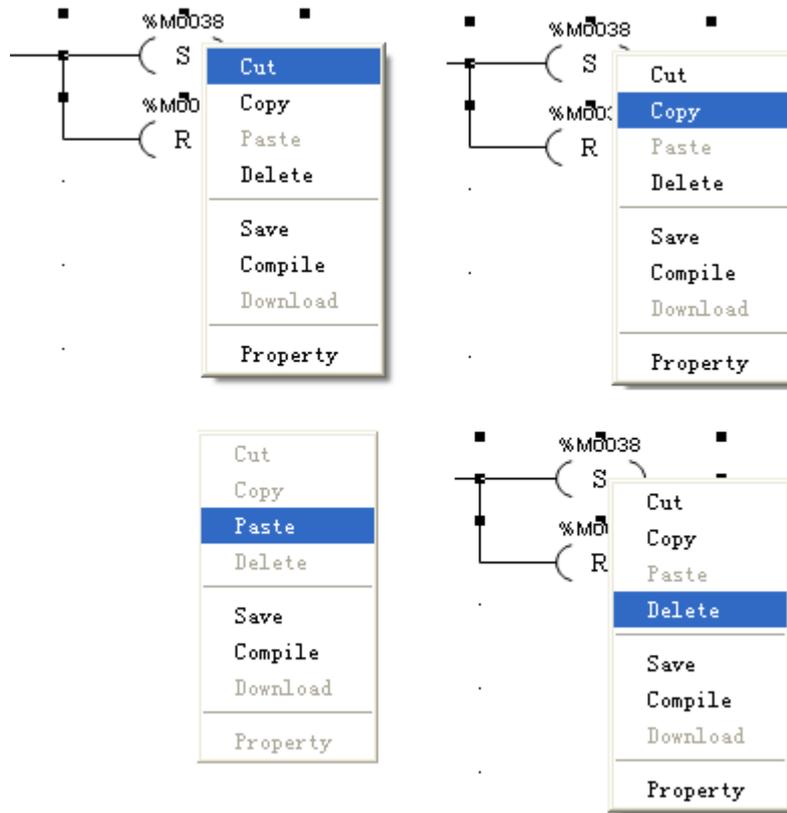


Fig.2.17 Cut, copy, paste, and delete operation

**【Select All】** : Select all of the contents in the current editing area. As shown in Fig.2.18:

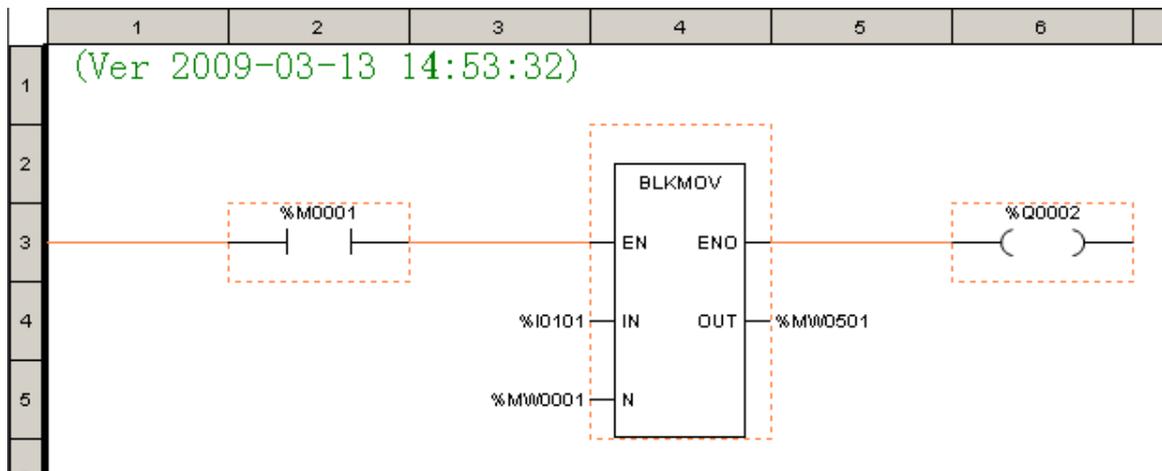


Fig.2.18 Select all operation

**【Find】**: Find the suitable function block, instruction or statement. The **Find** operation can be only executed in the current work area.

If the current work area is LD (or FBD), then find it in the current LD (or FBD). For example, if want to find the function block with the parameter “%M0036”, select **【Edit】 / 【Find】** , then pop up the **Find** dialog box, input “%M0036” to “Find What”, and “All” to “Find Limit”,

then click the **Find Next** button, then automatically jump to the first suitable function block, and display this function block with a virtual box. If continuously want to find other suitable function blocks, only need to repeatedly click the **Find Next** button, then jump to the next function block with “%M0036” in turn till find all. Finally the programming software will prompt “Finished”. The **Find** function can be executed in all of the function blocks, and also in the specified type of function blocks. For example, set “Find Limit” to be “Normally Open Contact”, then the suitable function blocks will only be found in the normally open contacts. As shown in Fig.2.19:

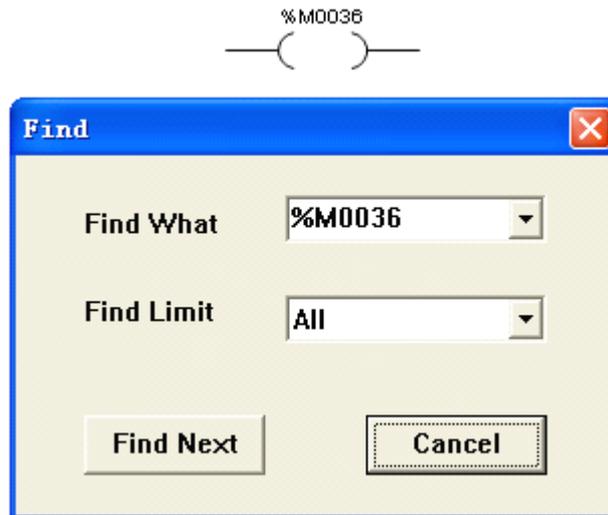


Fig.2.19 Find operation in LD

Till find all, then the programming software will pop up “Finished” prompt box. As shown in Fig.2.20:

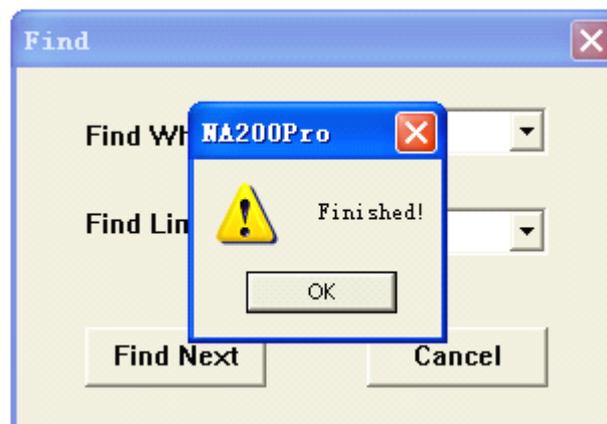


Fig.2.20 Find Finished

If the current work area is IL, then find it in the current IL. There are the alternatives of “Match Whole Word Only” and “Match Case” in IL **Find** dialog box. For example, to find “R1”, if input “r1” into “Find What”, and do not select “Match Case”, then can’t find it. As shown in Fig.2.21:

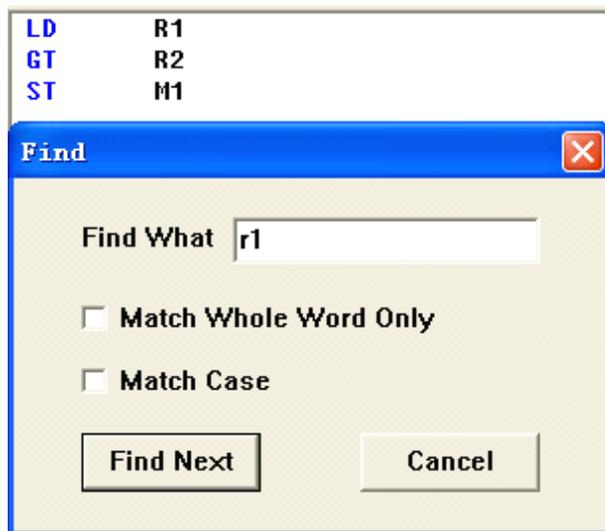


Fig.2.21 Find operation in IL

If the current work area is ST, then find it in the current ST. There are the alternatives of “Match Whole Word Only” and “Match Case” in ST **Find** dialog box. For example, to find “R1”, if input “r1” into “Find What”, and do not select “Match Case”, then can’t find it. As shown in Fig.2.22:

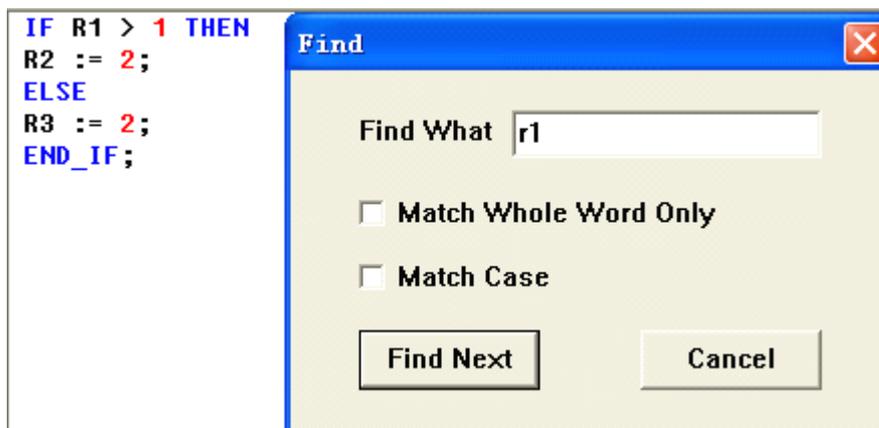


Fig.2.22 Find operation in ST

**【 Replace 】** : Find the suitable function block, instruction or statement in the current program and replace it. The **Replace** operation is similar to the **Find** operation, only add a replace function, it can be said that the **Replace** operation contains the **Find** operation. For the **Replace** operation, the selective replacement can be executed, and also the replacement all can be executed. The replacement only can be executed in the current work area.

**【 Global Find 】** : Find the suitable function block, instruction or statement globally. This function is executed in all programs (LD, FBD, IL and ST etc.). After the completion of **Global Find**, the result displays in the **Find** tab of Output Window, including the place and the number. As shown in Fig.2.23.

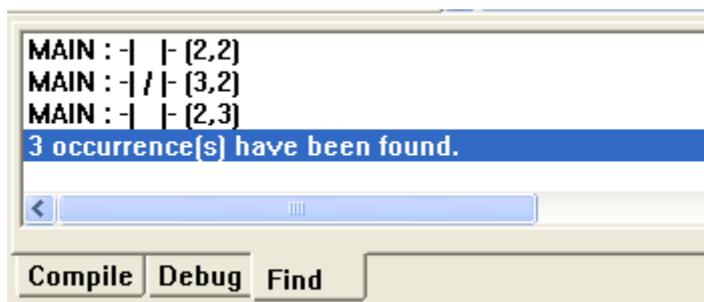


Fig.2.23 Global find

### 2.3.3 View

#### Composition of View menu

The **View** menu mainly contains: **System Toolbar**, **FB Toolbar**, **LD Toolbar**, **FBD Toolbar**, **IL Toolbar**, **ST Toolbar**, **Project Browser**, **Output Window**, **Point Table**, and **Status Bar** etc., as shown in Fig.2.24:

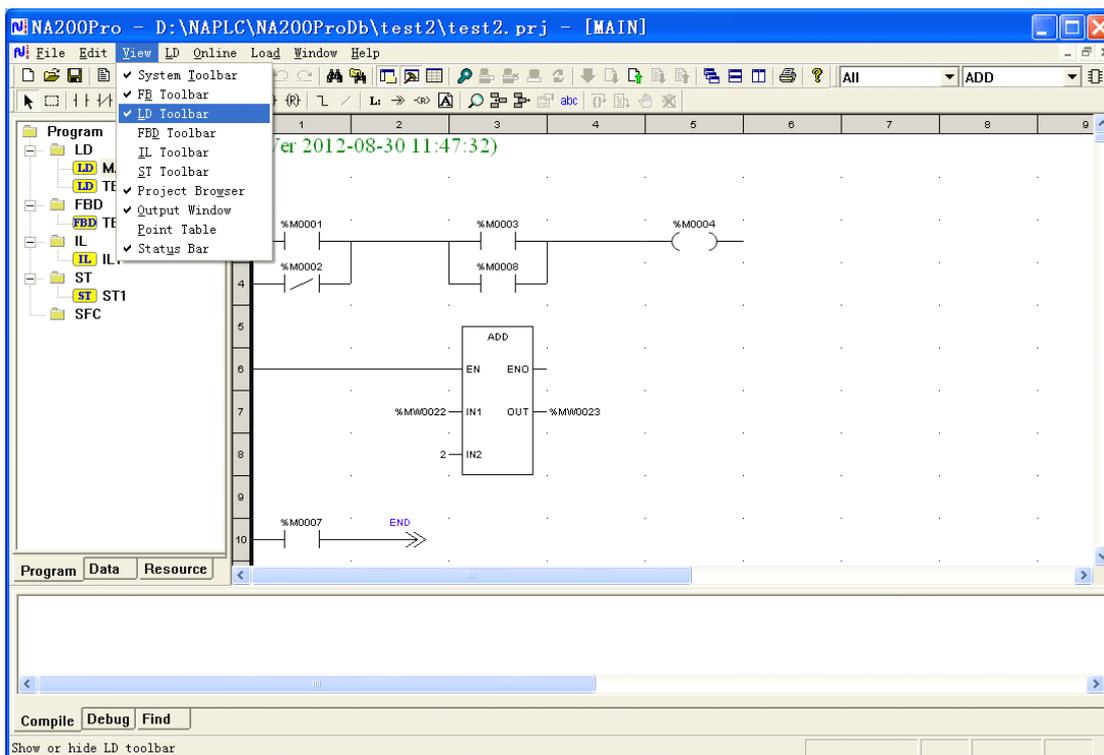


Fig.2.24 **View** menu

#### Functions of View menu

- 【System Toolbar】** : Show or hide the system toolbar.
- 【FB Toolbar】** : Show or hide the FB toolbar.
- 【LD Toolbar】** : Show or hide the LD toolbar.

**【FBD Toolbar】** : Show or hide the FBD toolbar.

**【IL Toolbar】** : Show or hide the IL toolbar.

**【ST Toolbar】** : Show or hide the ST toolbar.

**【Project Browser】** : Show or hide the project browser.

**【Output Window】** : Show or hide the output window.

**【Point Table】** : Show or hide the point table.

**【Status Bar】** : Show or hide the status bar.

## 2.3.4 LD

### Composition of LD menu

The **LD** menu contains: **Move**, **Block**, **Contact**, **Coil**, **Mathematics**, **Statistics**, **Logic**, **Comparison**, **Conversion**, **Data Move**, **Timer**, **Counter**, **Control**, **PLC**, **Others**, **Link**, **Negate**, **Label**, **Goto**, **Return**, **Comment**, **Grid**, **Page Line**, **Execution Order**, **Point Name**, **Insert**, **Remove**, **Zoom**, **Property**, and **Debug** etc., as shown in Fig.2.25:

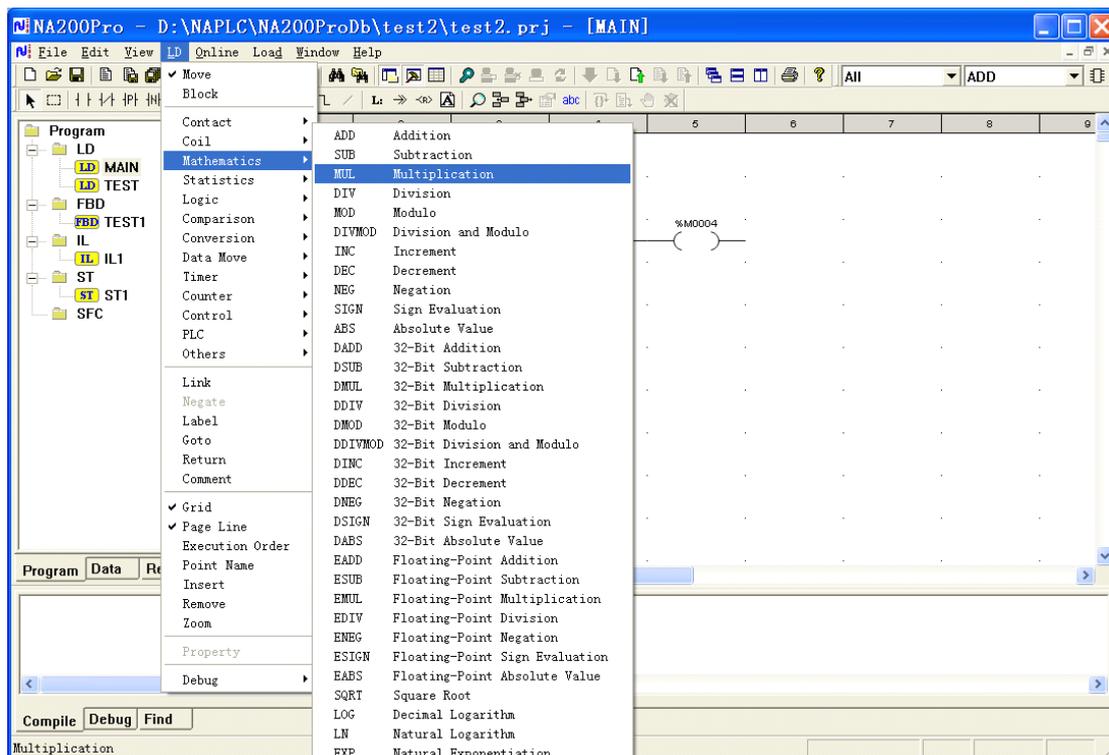


Fig.2.25 LD menu

### Functions of LD menu

**【Move】** : When any FB in **LD** menu is not selected, the editing area is always at move state. The currently selected FB position can be changed with the mouse move. When moving FB, hold the left mouse button, and move to the specified position then release it.

**【Block】** : **Block** operation is used to select all of the FB and links of one area. Use the mouse to delineate the desired area, and then all the elements in that area are chosen.

The operations of move, cut, copy, and delete etc. of multiple elements must be completed by **Block** operation.

**【Link】** : **Link** operation is used to place a current path between two FB pins. Move the mouse to the first parameter pin need to be linked, and click the left mouse button to select the first parameter pin; then move the mouse to the second parameter pin, and click the left mouse button, in this way, a link occurs between both of the parameter pins. If two of the selected parameter pins do not conform to the link principle, then a prompt dialog box will pop up as shown in Fig.2.26:



Fig.2.26 Link error

**【Negate】** : When the selected function block is a contact, the **Negate** operation can be cyclically switched among normally open contact, normally close contact, positive transition-sensing contact, and negative transition-sensing contact.

When the selected function block is a coil, the **Negate** operation can be cyclically switched among coil, negated coil, positive transition-sensing coil, negative transition-sensing coil, set coil and reset coil.

**【Grid】** : The point-like grid may be set on the background of LD editing area so as to easy to be visually observed.

**【Page Line】** : The page line and page number are set on the background of LD editing area so as to easy to be visually observed.

**【Execution Order】** : Mark out the execution orders of all function blocks at scanning state. As shown in Fig.2.27.

**【Point Name】** : Mark out the point names of all function blocks. Each point can be defined a name in the point table, when the point name is allowed to display, the defined name will also display at the bottom of function block. As shown in Fig.2.27:

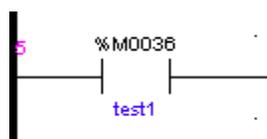


Fig.2.27 Execution order and point name

**【Insert】** : Determine the place to be inserted, click the left mouse button at this place, and select **Insert** to add one line. It is mainly used when it is necessary to add program

segment between existing segments during editing the LD program.

**【Remove】** : Determine the place to be removed, click the left mouse button at this place, and select **Remove** to delete one line. It is mainly used when it is necessary to delete the blank program segment during editing the LD program, and there shall be no any function block or link at the place where the line is to be deleted.

**【Zoom】** : **Zoom** operation can be used to change the display ratio of LD editing area. Select **Zoom**, then the programming software will pop up the **Zoom** dialog box. Directly fill the input box with the desired ratio, or click the up and down buttons beside input box, or drag the cursor to achieve the ratio, the input box will always display the set ratio. As shown in Fig.2.28:

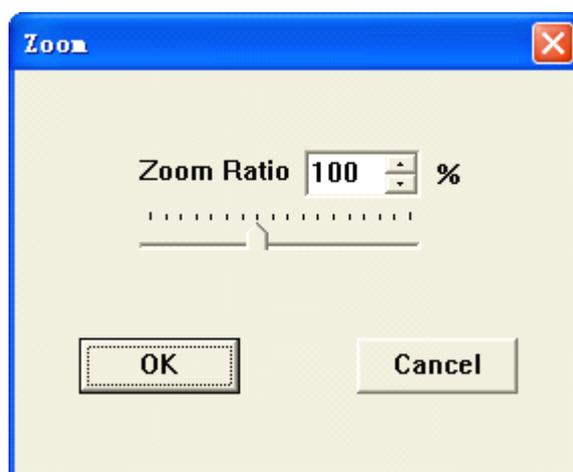


Fig.2.28 Zoom

**【Property】** : Display the selected object property.

For the LD function block, its properties include the type, program name, execution order, position, and input/output parameter definition etc. The function block type is displayed at the top of dialog box. “ADD” function block is shown as follows: the program name of LD which this function block is in is “MAIN”; The execution order is the 8<sup>th</sup>; The position is at the third column and the 5<sup>th</sup> row; The “Display EN/ENO” parameter in FBD editor can be used as an alternative; The “Input Number” parameter can be modified in some function blocks of LD and FBD; The parameter of each input/output pin is listed in sequential. As shown in Fig.2.29:

**FB Property** [Close]

**ADD Addition**

Program Name:  Execution Order:

Start Position:   Display EN/ENO

Input Number:

Input 1: <input type="text" value="T"/>	Output 1: <input type="text"/>
Input 2: <input type="text" value="%MW0022"/>	Output 2: <input type="text" value="%MW0023"/>
Input 3: <input type="text" value="2"/>	Output 3: <input type="text"/>
Input 4: <input type="text"/>	Output 4: <input type="text"/>
Input 5: <input type="text"/>	Output 5: <input type="text"/>
Input 6: <input type="text"/>	Output 6: <input type="text"/>
Input 7: <input type="text"/>	Output 7: <input type="text"/>
Input 8: <input type="text"/>	Output 8: <input type="text"/>
Input 9: <input type="text"/>	Output 9: <input type="text"/>
Input 10: <input type="text"/>	Output 10: <input type="text"/>
Input 11: <input type="text"/>	Output 11: <input type="text"/>
Input 12: <input type="text"/>	Output 12: <input type="text"/>
Input 13: <input type="text"/>	Output 13: <input type="text"/>
Input 14: <input type="text"/>	Output 14: <input type="text"/>
Input 15: <input type="text"/>	Output 15: <input type="text"/>
Input 16: <input type="text"/>	Output 16: <input type="text"/>

OK Cancel

Fig.2.29 Function block property

**【 Debug 】** : Debug the LD program; includes **Step**, **Continue**, **Insert/Remove Breakpoint** and **Remove All Breakpoints**, etc.

## 2.3.5 Online

### Composition of Online menu

The **Online** menu mainly contains: **Login**, **Logout**, **Select COM**, **Connect/PLC**, **Connect/Simulator**, **Disconnect**, **Display Format**, **Refresh Program**, **Unforce**, **Reset**, **Set Time**, and **Refresh FLASH** etc., as shown in Fig.2.30:

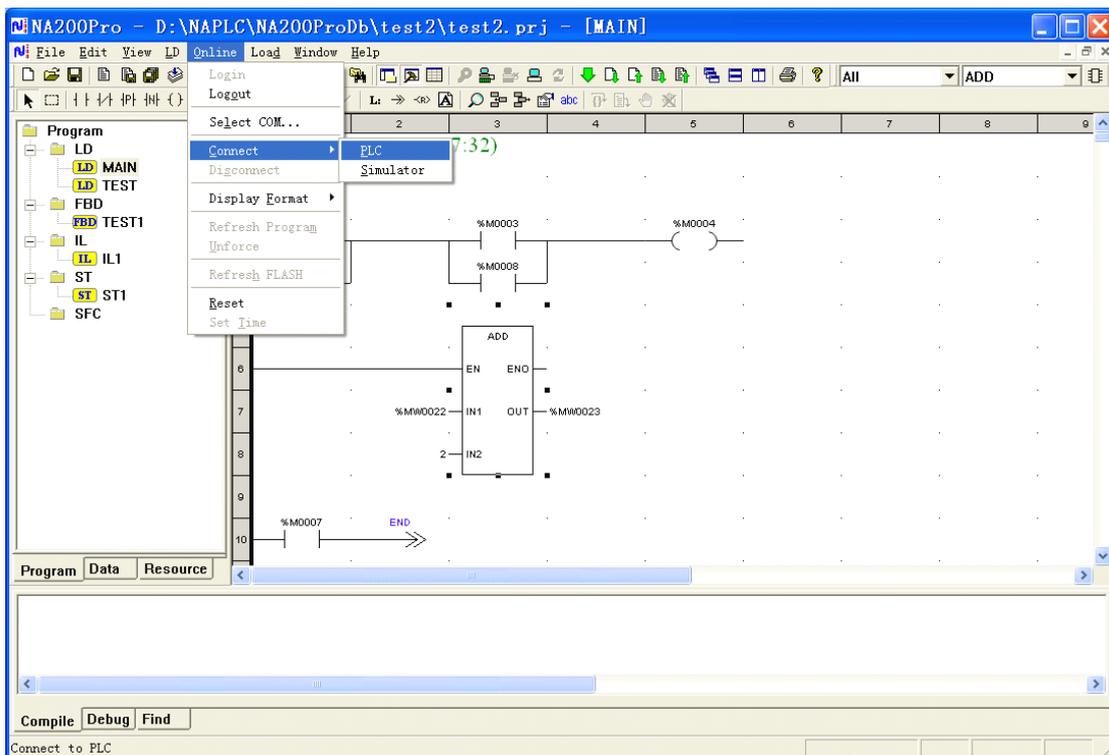


Fig.2.30 **Online** menu

### Functions of Online menu

**【Login】**: The **Connect**, **Upload**, and **Download** operations can be executed after login.

The initial login password of a new project is null, as shown in Fig.2.31:



Fig.2.31 Login

**【Logout】** : Cancel login. At this time, **Connect**, **Upload** and **Download** operations can't be executed.

**【Connect】 / 【PLC】** : Connect the current debugging computer with PLC. Before connecting, please make sure the serial port has been physically connected. Otherwise, the programming software will pop up a prompt dialog box as shown in Fig.2.32:



Fig.2.32 Connection failure with PLC

### The appearance after connecting successfully

After connecting successfully, the background color of program editing area turns lavender. The data information in PLC are sent into the programming software by serial port, the parameters with the current value of 1 and the conducted links are displayed in red, and the parameters with the current value of 0 and the un-conducted links are displayed in green. As shown in Fig.2.33:

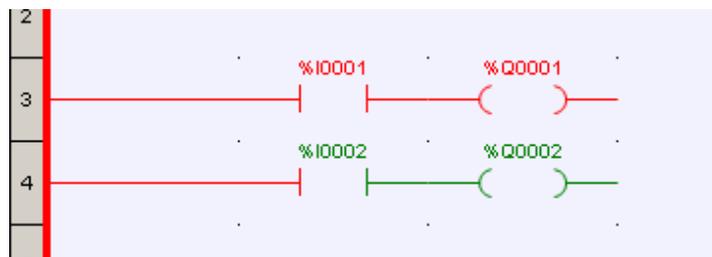


Fig.2.33 LD at online state

**【Connect】 / 【Simulator】** : It can simulate the PLC online state, force the actual point values, and debug the programs.

**【Disconnect】** : Turn off the connection of debugging computer with PLC or simulator.

**【Display Format】** : It indicates the display formats of integer data (non-BOOL type) at online mode. There are three kinds of display formats to be selected, respectively are decimal, hexadecimal and binary. The currently selected display format has a  $\surd$  mark. For example, the current “%MW0001” register is decimal 2002, the three kinds of display formats in point table are shown as follows. The following values all represent 2002 by different display formats, the hexadecimal and binary data are respectively followed by the letter “H” and “B” which represents the current display format. As shown in Fig.2.34:

Value	Value	Value
2002	07D2H	0000011111010010B
Decimal	Hexadecimal	Binary

Fig.2.34 Display format

**【 Refresh Program 】** : It provides a convenient online debugging to modify the programs. At online mode, if add or delete or move the function blocks, or modify the function block parameter, there will be a “\*” label on the right of program name in project browser, such as “MAIN\*”. At this time, it is necessary to select **Refresh Program** to download the modifications into PLC. After modifying it and before exiting the programming software, do save the modified programs, otherwise, it will result in the program inconsistency. Before refreshing, the programming software will automatically compile the program, if any mistake, the program will not be refreshed, and will point out the mistake. After refreshing, the PLC will be directly executed according to the downloaded programs, the system do not need to be restarted. Because the **Refresh Program** operation only modifies the execution program, however, does not modify the source program saved in PLC, if at this time upload the program, the uploaded program will be still the old program, so it is recommended that do save the modified program after modifying, and do completely download after program determination.

**【 Unforce 】** : At online mode, there is a **Force** function on point table, the scanned signal states of digital input, digital output, analog input and analog output after **Force** operation will never be sent into the corresponding store areas, they can be set values according to the debugging requirements without considering the actual states. **Unforce** is to release the forced points, and resume back to scan.

**【 Reset 】** : Reset the CPU module of PLC. If the CPU module of PLC to be reset can not be connected with the debugging computer, the programming software will alarm “Reset failure”.

**【 Set Time 】** : At online mode, set time for PLC by serial port. However, the set time is the time in debugging computer, not the standard time.

## 2.3.6 Download

Refer to Chapter 3 “Project Management”.

## 2.3.7 Window

### Composition of Window menu

The **Window** menu includes **Close**, **Close All**, **Cascade**, **Tile Horizontally**, and **Tile Vertically** etc. As shown in Fig.2.35:

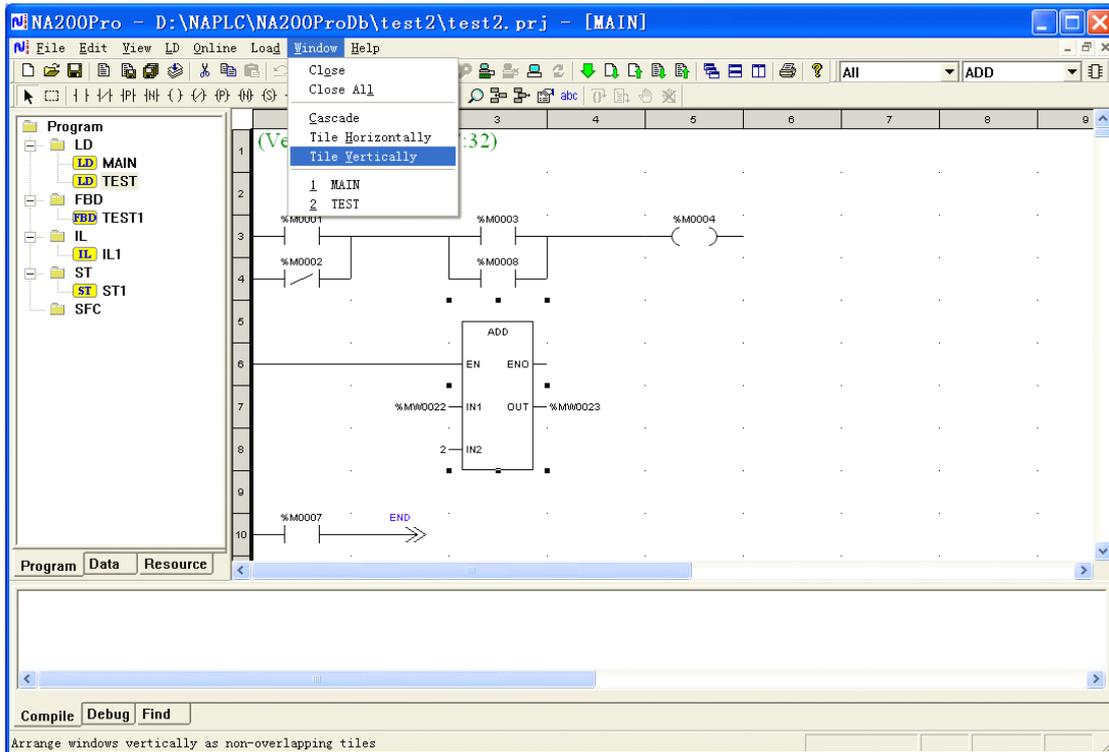


Fig.2.35 **Window** menu

## 2.3.8 Help

### Composition of Help menu

The **Help** menu includes **Contents**, **Index**, **Search**, **About NA200Pro** etc. As shown in Fig.2.36:

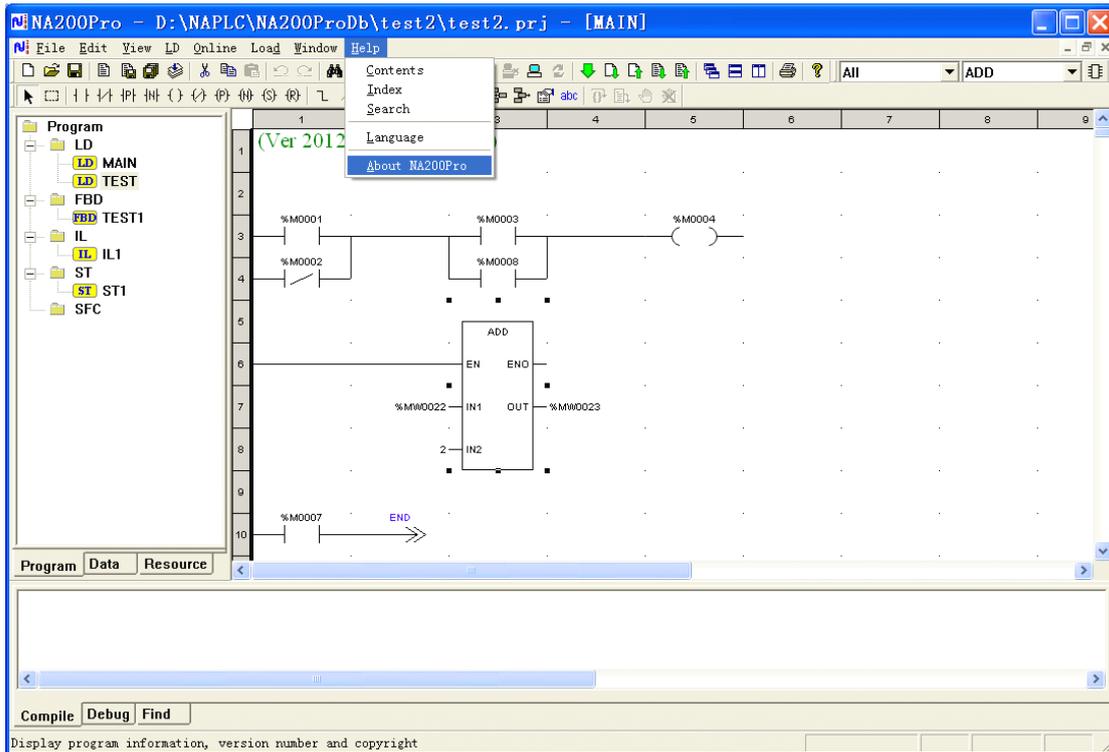


Fig.2.36 Help menu

### Functions of Help menu

**【Contents】** : Display the content property tab of help.

**【Index】** : Display the index property tab of help.

**【Search】** : Display the search property tab of help.

**【About NA200Pro】** : Display the version and copyright of programming software. As shown in Fig.2.37:

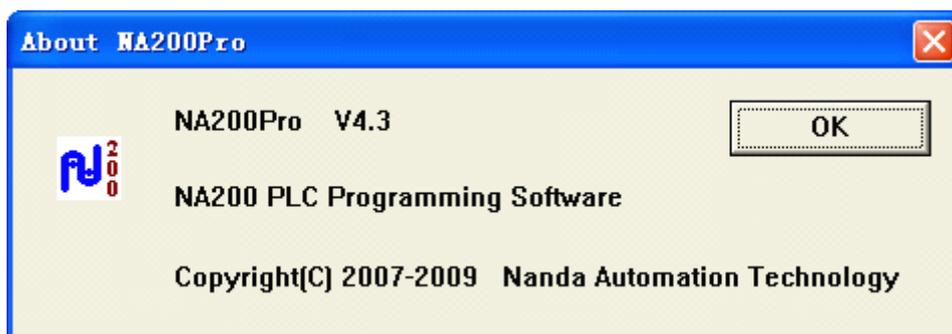


Fig.2.37 NA200Pro version information

## 2.4 System toolbar

For convenience, the system toolbar can place some functions by icons often used during the operations or editing processes on the top of editing area. All the functions can be

achieved by menu operations, so we just introduce the corresponding menu operations, and the detailed functions can be found in the introductions of menu.

- : **New**, corresponds to the menu operation **【File】 / 【New】** .
- : **Open**, corresponds to the menu operation **【File】 / 【Open】** .
- : **Save**, corresponds to the menu operation **【File】 / 【Save】** .
- : **New Program**, corresponds to the menu operation **【File】 / 【New Program】** .
- : **Save Program**, corresponds to the menu operation **【File】 / 【Save Program】** .
- : **Save All Programs**, corresponds to the menu operation **【File】 / 【Save All Programs】** .
- : **Compile Program**, corresponds to the menu operation **【File】 / 【Compile Program】**.
- : **Cut**, corresponds to the menu operation **【Edit】 / 【Cut】** .
- : **Copy**, corresponds to the menu operation **【Edit】 / 【Copy】** .
- : **Paste**, corresponds to the menu operation **【Edit】 / 【Paste】** .
- : **Undo**, corresponds to the menu operation **【Edit】 / 【Undo】** .
- : **Redo**, corresponds to the menu operation **【Edit】 / 【Redo】** .
- : **Find**, corresponds to the menu operation **【Edit】 / 【Find】** .
- : **Global Find**, corresponds to the menu operation **【Edit】 / 【Global Find】** .
- : **Project Browser**, corresponds to the menu operation **【View】 / 【Project Browser】** .
- : **Output Window**, corresponds to the menu operation **【View】 / 【Output Window】** .
- : **Point Table**, corresponds to the menu operation **【View】 / 【Point Table】** .
- : **Login**, corresponds to the menu operation **【Online】 / 【Login】** .
- : **PLC Connect**, corresponds to the menu operation **【Online】 / 【Connect】 / 【PLC】**.
- : **Disconnect**, corresponds to the menu operation **【Online】 / 【Disconnect】** .

: **Simulator Connect**, corresponds to the menu operation **【Online】 / 【Connect】 / 【Simulator】** .

: **Refresh Program**, corresponds to the menu operation **【Online】 / 【Refresh Program】** .

: **Download All**, corresponds to the menu operation **【Load】 / 【Download All】** .

: **Download Project**, corresponds to the menu operation **【Load】 / 【Download Project】** .

: **Upload Project**, corresponds to the menu operation **【Load】 / 【Upload Project】** .

: **Download Program**, corresponds to the menu operation **【Load】 / 【Download Program】** .

: **Upload Program**, corresponds to the menu operation **【Load】 / 【Upload Program】** .

: **Print**, corresponds to the menu operation **【File】 / 【Print】** .

: **About**, corresponds to the menu operation **【Help】 / 【About NA200Pro】** .

## 2.5 LD toolbar

For convenience during editing LD programs, the LD toolbar can place parts of operations and function blocks in **LD** menu by icons on the top of editing area.

: **Move**, corresponds to the menu operation **【LD】 / 【Move】** .

: **Block**, corresponds to the menu operation **【LD】 / 【Block】** .

: **Normally Open Contact**, corresponds to the menu operation **【LD】 / 【Normally Open Contact】** . Shortcut: F2

: **Normally Close Contact**, corresponds to the menu operation **【LD】 / 【Normally Close Contact】** . Shortcut: F3

: **Positive Transition-sensing Contact**, corresponds to the menu operation **【LD】 / 【Positive Transition-sensing Contact】** . Shortcut: F4

: **Negative Transition-sensing Contact**, corresponds to the menu operation **【LD】 / 【Negative Transition-sensing Contact】** . Shortcut: F5

- : **Coil**, corresponds to the menu operation **【LD】 / 【Coil】** . Shortcut: F6
- : **Negated Coil**, corresponds to the menu operation **【LD】 / 【Negated Coil】** . Shortcut: F7
- : **Positive Transition-sensing Coil**, corresponds to the menu operation **【LD】 / 【Positive Transition-sensing Coil】** . Shortcut: F8
- : **Negative Transition-sensing Coil**, corresponds to the menu operation **【LD】 / 【Negative Transition-sensing Coil】** . Shortcut: F9
- : **Set Coil**, corresponds to the menu operation **【LD】 / 【Set Coil】** . Shortcut: F10
- : **Reset Coil**, corresponds to the menu operation **【LD】 / 【Reset Coil】** . Shortcut: F11
- : **Link**, corresponds to the menu operation **【LD】 / 【Link】** .
- : **Negate**, corresponds to the menu operation **【LD】 / 【Negate】** .
- : **Label**, corresponds to the menu operation **【LD】 / 【Label】** .
- : **Goto**, corresponds to the menu operation **【LD】 / 【Goto】** .
- : **Return**, corresponds to the menu operation **【LD】 / 【Return】** .
- : **Comment**, corresponds to the menu operation **【LD】 / 【Comment】** .
- : **Zoom**, corresponds to the menu operation **【LD】 / 【Zoom】** .
- : **Insert**, corresponds to the menu operation **【LD】 / 【Insert】** .
- : **Remove**, corresponds to the menu operation **【LD】 / 【Remove】** .
- : **Property**, corresponds to the menu operation **【LD】 / 【Property】** .
- : **Step**, corresponds to the menu operation **【LD】 / 【Debug】 / 【Step】** .
- : **Continue**, corresponds to the menu operation **【LD】 / 【Debug】 / 【Continue】** .
- : **Insert/Remove Breakpoint**, corresponds to the menu operation **【LD】 / 【Debug】 / 【Insert/Remove Breakpoint】** .
- : **Clear All Breakpoints**, corresponds to the menu operation **【LD】 / 【Debug】 / 【Clear All Breakpoints】** .

The types of basic function blocks are so many that you should select one function block by two drop-down combo boxes. The first drop-down combo box is used to select the group, and the second drop-down combo box is used to select the specified function block. For example, the current display group is **Mathematics**, and then all of the mathematics function blocks will be listed in the second drop-down combo box to be selected. If the icon of basic function block is pressed, it indicates that the current basic function block has been selected, and can be directly placed in the editing area. As shown in Fig.2.38:

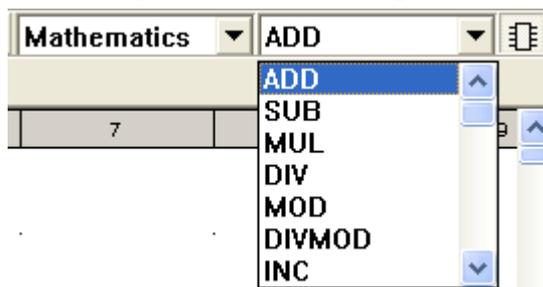


Fig.2.38 Basic function block

## 2.6 FBD toolbar

For convenience during editing FBD programs, the FBD toolbar can place parts of operations and function blocks in **FBD** menu by icons on the top of editing area.

-  : **Move**, corresponds to the menu operation **【FBD】 / 【Move】** .
-  : **Block**, corresponds to the menu operation **【FBD】 / 【Block】** .
-  : **Link**, corresponds to the menu operation **【FBD】 / 【Link】** .
-  : **Negate**, corresponds to the menu operation **【FBD】 / 【Negate】** .
-  : **Label**, corresponds to the menu operation **【FBD】 / 【Label】** .
-  : **Goto**, corresponds to the menu operation **【FBD】 / 【Goto】** .
-  : **Return**, corresponds to the menu operation **【FBD】 / 【Return】** .
-  : **Comment**, corresponds to the menu operation **【FBD】 / 【Comment】** .
-  : **Zoom**, corresponds to the menu operation **【FBD】 / 【Zoom】** .

: **Insert**, corresponds to the menu operation **【FBD】 / 【Insert】** .

: **Remove**, corresponds to the menu operation **【FBD】 / 【Remove】** .

: **Property**, corresponds to the menu operation **【FBD】 / 【Property】** .

: **Step**, corresponds to the menu operation **【FBD】 / 【Debug】 / 【Step】** .

: **Continue**, corresponds to the menu operation **【FBD】 / 【Debug】 / 【Continue】** .

: **Insert/Remove Breakpoint**, corresponds to the menu operation **【FBD】 / 【Debug】 / 【Insert/Remove Breakpoint】** .

: **Clear All Breakpoints**, corresponds to the menu operation **【FBD】 / 【Debug】 / 【Clear All Breakpoints】** .

## 2.7 IL toolbar

For convenience during editing IL programs, the IL toolbar can place parts of operations and function blocks in **IL** menu by icons on the top of editing area.

: **LD** instruction, corresponds to the menu operation **【IL】 / 【Load】 / 【LD】** .

: **LDN** instruction, corresponds to the menu operation **【IL】 / 【Load】 / 【LDN】** .

: **ST** instruction, corresponds to the menu operation **【IL】 / 【Store】 / 【ST】** .

: **STN** instruction, corresponds to the menu operation **【IL】 / 【Store】 / 【STN】** .

: **S** instruction, corresponds to the menu operation **【IL】 / 【Store】 / 【S】** .

: **R** instruction, corresponds to the menu operation **【IL】 / 【Store】 / 【R】** .

: **AND** instruction, corresponds to the menu operation **【IL】 / 【Logic】 / 【AND】** .

: **OR** instruction, corresponds to the menu operation **【IL】 / 【Logic】 / 【OR】** .

: **XOR** instruction, corresponds to the menu operation **【IL】 / 【Logic】 / 【XOR】** .

: **GT** instruction, corresponds to the menu operation **【IL】 / 【Comparison】 / 【GT】** .

: **GE** instruction, corresponds to the menu operation **【IL】 / 【Comparison】 / 【GE】** .

 **LT** instruction, corresponds to the menu operation **【IL】 / 【Comparison】 / 【LT】** .

 **LE** instruction, corresponds to the menu operation **【IL】 / 【Comparison】 / 【LE】** .

 **EQ** instruction, corresponds to the menu operation **【IL】 / 【Comparison】 / 【EQ】** .

 **NE** instruction, corresponds to the menu operation **【IL】 / 【Comparison】 / 【NE】** .

 **JMP** instruction, corresponds to the menu operation **【IL】 / 【Jump】 / 【JMP】** .

 **CAL** instruction, corresponds to the menu operation **【IL】 / 【Call】 / 【CAL】** .

 **RET** instruction, corresponds to the menu operation **【IL】 / 【Call】 / 【RET】** .

 **Step**, corresponds to the menu operation **【IL】 / 【Debug】 / 【Step】** .

 **Continue**, corresponds to the menu operation **【IL】 / 【Debug】 / 【Continue】** .

 **Insert/Remove Breakpoint**, corresponds to the menu operation **【IL】 / 【Debug】 / 【Insert/Remove Breakpoint】** .

 **Clear All Breakpoints**, corresponds to the menu operation **【IL】 / 【Debug】 / 【Clear All Breakpoints】** .

## 2.8 ST toolbar

For convenience during editing ST programs, the ST toolbar can place parts of operations and function blocks in **ST** menu by icons on the top of editing area.

 **Assignment**, corresponds to the menu operation **【ST】 / 【Operator】 / 【Assignment】** .

 **IF** statement, corresponds to the menu operation **【ST】 / 【IF】** .

 **CASE** statement, corresponds to the menu operation **【ST】 / 【CASE】** .

 **FOR** statement, corresponds to the menu operation **【ST】 / 【FOR】** .

 **WHILE** statement, corresponds to the menu operation **【ST】 / 【WHILE】** .

 **REPEAT** statement, corresponds to the menu operation **【ST】 / 【REPEAT】** .

-  : **EXIT** statement, corresponds to the menu operation **【ST】 / 【EXIT】** .
-  : **RETURN** statement, corresponds to the menu operation **【ST】 / 【RETURN】** .
-  : **Step**, corresponds to the menu operation **【ST】 / 【Debug】 / 【Step】** .
-  : **Continue**, corresponds to the menu operation **【ST】 / 【Debug】 / 【Continue】** .
-  : **Insert/Remove Breakpoint**, corresponds to the menu operation **【ST】 / 【Debug】 / 【Insert/Remove Breakpoint】** .
-  : **Clear All Breakpoints**, corresponds to the menu operation **【ST】 / 【Debug】 / 【Clear All Breakpoints】** .

## 2.9 Output window

The output window mainly includes the messages of compile result, debug status, find result, etc., so as to be convenient to compile, debug the programs and monitor the status.

**【Compile】** :It is the output window of compile result of programs.

In this window, double click the error message, the system will jump to the corresponding error location. As shown in Fig.2.39:

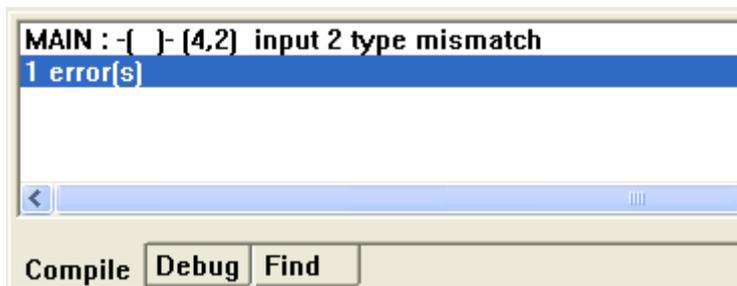


Fig.2.39 Output window

**【Debug】** : It is the output window of debug process message.

At online debugging state, the errors during the program execution process will output in this window.

**【Find】** : It is the result output window of **Global Find**. When find globally by **【Edit】 /**

**【Global Find】** or toolbar  button, the specific locations and total number of finding contents will output in this window. As shown in Fig.2.40:

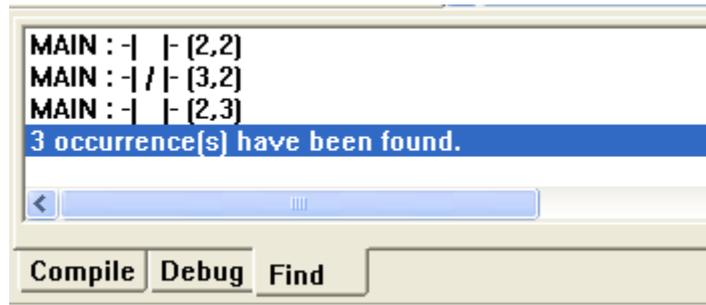


Fig.2.40 Global find result

## Chapter 3 Project Management

After entering into the NA200Pro development environment, the facing problem is to turn the actual project into the codes that can be executed in PLC. In the development environment, the project manages all of the elements, such as programs, data, and resources etc. For the technical staff, the project management includes the project configuration file and project program files. The project configuration file includes the hardware configuration, program management, PLC parameter setup etc. In the project program files, the program set of processes to be achieved is defined, including LD program, FBD program, IL program, and ST program, etc.

### 3.1 Project browser

In project browser, the NA200Pro project contents can be displayed, and contents of the project can be switched, such as program, data, and resource.

The project browser can be displayed by content tree, and can be directly navigated to: **Program**---LD, FBD, IL, and ST etc.; **Data**--- **Optional Point Table**, **FLASH Table**; **Resource**---**PLC Configuration**. As shown in Fig.3.1:

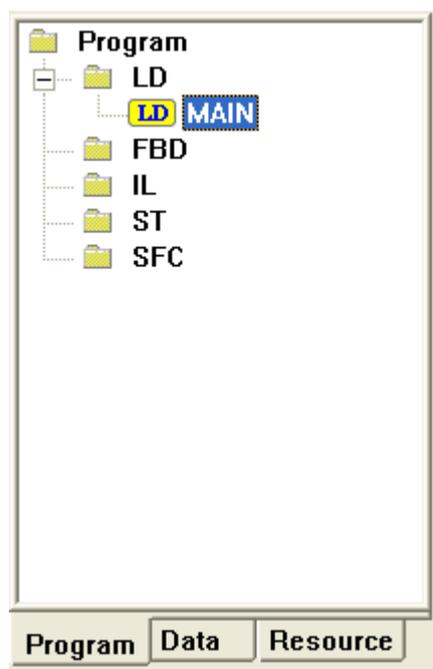


Fig.3.1 Project browser

By default, the content tree displays to the third level, double click the node to open the relative content.

## 3.2 Create new project

A new project creation includes the following several steps:

Step	Operation
1	Create a new project. See 3.2.1 “Create project”.
2	Configure PLC. See 3.2.2 “PLC hardware configuration”.
3	Create user programs. See 3.3 “Program”.
4	Save and compile programs.
5	Download project and programs. See 3.6 “Download and upload project file” and 3.7 “Download and upload program files”.

### 3.2.1 Create project

Firstly, create an empty project. Open NA200Pro programming software, click **【File】** / **【New】** on the main menu or the icon  on the system toolbar, as shown in Fig.3.2:

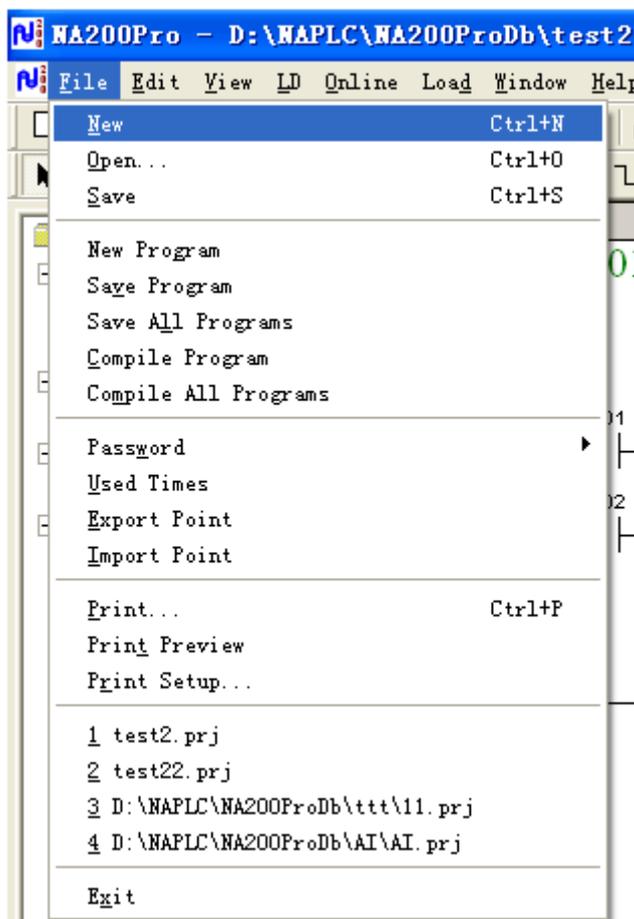


Fig.3.2 Create a new project

Click the **【File】** / **【Save】** on the main menu or the button  on the system toolbar, input

the file name of this project and save it. For example, save a project as "ss.prj". If any later modification, just open this file directly. As shown in Fig.3.3:

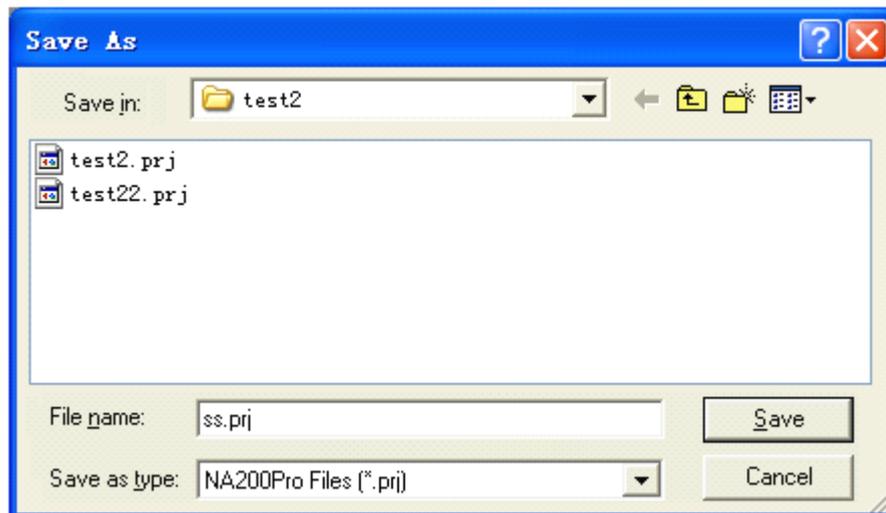


Fig.3.3 Save an existed file

NA200Pro software will pop up a warning dialog box, which point out "Please configure PLC!", choose **OK**, and then set hardware configuration, as shown in Fig.3.4:



Fig.3.4 Warning dialog box

### 3.2.2 PLC hardware configuration

In the project browser, double click **【Resource】 / 【PLC Configuration】** with the left mouse button, then PLC configuration can be done, as shown in Fig.3.5.

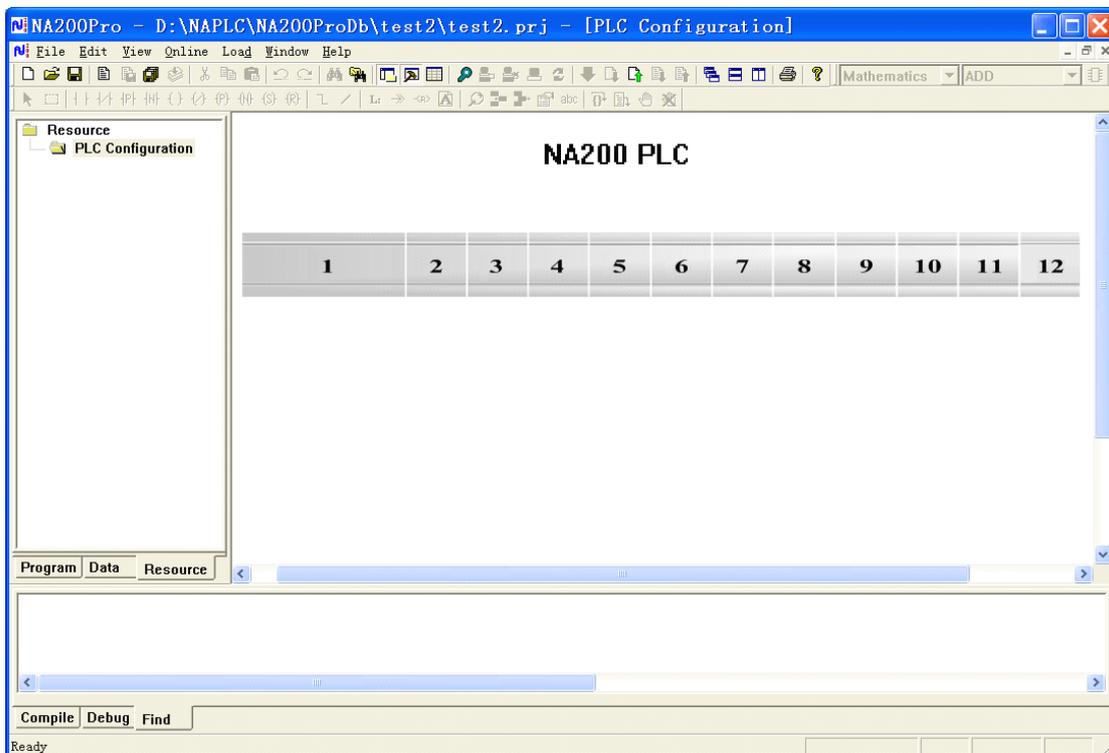
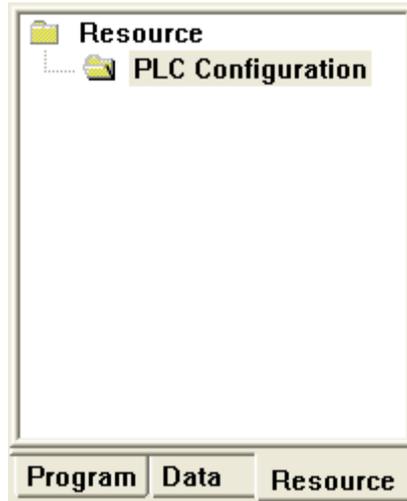


Fig.3.5 PLC configuration

Double click the module or empty slot, the **Module** dialog box will pop up, and then select the module group and module type, as shown in Fig.3.6:

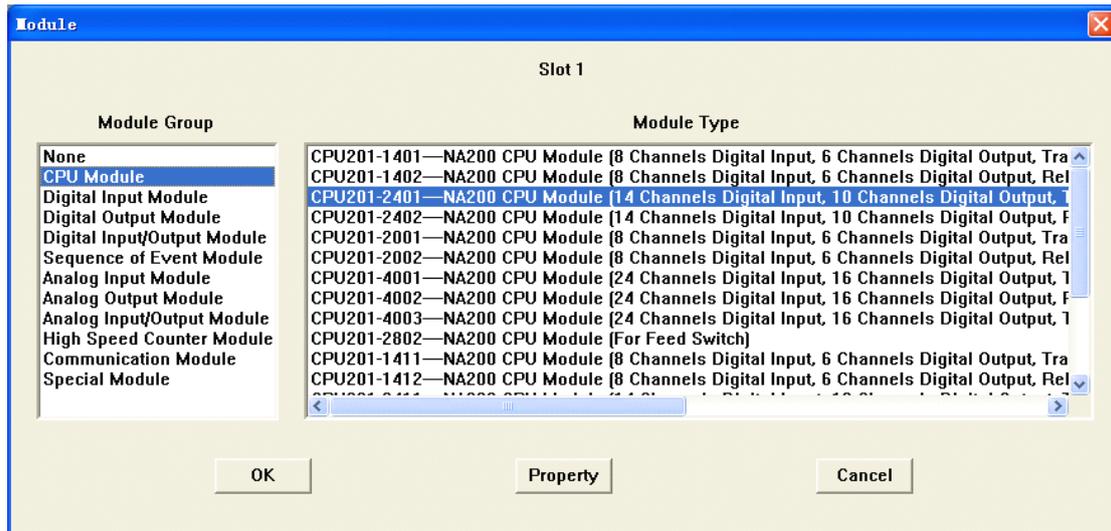


Fig.3.6 Module configuration

The module groups include the CPU, digital input, digital output, digital input/output, sequence of event, analog input, analog output, analog input/output, high speed counter, communication and special. "None" means the slot is empty, there is no module. Click the "Property" button to configure the module properties. As shown in Fig.3.7:

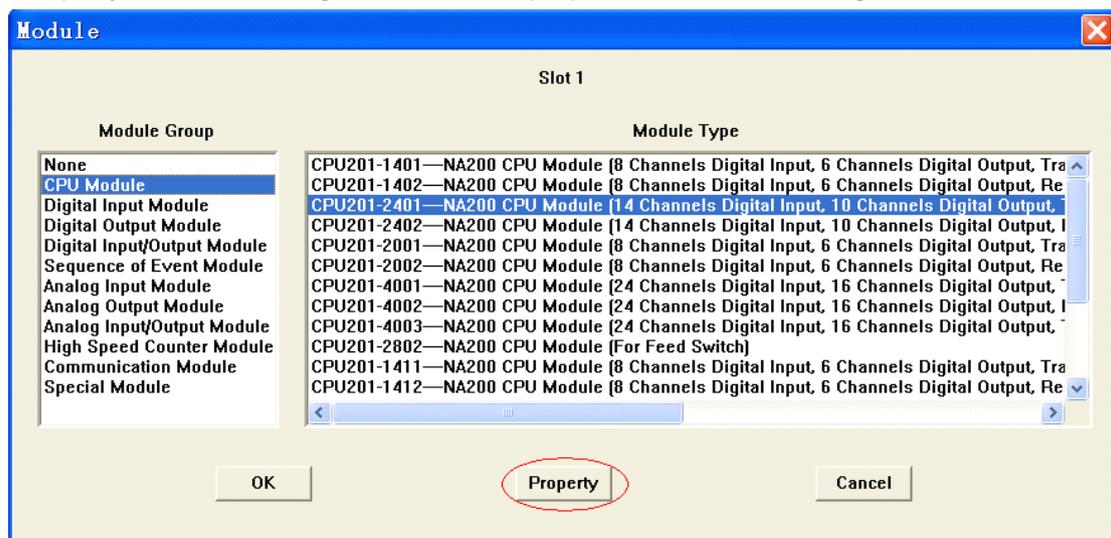


Fig.3.7 Module property

### CPU module

For different CPU modules, the configurations for the serial ports are same, select the "Baudrate", "Data Bit", "Stop Bit", "Parity" and "Protocol" by drop-down combo box. There are demands on configurations of address of serial port and start number of point. As shown in Fig.3.8:

**COM1**

Address 1

Baudrate 9600

Data Bit 8

Stop Bit 1

Parity None

Protocol MODBUS RTU

Start Number (%I) 1

Start Number (%IW) 1

Save Source File to PLC

**COM2**

Address 1

Baudrate 9600

Data Bit 8

Stop Bit 1

Parity None

Protocol MODBUS RTU

Start Number (%Q) 1

Start Number (%QW) 1

Create Debug Info

OK Cancel Advanced >>

Fig.3.8 CPU module parameter

### Digital input module

The digital input module is divided into two kinds, but the configurations are the same, only need to set the start number. After setting the start number, the other serial numbers will be arranged sequentially. As shown in Fig.3.9:

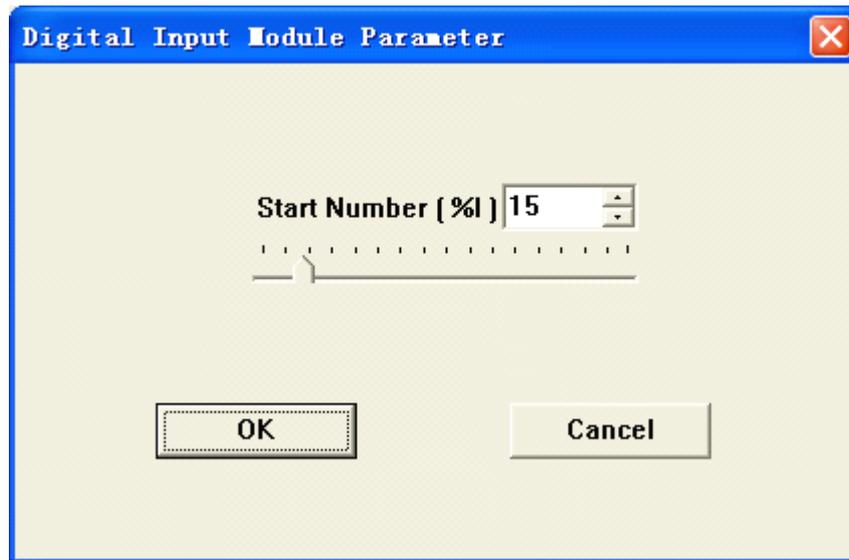


Fig.3.9 Digital input module parameter

**Digital output module**

The digital output module is divided into four kinds, but the configurations are the same, only need to set the start number. After setting the start number, the other serial numbers will be arranged sequentially. As shown in Fig.3.10:

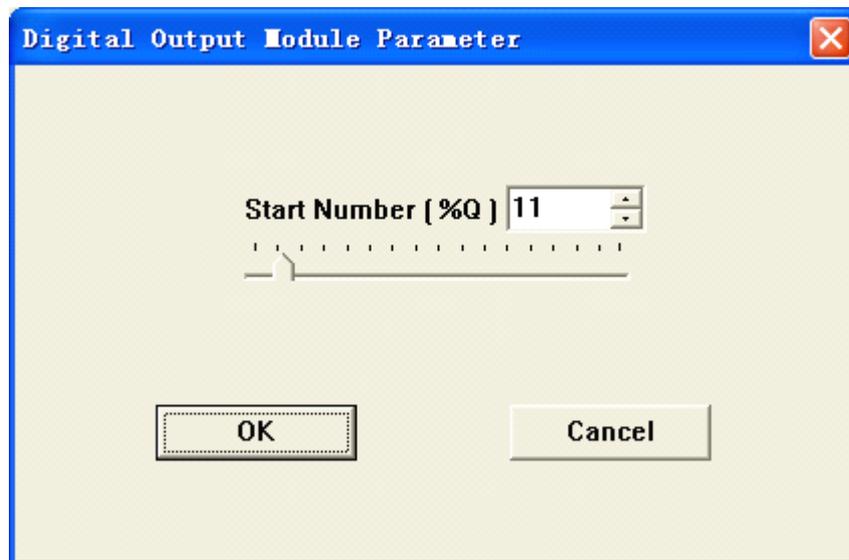


Fig.3.10 Digital output module parameter

**Digital input/output module**

The digital input/output module is divided into six kinds, but the configurations are the same, only need to set the start number. After setting the start number, the other serial numbers will be arranged sequentially. As shown in Fig.3.11:

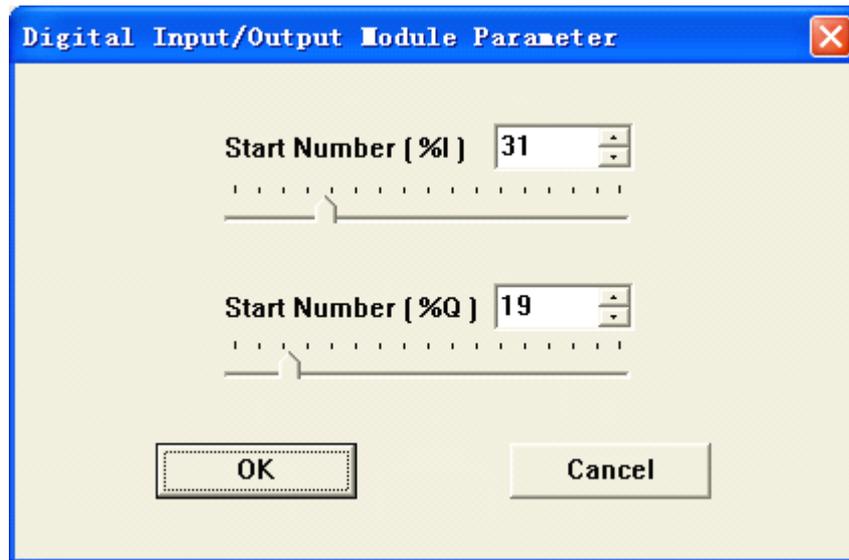


Fig.3.11 Digital input/output module parameter

### Analog input module

The analog input module is divided into several kinds, but the configurations are the same, only need to set the start number. After setting the start number, the other serial numbers will be arranged sequentially. As shown in Fig.3.12:

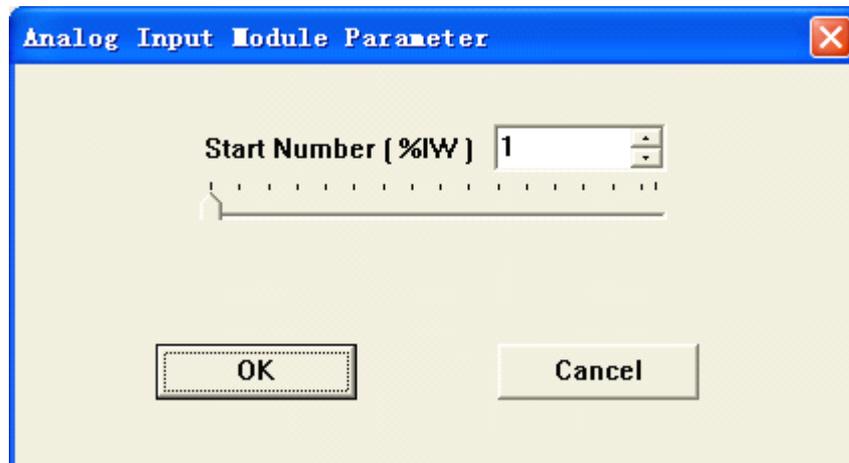


Fig.3.12 Analog input module parameter

### Analog output module

The analog output module is divided into four kinds, but configurations are the same, only need to set the start number. After setting the start number, the other serial numbers will be arranged sequentially. As shown in Fig.3.13:

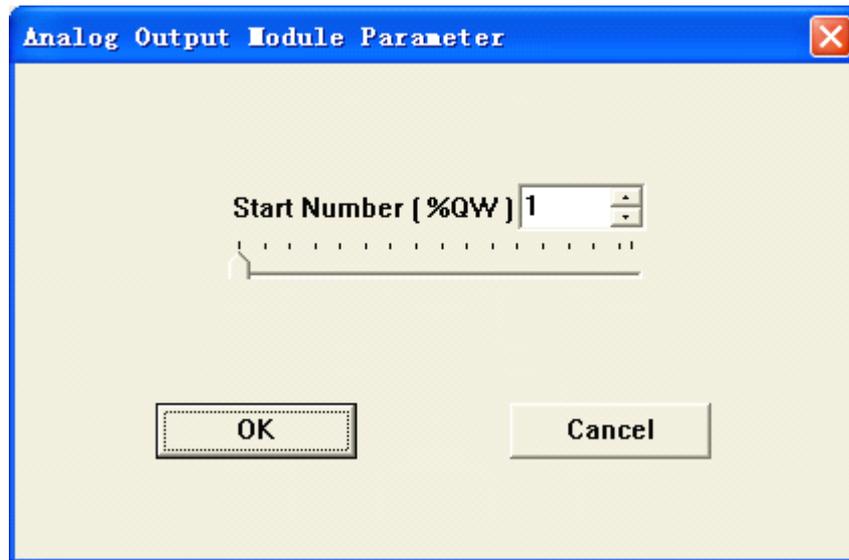


Fig.3.13 Analog output module parameter

### Analog input/output module

The analog input/output module is only one kind, for configurations only need to set the start number. After setting the start number, the other serial numbers will be arranged sequentially. As shown in Fig.3.14:

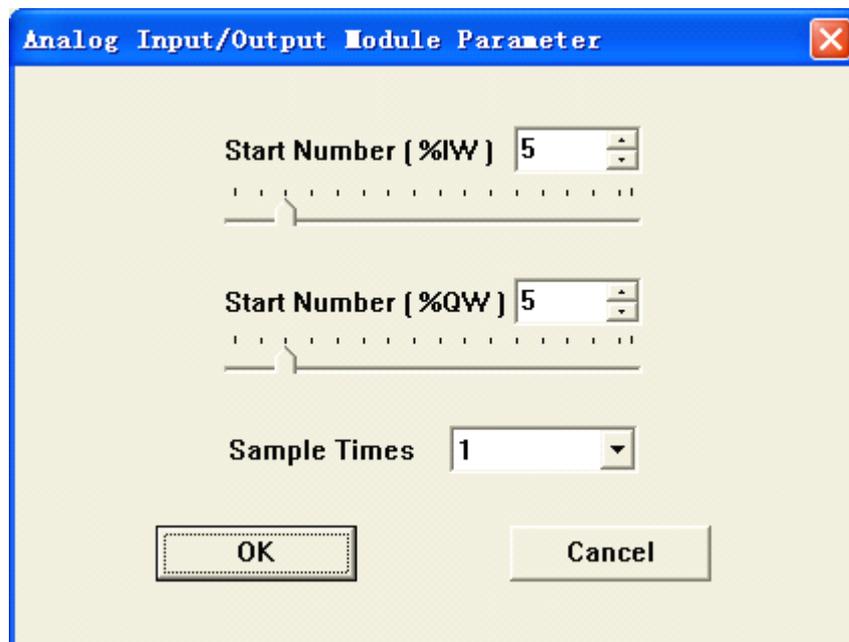


Fig.3.14 Analog input/output module parameter

## 3.3 Program

NA200 PLC program at least includes one main program "MAIN". Each scan cycle starts from the main program, and the scans of the other sub-programs are achieved by the way

of program call from the main program. The main program “MAIN” and all of the sub-programs are listed in the **Program** tab of project browser. Double click the program name to view the program contents. The programs are divided into the types of LD, FBD, IL, ST, etc., as shown in Fig.3.15:

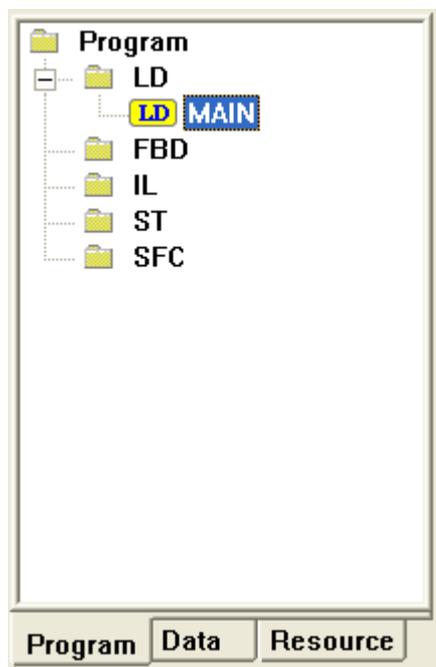


Fig.3.15 Program structure

### 3.3.1 Add program

If want to add one new program, click main menu **【File】 / 【New Program】** or toolbar  button. Then select the program type and name the program. “Program Description” is useful for user to understand the program usage, it can be filled or not. As shown in Fig.3.16:

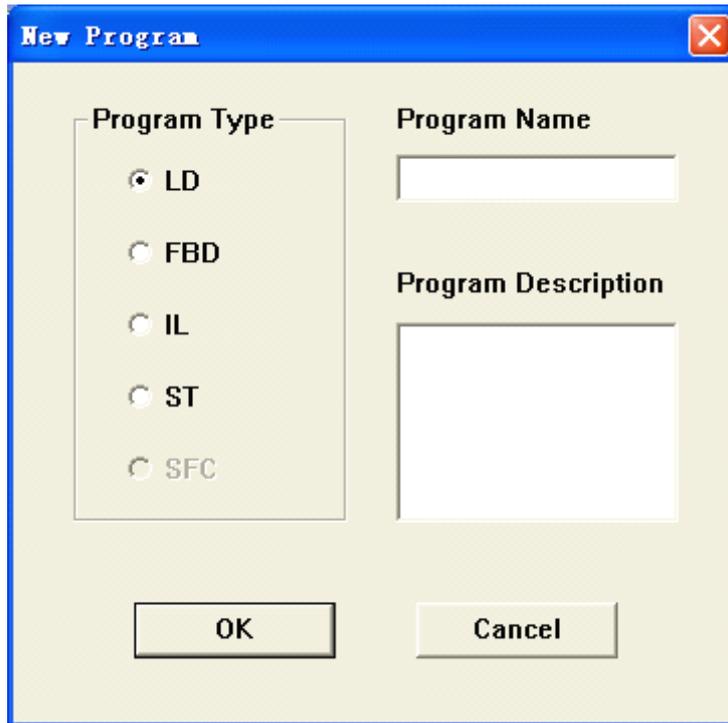


Fig.3.16 Add program

### 3.3.2 Delete program

If want to delete one program, click the program by the right mouse button to select **Delete** (the main program “MAIN” can’t be deleted). As shown in Fig.3.17:

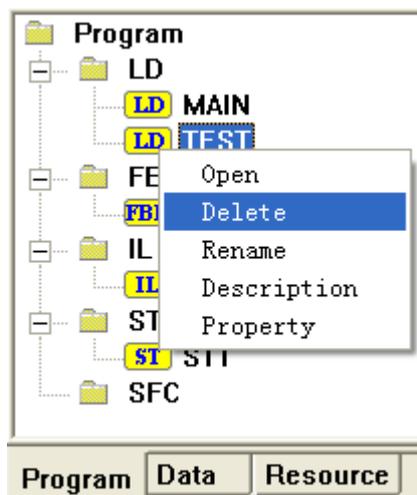


Fig.3.17 Delete program

### 3.3.3 Rename program

If want to rename one program, click the program by the right mouse button to select **Rename** (the main program “MAIN” can’t be renamed). As shown in Fig.3.18:

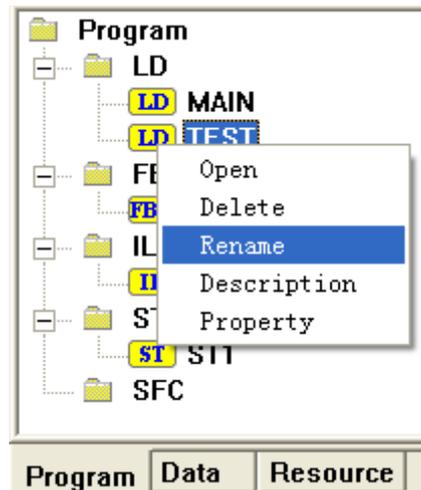


Fig.3.18 Rename program

### 3.3.4 Describe program

If want to describe one program, click the program by the right mouse button to select **Description**. The description contents are shown in the program editing area. As shown in Fig.3.19:

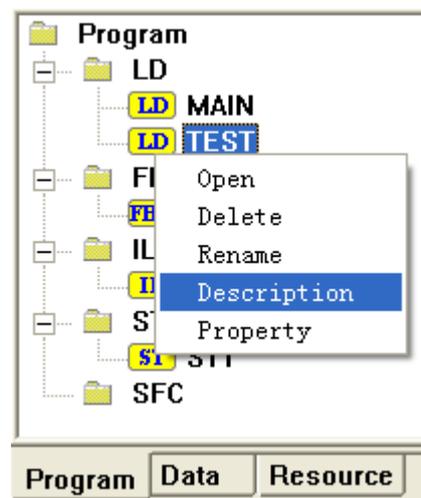


Fig.3.19 Describe program

## 3.4 Protect project

For a project, it can be set password to protect the project configurations, and protect the programs from being viewed and modified without allowance. There are two levels of file and login passwords. The file password is used to open the project file, and the login password is used to connect, upload and download. A new project does not have password, the developer can modify the password.

### 3.5 Connect and disconnect

Connect means that the NA200Pro programming software is connected and communicated with PLC by serial port. In the system toolbar, click the icon **【Connect】**, then NA200Pro enters into connection process. If connected successfully, the program execution status can be viewed, also each register status can be directly obtained, and also some registers can be forced to call programs. If connected unsuccessfully, the connect failure will be alarmed.

Disconnect means that the NA200Pro programming software is disconnected with PLC, and in the disconnect condition, each content of project can be modified, also project and program files can be downloaded.

### 3.6 Download and upload project

If want to execute a online command, after the project file is created or modified, it shall be transmitted into PLC, otherwise, when connecting with PLC, the following prompt will pop up, as shown in Fig.3.20:

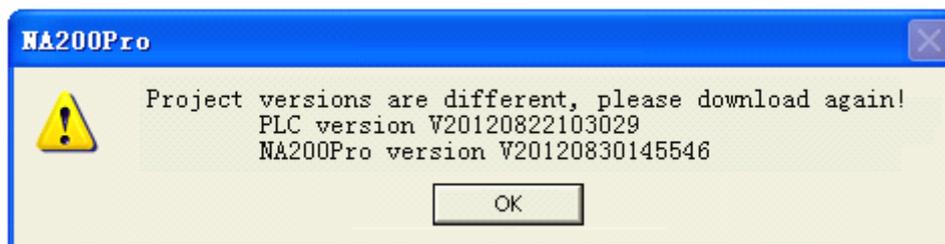


Fig.3.20 Connect warning

Download means that the edited programs are downloaded into PLC, including project and program files.

Upload means that the project and program files in PLC are saved into computer (it can be uploaded only when “Save Source File to PLC” is checked.).

#### Download project

Download the project file into PLC. During the downloading processes, the programming software will display the words of “Please wait while downloading project...”. After the project and the programs are downloaded into PLC, the CPU module must be reset and restarted once, and then the downloaded programs can be executed, otherwise, the system will still execute the former programs before downloading. The reset operation can be completed by **Reset** command of programming software. As shown in Fig.3.21:



Fig.3.21 Download project

### Upload project

It is to upload the project from PLC to the debugging computer. After uploading, the programming software will prompt to input the project save place, and it can directly overwrite the file with the same name or other files in the current computer, also, you can input a file name to save as a new file.

## 3.7 Download and upload program

### Download program

If the project file is not modified, only the program file is changed, the program file in PLC can be refreshed by the **Download Program** function.

### Upload program

Upload all of the program files from PLC, including LD, FBD, ST, and IL etc. The program upload is the same to project file upload. And then popup the following dialog box, select the program name to be uploaded, also all can be selected, as shown in Fig.3.22:

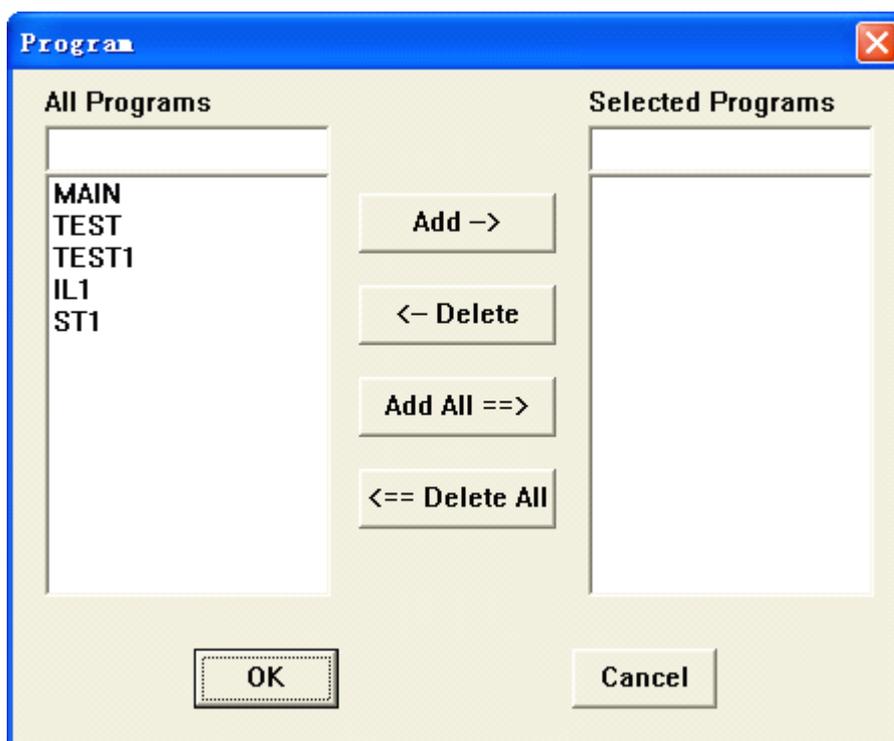


Fig.3.22 Upload program

### Download All

Download the project file and all the program files into PLC at the same time.

### Download All Programs

Download all the program files into PLC.

## Chapter 4 Data Management

The IEC61131-3 standard has determined the most commonly used data types of PLC programming, so in PLC field, the definitions and usages of these data types are uniform. It is benefit for the machine and device manufacturers and engineering and technical personnel using multiple PLC from different manufacturers to configure system: the uniform data types will enhance the PLC program portability.

### 4.1 Data type

NA200 PLC has the following several types of data:

Keyword	Type	Bit	Allowable range	Description
BOOL	Boolean	1	0 or 1	Store by the unit of bit, only two states: 1 or 0.
BYTE	Byte	8	0~255	Use 8 bits of data register, 8 bits of data can be independent, and only indicate the state of current bit: 0 or 1; also can be an unsigned integer with the range of 0~255.
WORD	Word	16	0~65535	Use 16 bits of data register, 16 bits of data can be independent, and only indicate the state of current bit: 0 or 1; also can be an unsigned integer with the range of 0~65535.
DWORD	Double word	32	0~4294967295	Use 32 bits of data register, 32 bits of data can be independent, and only indicate the state of current bit: 0 or 1; also can be an unsigned integer with the range of 0~4294967295.
SINT	Short integer	8	-128~+127	Use 8 bits of data register, indicate a signed integer with the range of -128~+127.
INT	Integer	16	-32768~+32767	Use 16 bits of data register, indicate a signed integer with the range of -32768~+32767.
DINT	Double integer	32	-2147483648~+2147483647	Use 32 bits of data register, indicate a signed integer with the range of -2147483648~+2147483647.
REAL	Real	32	—	Indicate a floating point.
USINT	Unsigned short integer	8	0~255	Use 8 bits of data register, indicate an unsigned integer with the range of 0~255.

UINT	Unsigned integer	16	0~65535	Use 16 bits of data register, indicate an unsigned integer with the range of 0~65535.
UDINT	Unsigned double integer	32	0~4294967295	Use 32 bits of data register, indicate an unsigned integer with the range of 0~4294967295.

## 4.2 Data management

### 4.2.1 Data tab

In project browser, **【Data】** tab includes **Optional Point Table** and **FLASH Table**, etc., as shown in Fig.4.1:

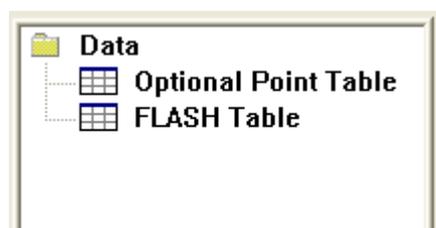


Fig.4.1 Data tab

### 4.2.2 Point table

#### Point type

Type	Name	Data type	Description
I	Digital input	BOOL, 0 or 1	The current state of basic digital input point.
Q	Digital output	BOOL, 0 or 1	The current state of basic digital output point.
IW	Analog input	INT Voltage and current signal:0~20000 Temperature signal: -32768~+32767	The current value of basic analog input point.
QW	Analog output	INT, 0~20000	The current value of basic analog output point.
M	Bit register	BOOL, 0 or 1	Boolean variable store area offered to customers by system.
MW	Word register	INT, -32768~32767	Word variable store area offered to customers by system.
N	Non-volatile bit register	BOOL, 0 or 1	The difference from M register is non-volatile. After the PLC power is off, the

			data in N register will never be lost.
NW	Non-volatile word register	INT, -32768~32767	The difference from MW register is non-volatile. After the PLC power is off, the data in NW register will never be lost.
S	System bit register	BOOL, 0 or 1	Boolean variable reflecting the system current state defined inside system. Each has its specific definition. Can be read, but can't be written.
SW	System word register	INT, -32768~32767	Word variable reflecting the system current state defined inside system. Each has its specific definition. Can be read, but can't be written.
T	Timer	INT, 0~30000	The timer offered to the customer by system.
C	Counter	INT, 0~32767	The counter offered to the customer by system.

NA200 PLC provides a large number of system registers to store the system operation status, so that the customer can read conveniently and real-time monitor the PLC operation status.

The system register definitions are as follows:

Number	Name	Description
<b>System bit register</b>		
%S0001	FIRST_SCAN	First Scan
%S0002	ALWAYS_ON	Always On
%S0003	ALWAYS_OFF	Always Off
%S0004	T_SECOND	1s Timer
%S0006	PRG_OVERRUN	Program Execution Overflow
%S0007	PRG_EXECERR	Program Execution Error
%S0011	FLASH_ERR	CPU Flash Error
%S0012	NVRAM_ERR	CPU NVRAM Error
%S0017	CPU_LED	CPU Run LED
%S0018	CPU_COMFLT	CPU Communication Error
%S0019	CPU_DEBUG	CPU Debug
%S0020	CPU_RUN	CPU Run
%S0021	CPU_STOP	CPU Stop
%S0025	IO_COMERR	I/O Module Communication Error
%S0026	IO_DIAGERR	I/O Module Self Diagnosis Error
%S0027	IO_CFGERR	I/O Module Type Mismatch

%S0028	IO_DOWNLOAD	I/O Module Init OK
<b>Module operation status</b>		
%S0129	MDU01_COMERR	Module (01) Communication Error
%S0130	MDU01_DIAGERR	Module (01) Self Diagnosis Error
%S0131	MDU01_CFGERR	Module (01) Type Mismatch
%S0132	MDU01_RVS1	Module (01) Reserve 1
%S0133	MDU01_RVS2	Module (01) Reserve 2
%S0134	MDU01_RVS3	Module (01) Reserve 3
%S0135	MDU01_RVS4	Module (01) Reserve 4
%S0136	MDU01_RVS5	Module (01) Reserve 5
%S0137	MDU02_COMERR	Module (02) Communication Error
%S0138	MDU02_DIAGERR	Module (02) Self Diagnosis Error
%S0139	MDU02_CFGERR	Module (02) Type Mismatch
%S0140	MDU02_RVS1	Module (02) Reserve 1
%S0141	MDU02_RVS2	Module (02) Reserve 2
%S0142	MDU02_RVS3	Module (02) Reserve 3
%S0143	MDU02_RVS4	Module (02) Reserve 4
%S0144	MDU02_RVS5	Module (02) Reserve 5
.....	.....	.....
.....	.....	.....
<b>System word register</b>		
%SW0001	TIME_YEAR	Clock : Year
%SW0002	TIME_MONTH	Clock : Month
%SW0003	TIME_DAY	Clock : Day
%SW0004	TIME_HOUR	Clock : Hour
%SW0005	TIME_MINUTE	Clock : Minute
%SW0006	TIME_SECOND	Clock : Second
%SW0007	TIME_MS	Clock : Millisecond
%SW0008	TIME_WEEK	Clock : Week
%SW0009	POT1_VALUE	Potentiometer 1 Value
%SW0010	POT2_VALUE	Potentiometer 2 Value
%SW0011	OVERRUN_INFO	Program Overflow Location
%SW0014	EXERRERR_INFO	Program Error Location
%SW0019	CPU_VER	CPU Firmware Version
%SW0021	COM1_SEND	COM1 Transmit State
%SW0022	COM1_RECV	COM1 Receive State
%SW0023	COM2_SEND	COM2 Transmit State
%SW0024	COM2_RECV	COM2 Receive State
%SW0025	STTM_YEAR	Start Time : Year
%SW0026	STTM_MONTH	Start Time : Month
%SW0027	STTM_DAY	Start Time : Day
%SW0028	STTM_HOUR	Start Time : Hour

%SW0029	STTM_MINUTE	Start Time : Minute
%SW0030	STTM_SECOND	Start Time : Second
%SW0031	STTM_MS	Start Time : Millisecond
%SW0032	SCAN_TIME	Scan Time
%SW0033	TIME1	Program 1 Execution Time
%SW0034	MAXTIME1	Program 1 Maximum Execution Time
%SW0035	MINTIME1	Program 1 Minimum Execution Time
%SW0036	TIME2	Program 2 Execution Time
%SW0037	MAXTIME2	Program 2 Maximum Execution Time
%SW0038	MINTIME2	Program 2 Minimum Execution Time
.....	.....	.....
.....	.....	.....

### Point property

The point property has two kinds of the common property that all types of points have and the special property that only some points have.

#### Common property

**【 Number 】**: Number is used to distinguish the points. It generates automatically and can't be modified. The number can be directly used as the operation object of function block or instruction. The number has two parts: point type and point number, such as %I0001, %MW0100 etc. The point type indicates the current type of point, such as %I indicates the digital input point, %MW indicates the word register. The point number indicates the current point serial number, which can't exceed its maximum value.

**【 Name 】**: Each point can be defined a name that can be directly used as operation object in programming of LD and FBD etc., and can be directly displayed in the program. For example, %I0001 is defined as "DL\_ON", in the LD editing area, place a contact, and input parameter "DL\_ON", after compiling, the programming software will automatically identify. If the point definition is changed, "DL\_ON" point is changed to the second digital input point, only need to define the %I0002 name as "DL\_ON", and no need any other modification in LD.

**【 Description 】**: Each point also can be defined a description which is to describe the point in detail. For example, %I0001 name is "DL\_ON", the description is "circuit breaker is on". When reading the LD program, if do not understand the meaning of "DL\_ON", you can view the description. However, the description content can't display in LD.

**【 Used Times 】**: Used Times is used to display the used times of current point in the programs, it can be obtained by **【 File 】/ 【 Used Times 】**, after saved, it is no need to count again at the next time. **Especially for timer and counter, the used times is very useful, it can be viewed to avoid reusing.**

#### Special property

**【 Module Address 】** :It is only valid to the actual points such as digital input I, digital output Q, analog input IW and analog output QW. The module address is the module address where the current point is, which is automatically generated after defined and can't be modified.

**【 Force 】** : It is only valid to actual points such as digital input I, digital output Q, analog input IW, and analog output QW. After forcing the point, no matter what the current value of signal it is, you can set the point value according to your demand. Force only can be executed at online state. Double click the force bar of point to be set value, at this time, a  $\surd$  mark will occur in the force bar, it indicates that this point has been forced. Double click again, this force bar is empty, it indicates that this point has not been forced. In force state, the signal state that scanned by system will never send into the point store area.

**【 Value 】** : It shows the current values of actual points at online state of the digital input I, digital output Q, analog input IW, and analog output QW; and also shows the current values of virtual points at online state of the bit register M, word register MW, non-volatile bit register N, non-volatile word register NW, system bit register S, and system word register SW. For the system register S and SW, it only can be read and can't be written.

**【 Signal Type 】** :It is only valid to analog input IW and analog output QW. The analog module can have different types of signals, such as voltage type, current type and thermal resistance.

**【 Preset Value 】** : It is only valid to the timer T and counter C. At online state, the preset time of timer or the preset value of counter can be viewed.

**【 Current Value 】** : It is only valid to the timer T and counter C. At online state, the current time of timer or the current value of counter can be viewed.

**【 Zero Offset 】** :It is only valid to analog input IW, and it is used to make an error compensation for the actual signal.

### 4.2.3 Optional point table

At online state, view/modify the point value by optional point table. Optional point table can freely define the desired points, and so as to only observe the desired points when program debugging. In project browser, select **【 Data 】 / 【 Optional Point Table 】** , then pop up a optional point table. If it is used for the first time, then this point table is an empty table. As shown in Fig.4.2:

Enter	Value	Number	Name	Force	No.

Fig.4.2 Empty optional point table

**Add point**

Click the right mouse button in the blank to select **Add**, and then the **Point Name** dialog box will pop up, input the point name. As shown in Fig.4.3:

Enter	Value	Number	Name	Force	No.
...		%M0001			1
...		%M0002			2
...		%M0003			3
...		%M0004			4
...		%M0005			5
...		%M0006			6
...		%M0007			7
...		%M0008			11
...		%M0011			9

**Point Name** ✖

Point Name

Fig.4.3 Add point

**Add from Program**

Click the right mouse button in the optional point table to select **Add from Program**, then **Select Program** dialog box will pop up, select the name of program to be added and click **OK**, then all the points used in this program will be added into this optional point table. As shown in Fig.4.4:

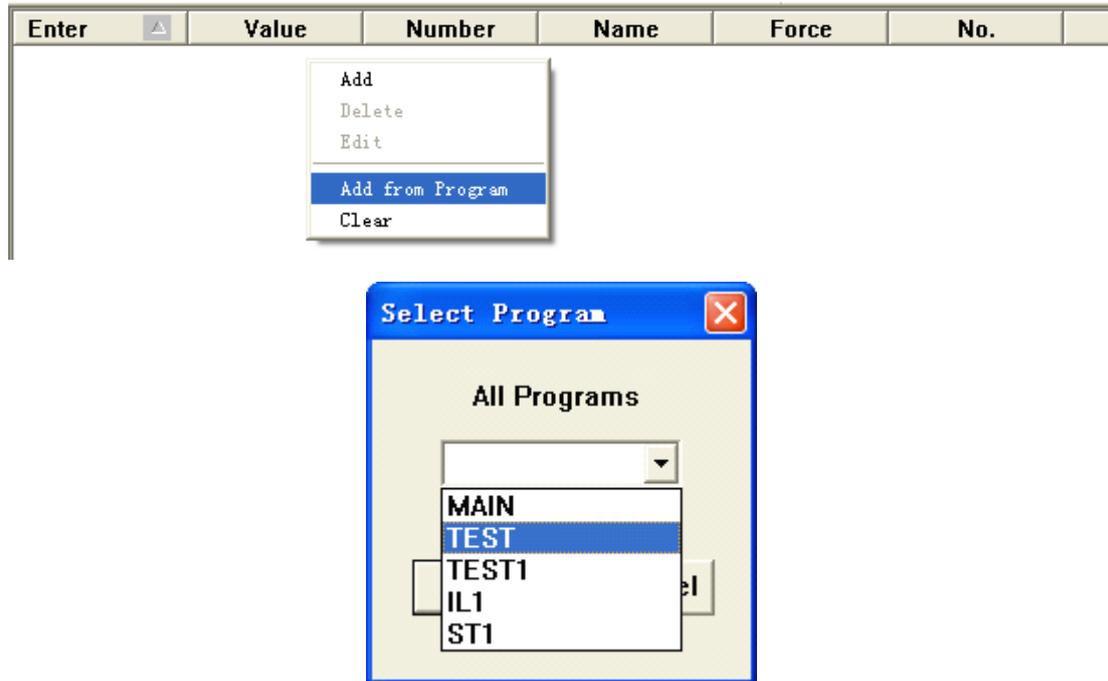


Fig.4.4 Add point from program

After adding the points, the information of point value, number, name, force etc. can be observed from the optional point table. As shown in Fig.4.5:

Enter	Value	Number	Name	Force	No.
%M0001		%M0001			1
%M0002		%M0002			2
%M0003		%M0003			3
%M0004		%M0004			4
%M0005		%M0005			5
%M0006		%M0006			6
%M0007		%M0007			7
%M0008		%M0008			11
%M0011		%M0011			9
%M0012		%M0012			10
%MW0002		%MW0002			8
%MW0022		%MW0022			12
%MW0023		%MW0023			13

Fig.4.5 Optional point table

### Force point value

It is used to modify the point value in the optional point table. For the points of %M, %MW, %N, %NM, %Q, and %QW, the point value can be modified by double clicking the point value to be modified in the **Value** column; For the points of %I and %IW, the point values can be modified in the **Value** column only after the force option is ticked by double clicking the **Force** column.

### Delete and edit point

If do not need some points, they can be deleted or modified to be other points. The

operation can be selected by clicking the right mouse button on this column. As shown in Fig.4.6:

Enter	Value	Number	Name	Force	No.
%M0001		%M0001			1
%M0002		%M0002			2
%M0003		%M0003			3
%M0004		%M0004			4
%M0005		%M0005			5
%M0006		%M0006			6
%M0007		%M0007			7
%M0008		%M0008			11
%M0011		%M0011			9
%M0012		%M0012			10
%MW0002		%MW0002			8
%MW0022		%MW0022			12
%MW0023		%MW0023			13

Fig.4.6 Delete and edit point

### 4.3 Addressing mode

Addressing is the method of LD and other programming modes accessing to the points. There are the following several kinds of addressing modes:

#### 1. Immediate addressing

The constant is the access object. For the following **MOVE** function block, the execution function is to assign 100 to %MW0001, and then the input 100 is the immediate addressing mode.

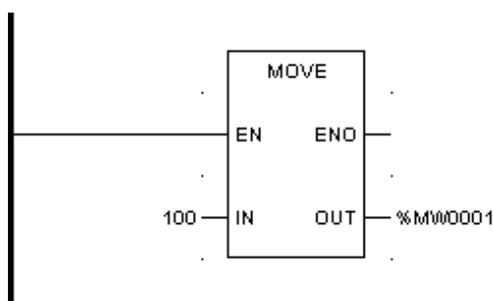


Fig.4.7 Immediate addressing

※ In immediate addressing mode, when input data, the hexadecimal data shall be followed by the letter “H”, if the first number of hexadecimal data is a letter, then the “0” shall be added in front of the letter. For example, if you want to input B021H, it shall be written as 0B021H, otherwise, error occurs when system compiling.

#### 2. Direct addressing

The point type adding point number is the access object of direct addressing. As shown in Fig.4.7, for the **MOVE** operation, %MW0001 is a specified point, it is direct addressing mode.

As a special example, using the point name also belongs to direct addressing, because the point name corresponds to a specified point. For example, define the name of point %I0001 as “GD\_ON”, then “GD\_ON” can be directly used in the program, it is referred to %I0001.

### 3. Indirect addressing

The point number of access object for the indirect addressing is not a constant, but the other point, that is, the current value of this point is used as the point number. It is different from the point number of direct addressing, the indirect addressing point starts from 0, that is, for an indirect addressing point %I[%MW0001], the current value of %MW0001 is 1, the accessing point is %I0002, instead of %I0001. As shown in Fig.4.8, %Q[%MW0001] is indirect addressing mode. Its point type is Q, the point number is not a constant, but the other point %MW0001. The LD execution operation is: %MW0001 is assigned as 0, the coil outputs 1 for the output point %Q0001.

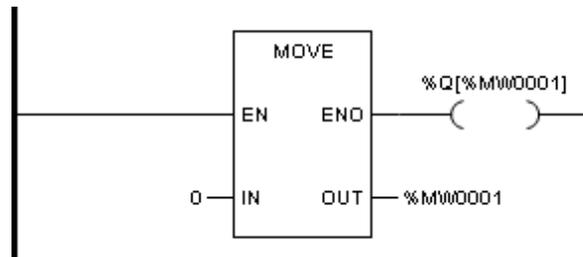


Fig.4.8 Indirect addressing

# Chapter 5 Basic Function Block

## 5.1 Introduction

For the graphic languages (LD and FBD), the function block is indicated as a frame with input and output pins. The input is always at the left of frame and output is always at the right of frame. The name of function block (that is the type of function block) displays in the middle of frame.

All the basic function blocks in this chapter can be called in the ladder diagram (LD), function block diagram (FBD), structured text (ST), and instruction list (IL).

### 5.1.1 Property modification

The properties of each function block can be modified. In FBD, “EN/ENO” can be displayed/hidden. For some function blocks, the number of inputs can be added.

As shown in Fig.5.1, in LD editor, click the function block whose property needs to be viewed (the **ADD** addition function block in the Fig.) by the left mouse button, and choose **Property** by clicking the right mouse button, then pop up the dialog box as shown in Fig.5.2, and the “Input Number” can be chosen by drop-down combo box. In FBD editor, the “Display EN/ENO” can be modified as an alternative.

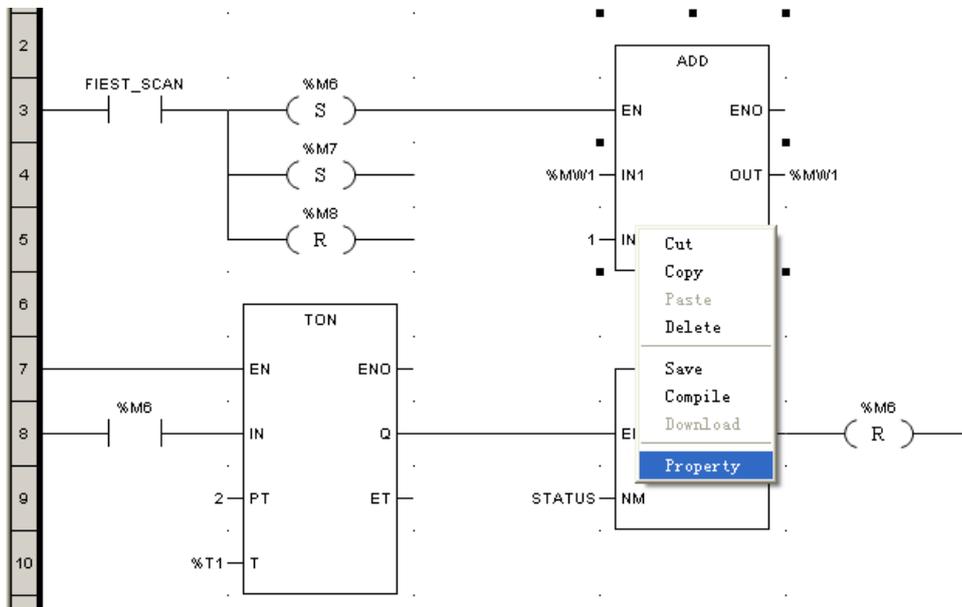


Fig.5.1 FB property modification

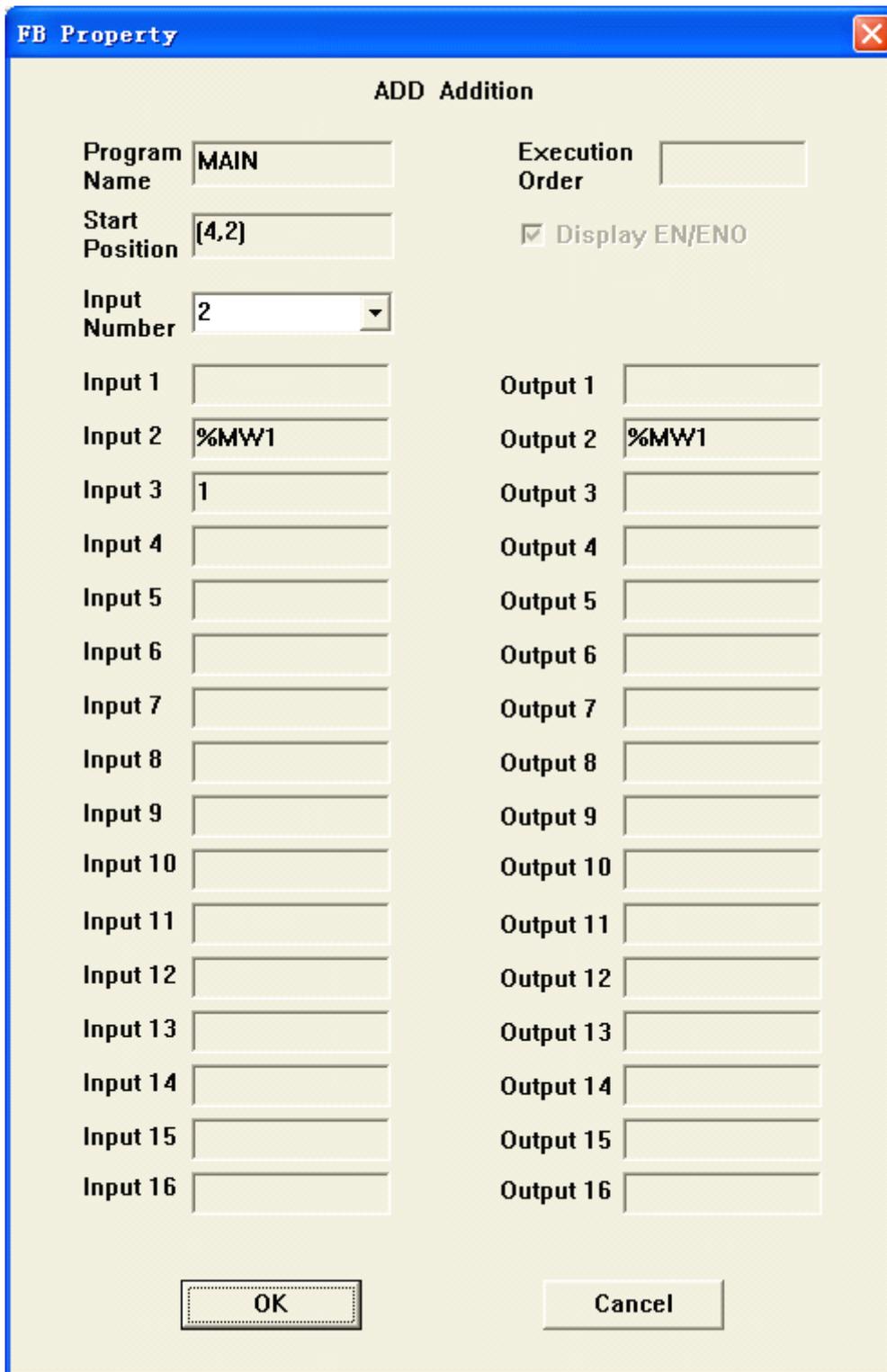


Fig.5.2 Dialog box of property modification

### 5.1.2 EN/ENO

It can configure **EN** input and **ENO** output for all function blocks. If the **EN** input is 0 when the function block is called, then the defined algorithm by the function block is not

executed, and the **ENO** is set to 0. If the **EN** input is 1 when the function block is called, then the defined algorithm by the function block is executed, and the **ENO** is set to 1, and if there is any error during execution, the **ENO** will be set to 0.

## 5.2 Mathematics

This chapter describes the elementary function blocks of the Mathematics family.

This chapter contains the following sections.

Type	Description	Formula
<b>ADD</b>	16-Bit Addition	$OUT = IN1 + IN2 + \dots + INn$
<b>DADD</b>	32-Bit Addition	
<b>EADD</b>	Floating-Point Addition	
<b>SUB</b>	16-Bit Subtraction	$OUT = IN1 - IN2$
<b>DSUB</b>	32-Bit Subtraction	
<b>ESUB</b>	Floating-Point Subtraction	
<b>MUL</b>	16-Bit Multiplication	$OUT = IN1 * IN2 * \dots * INn$
<b>DMUL</b>	32-Bit Multiplication	
<b>EMUL</b>	Floating-Point Multiplication	
<b>DIV</b>	16-Bit Division	$OUT = IN1 / IN2$
<b>DDIV</b>	32-Bit Division	
<b>EDIV</b>	Floating-Point Division	
<b>MOD</b>	16-Bit Modulo	$OUT = IN1 \% IN2$
<b>DMOD</b>	32-Bit Modulo	
<b>DIVMOD</b>	16-Bit Division and Modulo	$DV = IN1 / IN2$
<b>DDIVMOD</b>	32-Bit Division and Modulo	$MD = IN1 \% IN2$
<b>INC</b>	16-Bit Increment	$INOUT = INOUT + 1$
<b>DINC</b>	32-Bit Increment	
<b>DEC</b>	16-Bit Decrement	$INOUT = INOUT - 1$
<b>DDEC</b>	32-Bit Decrement	
<b>NEG</b>	16-Bit Negation	$OUT = 0 - IN$

<b>DNEG</b>	32-Bit Negation	
<b>ENEG</b>	Floating-Point Negation	
<b>SIGN</b>	16-Bit Sign Evaluation	
<b>DSIGN</b>	32-Bit Sign Evaluation	if $IN < 0$ , $OUT = 1$ If $IN \geq 0$ , $OUT = 0$
<b>ESIGN</b>	Floating-Point Sign Evaluation	
<b>ABS</b>	16-Bit Absolute Value	
<b>DABS</b>	32-Bit Absolute Value	$OUT =  IN $
<b>EABS</b>	Floating-Point Absolute Value	
<b>SQRT</b>	Floating-Point Square Root	$OUT = \sqrt{IN}$
<b>LOG</b>	Floating-Point Decimal Logarithm	$OUT = \log_{10}^{IN}$
<b>LN</b>	Floating-Point Natural Logarithm	$OUT = \ln^{IN}$ , or $OUT = \log_e^{IN}$ , wherein $e = 2.718282$
<b>EXP</b>	Floating-Point Natural Exponentiation	$OUT = e^{IN}$ , wherein $e = 2.718282$
<b>EXPT</b>	Floating-Point Exponentiation	$OUT = IN1^{IN2}$
<b>SIN</b>	Floating-Point Sine	$OUT = \sin IN$
<b>COS</b>	Floating-Point Cosine	$OUT = \cos IN$
<b>TAN</b>	Floating-Point Tangent	$OUT = \tan IN$
<b>ASIN</b>	Floating-Point Arc Sine	$OUT = \text{asin } IN$
<b>ACOS</b>	Floating-Point Arc Cosine	$OUT = \text{acos } IN$
<b>ATAN</b>	Floating-Point Arc Tangent	$OUT = \text{atan } IN$

### 5.2.1 ADD: 16-bit addition

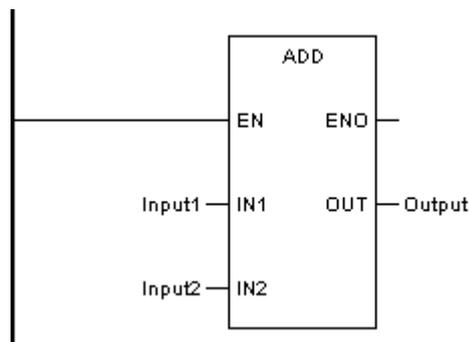
#### ◆ Function description

This function block adds the input values and assigns the result to the output.  
The number of inputs can be increased to a maximum of 8.

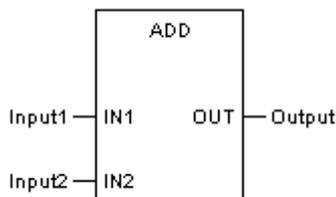
#### ◆ Formula

$$OUT = IN1 + IN2 + \dots + INn$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

```
LD      Input1
ADD     Input2
ST      Output
```

◆ **Representation in ST**

```
Output := ADD (Input1, Input2);
```

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Summand	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Summand	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Sum	INT	MW, NW

**5.2.2 DADD: 32-bit addition**

◆ **Function description**

This function block adds the input values and assigns the result to the output. Input

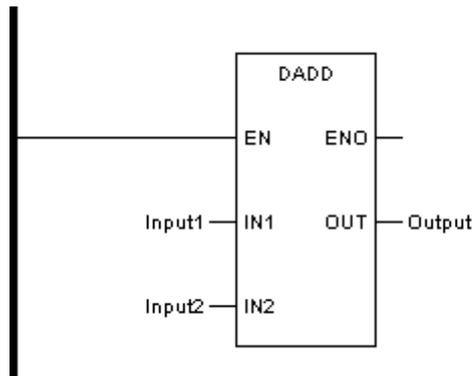
and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

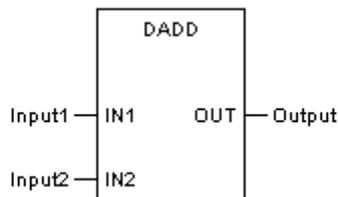
◆ **Formula**

$$OUT = IN1 + IN2 + \dots + INn$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Summand	DINT	Constant, MW, NW
IN2	Input2	Summand	DINT	Constant MW, NW
OUT	Output	Sum	DINT	MW, NW

### 5.2.3 EADD: Floating-point addition

◆ **Function description**

This function block adds the input values and assigns the result to the output. Input

and output can only be floating-point register, occupying 2 continuous word registers.

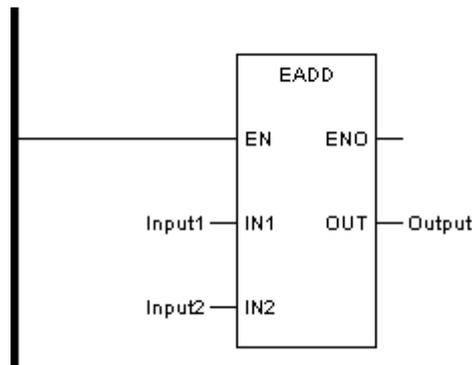
The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

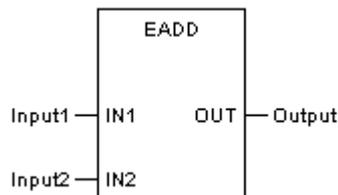
◆ **Formula**

$$OUT = IN1 + IN2 + \dots + INn$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Summand	REAL	Constant, MW, NW
IN2	Input2	Summand	REAL	Constant, MW, NW
OUT	Output	Sum	REAL	MW, NW

## 5.2.4 SUB: 16-bit subtraction

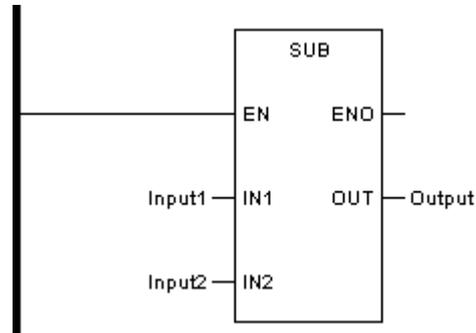
◆ **Function description**

This function block subtracts the value at the IN2 input from the value at the IN1 input and assigns the result to the output.

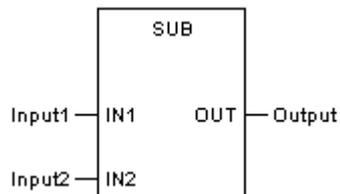
◆ **Formula**

$$\text{OUT} = \text{IN1} - \text{IN2}$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD	Input1
SUB	Input2
ST	Output

◆ **Representation in ST**

Output := SUB (Input1, Input2);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Minuend	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Subtrahend	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Difference	INT	MW, NW

## 5.2.5 DSUB: 32-bit subtraction

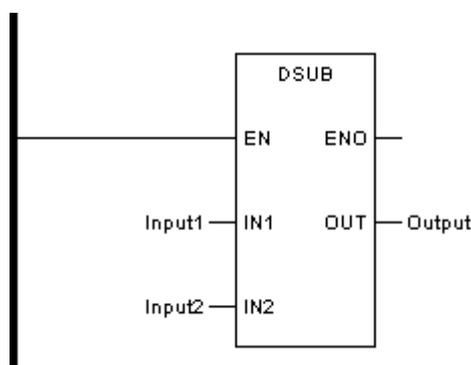
### ◆ Function description

This function block subtracts the value at the IN2 input from the value at the IN1 input and assigns the result to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

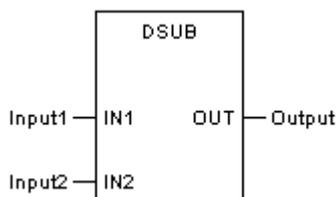
### ◆ Formula

$$OUT = IN1 - IN2$$

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Minuend	DINT	Constant, MW, NW
IN2	Input2	Subtrahend	DINT	Constant, MW, NW
OUT	Output	Difference	DINT	MW, NW

## 5.2.6 ESUB: Floating-point subtraction

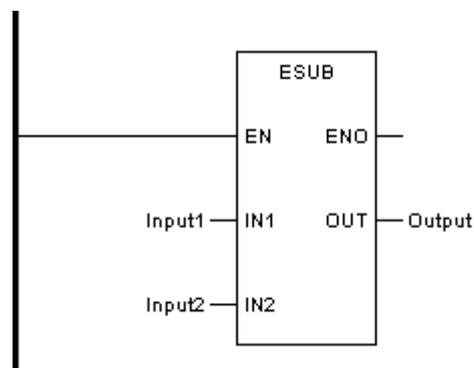
### ◆ Function description

This function block subtracts the value at the IN2 input from the value at the IN1 input and assigns the result to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

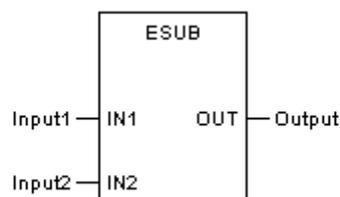
### ◆ Formula

$$\text{OUT} = \text{IN1} - \text{IN2}$$

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Minuend	REAL	Constant, MW, NW
IN2	Input2	Subtrahend	REAL	Constant, MW, NW
OUT	Output	Difference	REAL	MW, NW

## 5.2.7 MUL: 16-bit multiplication

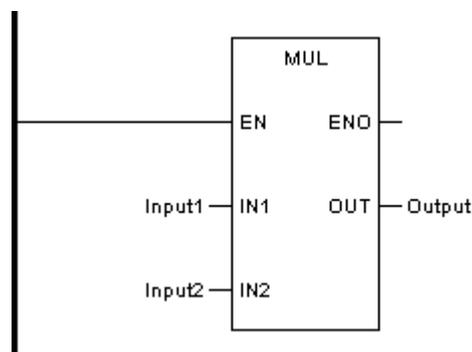
### ◆ Function description

This function block multiplies the input values and assigns the result to the output. The number of inputs can be increased to a maximum of 8.

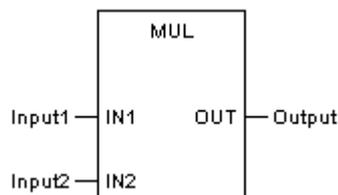
### ◆ Formula

$$\text{OUT} = \text{IN1} * \text{IN2} * \dots * \text{INn}$$

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Representation in IL

```
LD      Input1
MUL    Input2
ST     Output
```

### ◆ Representation in ST

```
Output := MUL (Input1, Input2);
```

### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
------	-----------	-------------	-----------	------------

IN1	Input1	Multiplicand	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Multiplier	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Product	INT	MW, NW

## 5.2.8 DMUL: 32-bit multiplication

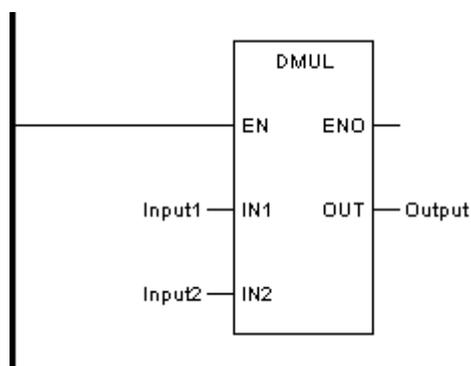
### ◆ Function description

This function block multiplies the input values and assigns the result to the output. The number of inputs can be increased to a maximum of 8. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

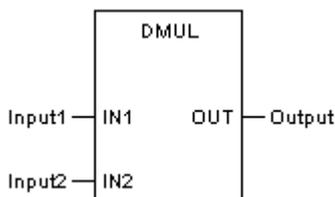
### ◆ Formula

$$OUT = IN1 * IN2 * \dots * INn$$

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Multiplicand	DINT	Constant, MW, NW

IN2	Input2	Multiplier	DINT	Constant, MW, NW
OUT	Output	Product	DINT	MW, NW

## 5.2.9 EMUL: Floating-point multiplication

### ◆ Function description

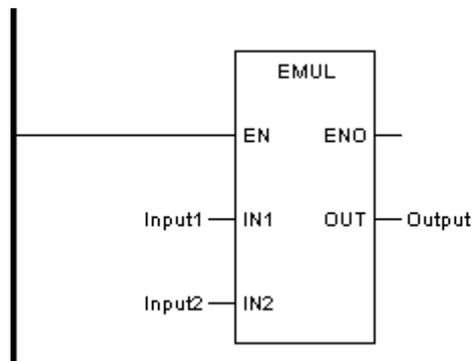
This function block multiplies the input values and assigns the result to the output.

The number of inputs can be increased to a maximum of 8. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

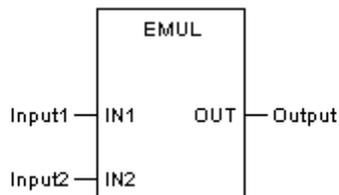
### ◆ Formula

$$OUT = IN1 * IN2 * \dots * INn$$

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Multiplicand	REAL	Constant, MW, NW

IN2	Input2	Multiplier	REAL	Constant, MW, NW
OUT	Output	Product	REAL	MW, NW

## 5.2.10 DIV: 16-bit division

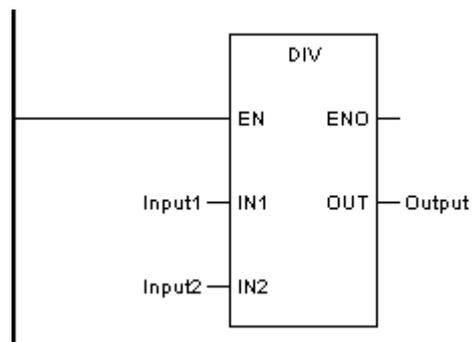
### ◆ Function description

This function block divides the value at the IN1 input by the value at the IN2 input and assigns the result to the output.

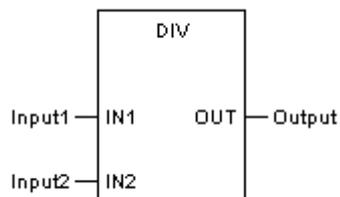
### ◆ Formula

$$\text{OUT} = \text{IN1} / \text{IN2}$$

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Representation in IL

```

LD      Input1
DIV     Input2
ST      Output
    
```

### ◆ Representation in ST

```

Output := DIV (Input1, Input2);
    
```

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Dividend	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Divisor	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Quotient	INT	MW, NW

**5.2.11 DDIV: 32-bit division**

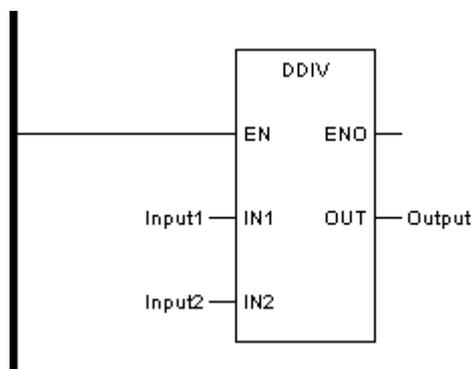
◆ **Function description**

This function block divides the value at the IN1 input by the value at the IN2 input and assigns the result to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

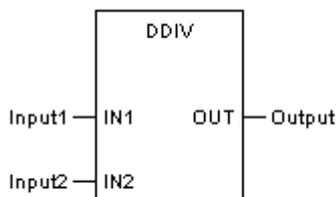
◆ **Formula**

$$OUT = IN1 / IN2$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Dividend	DINT	Constant, MW, NW
IN2	Input2	Divisor	DINT	Constant, MW, NW
OUT	Output	Quotient	DINT	MW, NW

## 5.2.12 EDIV: Floating-point division

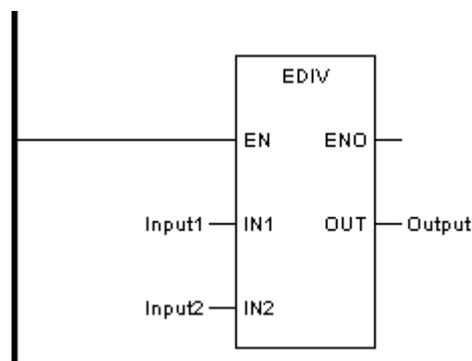
### ◆ Function description

This function block divides the value at the IN1 input by the value at the IN2 input and assigns the result to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

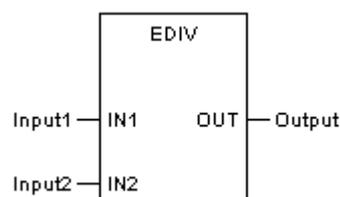
### ◆ Formula

$$\text{OUT} = \text{IN1} / \text{IN2}$$

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Dividend	REAL	Constant, MW, NW
IN2	Input2	Divisor	REAL	Constant, MW, NW
OUT	Output	Quotient	REAL	MW, NW

### 5.2.13 MOD: 16-bit modulo

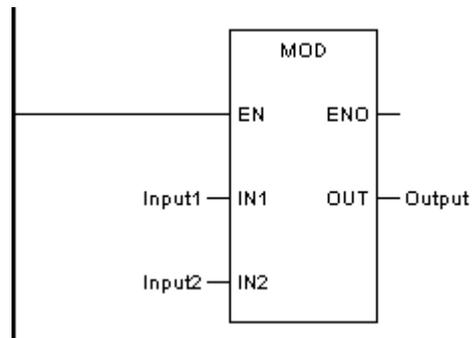
#### ◆ Function description

This function block divides the value at the IN1 input by the value at the IN2 input and assigns the modulo to the output.

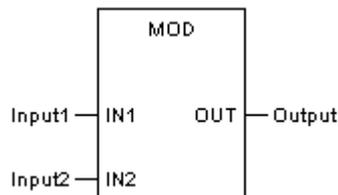
#### ◆ Formula

$$OUT = IN1 \% IN2$$

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

LD	Input1
MOD	Input2
ST	Output

◆ **Representation in ST**

Output := MOD (Input1, Input2);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Dividend	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Divisor	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Modulo	INT	MW, NW

**5.2.14 DMOD: 32-bit modulo**

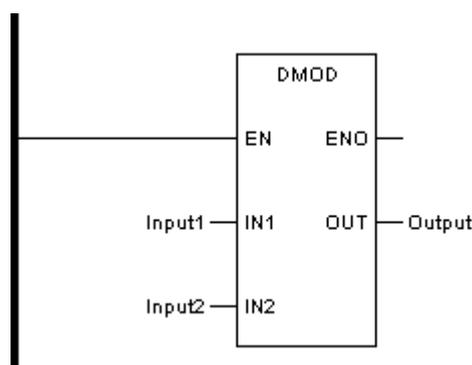
◆ **Function description**

This function block divides the value at the IN1 input by the value at the IN2 input and assigns the modulo to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

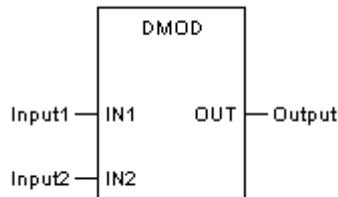
◆ **Formula**

OUT = IN1 % IN2

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Dividend	DINT	Constant, MW, NW
IN2	Input2	Divisor	DINT	Constant, MW, NW
OUT	Output	Modulo	DINT	MW, NW

**5.2.15 DIVMOD: 16-bit division and modulo**

◆ **Function description**

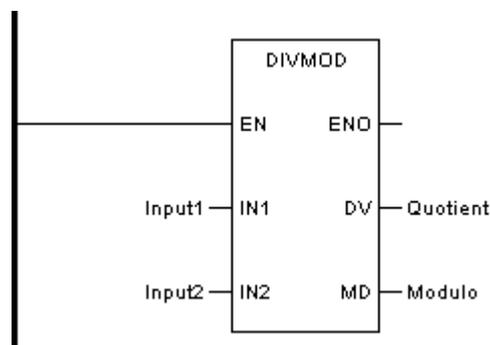
This function block divides the value at the IN1 input by the value at the IN2 input. The result of the division is delivered at the DV output. The remainder of the division is delivered at the MD output.

◆ **Formula**

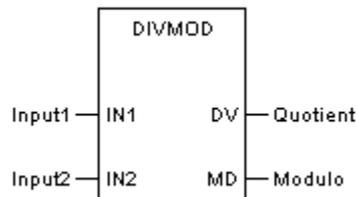
$$DV = IN1 / IN2$$

$$MD = IN1 \% IN2$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL DIVMOD (IN1:=Input1, IN2:=Input2, DV=>Quotient, MD=>Modulo)

◆ **Representation in ST**

DIVMOD (IN1:=Input1, IN2:=Input2, DV=>Quotient, MD=>Modulo);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Dividend	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Divisor	INT	Constant, IW, QW, MW, NW, SW
DV	Quotient	Quotient	INT	MW, NW
MD	Modulo	Modulo	INT	MW, NW

**5.2.16 DDIVMOD: 32-bit division and modulo**

◆ **Function description**

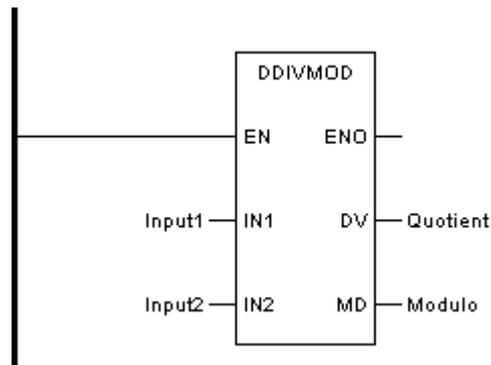
This function block divides the value at the IN1 input by the value at the IN2 input. The result of the division is delivered at the DV output. The remainder of the division is delivered at the MD output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Formula**

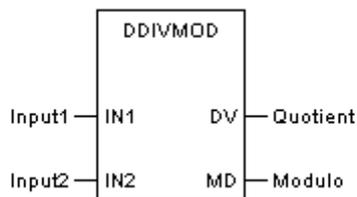
$$DV = IN1 / IN2$$

$$MD = IN1 \% IN2$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Dividend	DINT	Constant, MW, NW
IN2	Input2	Divisor	DINT	Constant, MW, NW
DV	Quotient	Quotient	DINT	MW, NW
MD	Modulo	Modulo	DINT	MW, NW

**5.2.17 INC: 16-bit increment**

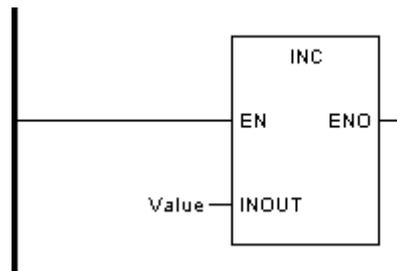
◆ **Function description**

This function block increments a variable by 1.

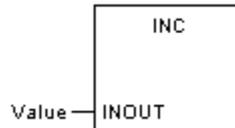
◆ **Formula**

$$INOUT = INOUT + 1$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL INC (Value)

◆ **Representation in ST**

INC (Value);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
INOUT	Value	Each time the program uses this function block, the variable value is incremented by one unit.	INT	MW, NW

**5.2.18 DINC: 32-bit increment**

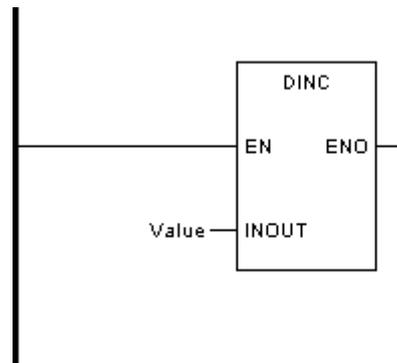
◆ **Function description**

This function block increments a variable by 1. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

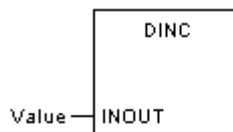
◆ **Formula**

$$INOUT = INOUT + 1$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
INOUT	Value	Each time the program uses this function block, the variable value is incremented by one unit.	DINT	MW, NW

**5.2.19 DEC: 16-bit decrement**

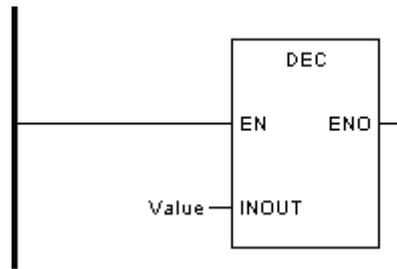
◆ **Function description**

This function block decrements a variable by 1.

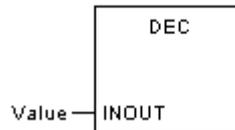
◆ **Formula**

$$\text{INOUT} = \text{INOUT} - 1$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL DEC (Value)

◆ **Representation in ST**

DEC (Value);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
INOUT	Value	Each time the program uses this function block, the variable value is decremented by one unit.	INT	MW, NW

**5.2.20 DDEC: 32-bit decrement**

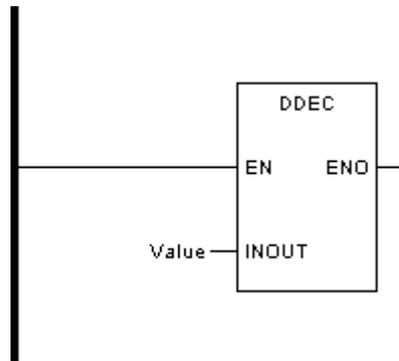
◆ **Function description**

This function block decrements a variable by 1. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

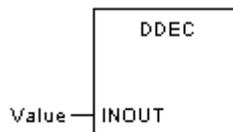
◆ **Formula**

$$INOUT = INOUT - 1$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
INOUT	Value	Each time the program uses this function block, the variable value is decremented by one unit.	DINT	MW, NW

**5.2.21 NEG: 16-bit negation**

◆ **Function description**

This function block negates the input and assigns the result to the output.

The negation causes a sign reversal, e.g.

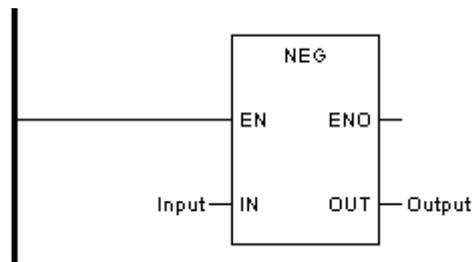
300 -> -300

-200->200

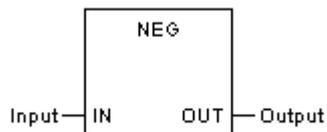
◆ **Formula**

$$OUT = 0 - IN$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

```

LD      Input
NEG
ST      Output
    
```

◆ **Representation in ST**

```

Output := NEG (Input );
    
```

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Negated output	INT	MW, NW

## 5.2.22 DNEG: 32-bit negation

◆ **Function description**

This function block negates the input and assigns the result to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

The negation causes a sign reversal, e.g.

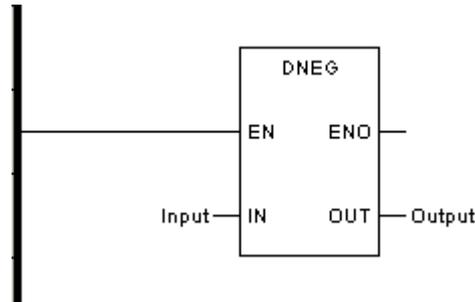
300000 -> -300000

-200000->200000

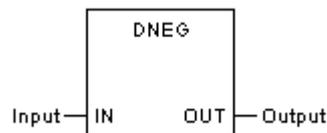
◆ **Formula**

$$OUT = 0 - IN$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	DINT	Constant, MW, NW
OUT	Output	Negated output	DINT	MW, NW

### 5.2.23 ENEG: Floating-point negation

◆ **Function description**

This function block negates the input and assigns the result to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

The negation causes a sign reversal, e.g.

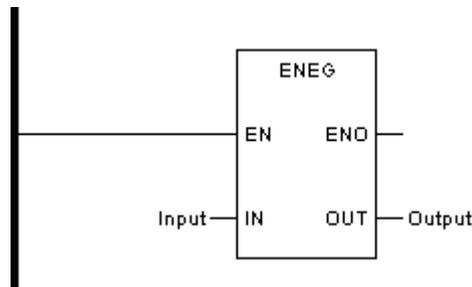
30.33 -> -30.33

-200.55->200.55

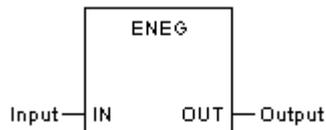
◆ **Formula**

OUT = 0 - IN

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW
OUT	Output	Negated output	REAL	MW, NW

### 5.2.24 SIGN: 16-bit sign evaluation

◆ **Function description**

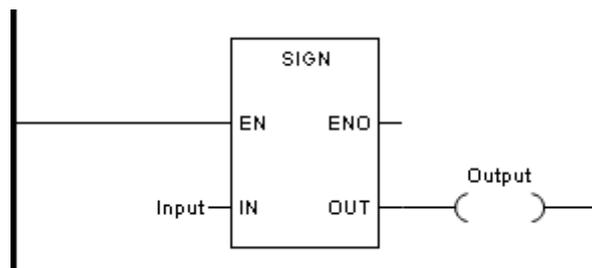
This function block is used for the detection of negative sign.

With a value  $\geq 0$  at the IN input, the OUT output becomes “0”. With a value  $< 0$  at the IN input, the OUT output becomes “1”.

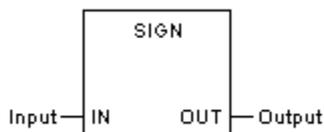
◆ **Formula**

OUT = 1, if  $IN < 0$ ,  
 OUT = 0, if  $IN \geq 0$ .

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

```
LD      Input
SIGN
ST      Output
```

◆ **Representation in ST**

```
Output := SIGN (Input );
```

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Signed input	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Sign evaluation	BOOL	Q, M, N

**5.2.25 DSIGN: 32-bit sign evaluation**

◆ **Function description**

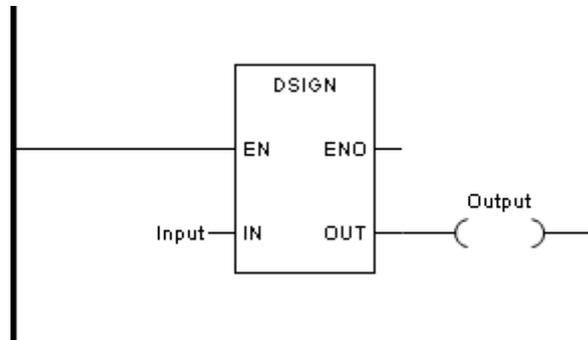
This function block is used for the detection of negative sign. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

With a value  $\geq 0$  at the IN input, the OUT output becomes "0". With a value  $< 0$  at the IN input, the OUT output becomes "1".

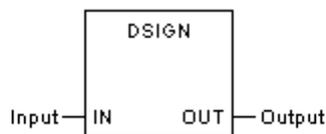
◆ **Formula**

OUT = 1, if IN < 0,  
 OUT = 0, if IN ≥ 0.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Signed input	DINT	Constant, MW, NW
OUT	Output	Sign evaluation	BOOL	Q, M, N

**5.2.26 ESIGN: Floating-point sign evaluation**

◆ **Function description**

This function block is used for the detection of negative sign. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

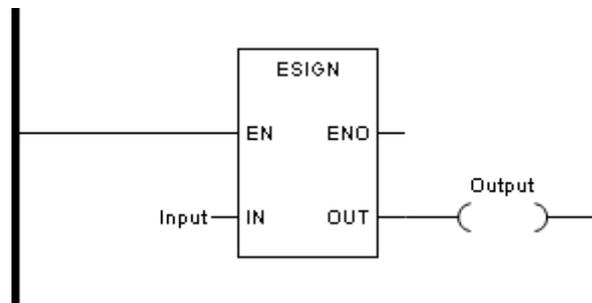
With a value ≥ 0 at the IN input, the OUT output becomes “0”. With a value < 0 at the IN input, the OUT output becomes “1”.

◆ **Formula**

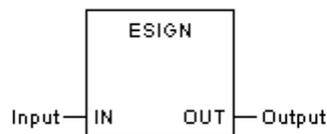
OUT = 1, if IN < 0,

OUT = 0, if IN ≥ 0.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Signed input	REAL	Constant, MW, NW
OUT	Output	Sign evaluation	BOOL	Q, M, N

**5.2.27 ABS: 16-bit absolute value**

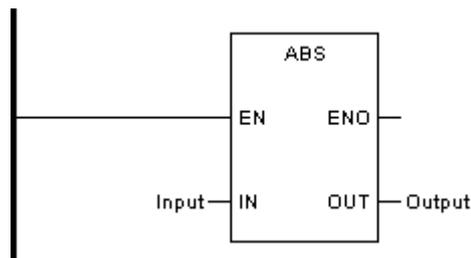
◆ **Function description**

This function block computes the absolute value of the input value and assigns the result to the output.

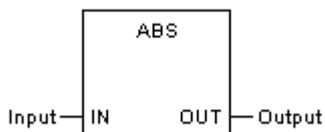
◆ **Formula**

$$OUT = |IN|$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input  
 ABS  
 ST            Output

◆ **Representation in ST**

Output := ABS (Input );

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Absolute value output	INT	MW, NW

**5.2.28 DABS: 32-bit absolute value**

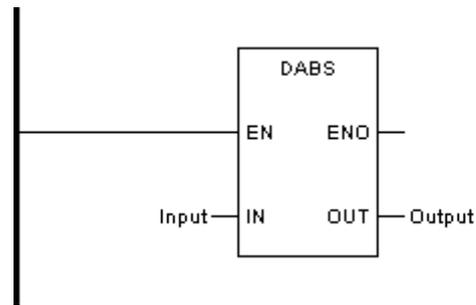
◆ **Function description**

This function block computes the absolute value of the input value and assigns the result to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

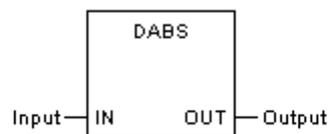
◆ **Formula**

$$OUT = |IN|$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	DINT	Constant, MW, NW
OUT	Output	Absolute value output	DINT	MW, NW

**5.2.29 EABS: Floating-point absolute value**

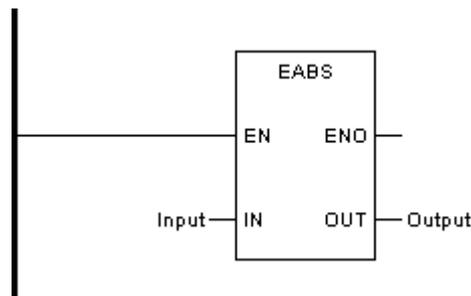
◆ **Function description**

This function block computes the absolute value of the input value and assigns the result to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

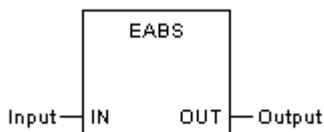
◆ **Formula**

$$OUT = |IN|$$

◆ **Representation in LD**



◆ Representation in FBD



◆ Parameter description

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW
OUT	Output	Absolute value output	REAL	MW, NW

### 5.2.30 SQRT: Floating-point square root

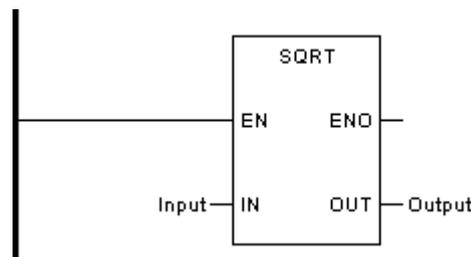
◆ Function description

This function block extracts the square root from the input and assigns the result to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

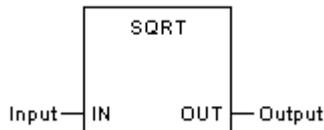
◆ Formula

$$OUT = \sqrt{IN}$$

◆ Representation in LD



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW
OUT	Output	Square root output	REAL	MW, NW

**5.2.31 LOG: Floating-point decimal logarithm**

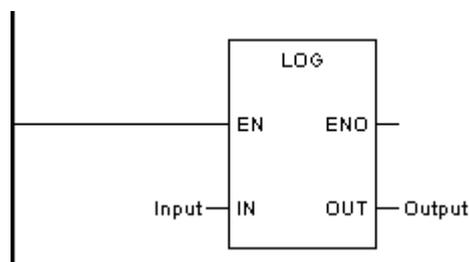
◆ **Function description**

This function block calculates the base 10 logarithm of the input and assigns the result to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

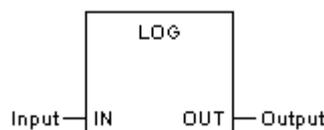
◆ **Formula**

$$OUT = \log_{10}^{IN}$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW
OUT	Output	Base 10 logarithm output	REAL	MW, NW

**5.2.32 LN: Floating-point natural logarithm**

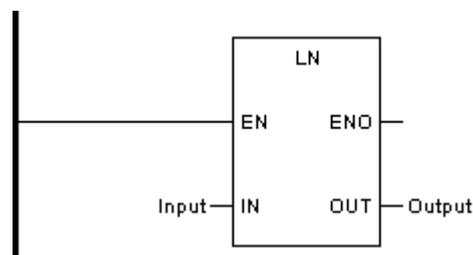
◆ **Function description**

This function block calculates the natural logarithm of the input and assigns the result to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

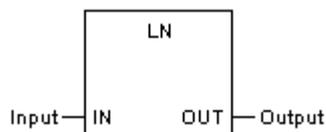
◆ **Formula**

$$OUT = \log_e^{IN}, e=2.718282$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW

OUT	Output	Natural logarithm output	REAL	MW, NW
-----	--------	--------------------------	------	--------

### 5.2.33 EXP: Floating-point natural exponentiation

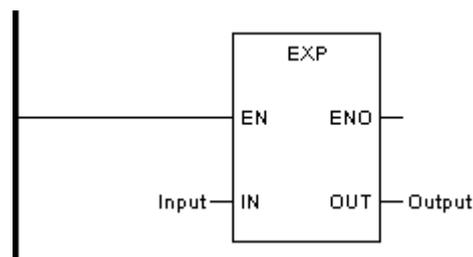
◆ **Function description**

This function block calculates the natural exponentiation of the input and assigns the result to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

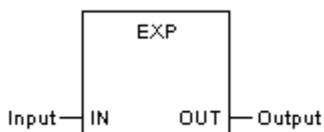
◆ **Formula**

$$OUT = e^{IN}, e=2.718282$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW
OUT	Output	Natural exponentiation output	REAL	MW, NW

### 5.2.34 EXPT: Floating-point exponentiation

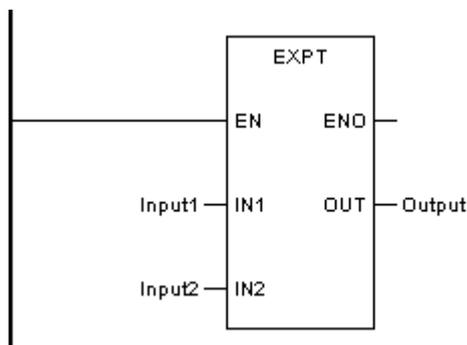
◆ **Function description**

This function block calculates the exponentiation of one value by another value and assigns the result to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

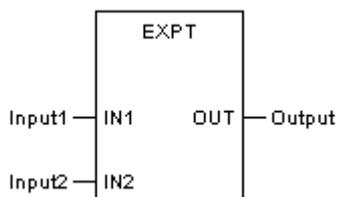
◆ **Formula**

$$OUT = IN1^{IN2}$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Base	REAL	Constant, MW, NW
IN2	Input2	Exponent	REAL	Constant, MW, NW
OUT	Output	Exponentiation output	REAL	MW, NW

### 5.2.35 SIN: Floating-point sine

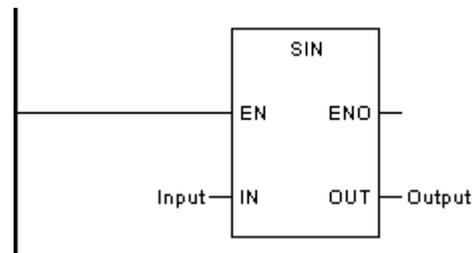
◆ **Function description**

This function block calculates the sine of an angle input and assigns the result to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

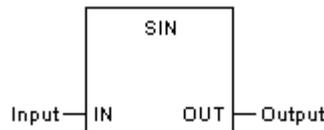
◆ **Formula**

$$\text{OUT} = \sin \text{IN}$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW
OUT	Output	Sine output	REAL	MW, NW

### 5.2.36 COS: Floating-point cosine

◆ **Function description**

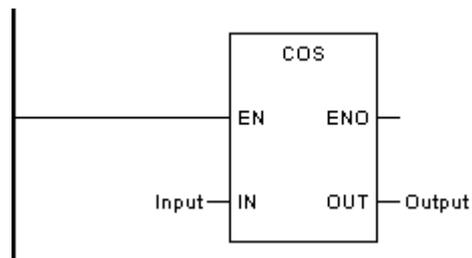
This function block calculates the cosine of an angle input and assigns the result to the output. Input and output can only be floating-point register, occupying 2

continuous word registers. The serial number of word register must be odd.

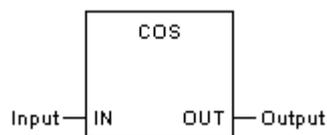
◆ **Formula**

$$OUT = \cos IN$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW
OUT	Output	Cosine output	REAL	MW, NW

### 5.2.37 TAN: Floating-point tangent

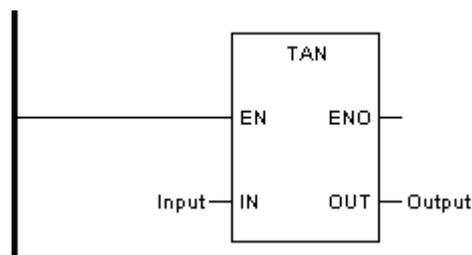
◆ **Function description**

This function block calculates the tangent of an angle input and assigns the result to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

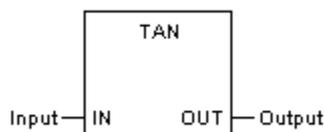
◆ **Formula**

$$OUT = \tan IN$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW
OUT	Output	Tangent output	REAL	MW, NW

### 5.2.38 ASIN: Floating-point arc sine

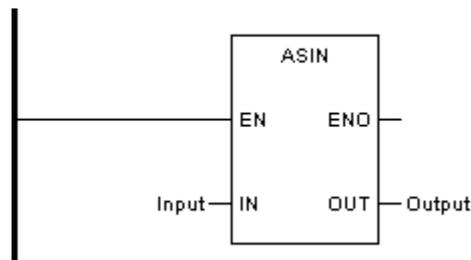
◆ **Function description**

This function block calculates the principal arc sine of the input and assigns the result to the output in the form of an angle in radians. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

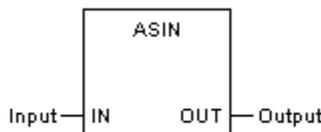
◆ **Formula**

$$OUT = \text{asin } IN$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW
OUT	Output	Arc sine output	REAL	MW, NW

**5.2.39 ACOS: Floating-point arc cosine**

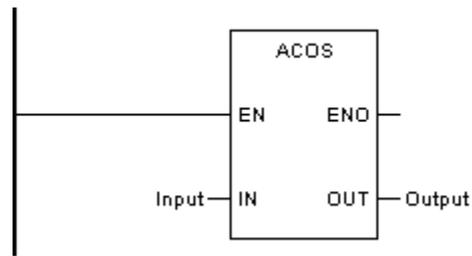
◆ **Function description**

This function block calculates the principal arc cosine of the input and assigns the result to the output in the form of an angle in radians. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

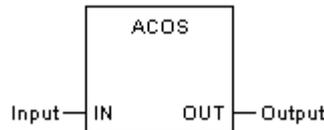
◆ **Formula**

$$OUT = \text{acos } IN$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW
OUT	Output	Arc cosine output	REAL	MW, NW

## 5.2.40 ATAN: Floating-point arc tangent

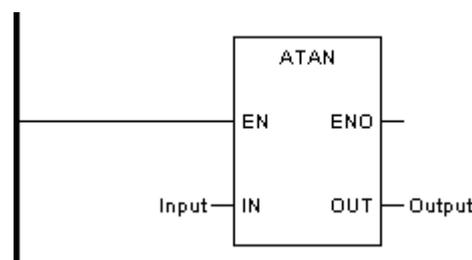
◆ **Function description**

This function block calculates the principal arc tangent of the input and assigns the result to the output in the form of an angle in radians. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

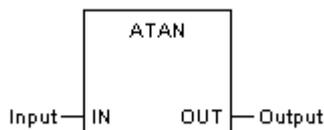
◆ **Formula**

$$OUT = \text{atan } IN$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input	REAL	Constant, MW, NW
OUT	Output	Arc tangent output	REAL	MW, NW

## 5.3 Statistics

This chapter describes the elementary function blocks of the Statistics family. This chapter contains the following sections.

Type	Description	Formula
<b>MIN</b>	16-Bit Minimum Value	OUT = MIN {IN1, IN2, ... , INn} (n≤8)
<b>DMIN</b>	32-Bit Minimum Value	
<b>EMIN</b>	Floating-Point Minimum Value	
<b>MAX</b>	16-Bit Maximum Value	OUT = MAX { IN1, IN2, ... , INn} (n≤8)
<b>DMAX</b>	32-Bit Maximum Value	
<b>EMAX</b>	Floating-Point Maximum Value	
<b>AVE</b>	16-Bit Averaging	OUT = $\frac{IN1 + IN2 + \dots + INn}{n}$ (n≤8)
<b>DAVE</b>	32-Bit Averaging	
<b>EAVE</b>	Floating-Point Averaging	
<b>LIMIT</b>	16-Bit Limit	OUT = IN, if (IN ≥ MN) & (IN ≤ MX)
<b>DLIMIT</b>	32-Bit Limit	OUT = MN, if (IN < MN)
<b>ELIMIT</b>	Floating-Point Limit	OUT = MX, if (IN > MX)
<b>SEL</b>	16-Bit 0/1 Selection	G = 0 -> OUT = IN0
<b>DSEL</b>	32-Bit 0/1 Selection	G = 1 -> OUT = IN1

<b>ESEL</b>	Floating-Point 0/1 Selection	
<b>MUX</b>	16-Bit Multiplexer	K = 0 -> OUT = IN0 K = 1 -> OUT = IN1 K = 2 -> OUT = IN2 ..... K = n -> OUT = INn (n≤6)
<b>DMUX</b>	32-Bit Multiplexer	
<b>EMUX</b>	Floating-Point Multiplexer	
<b>LMT</b>	16-Bit Limit Detection	
<b>DLMT</b>	32-Bit Limit Detection	OUT = 1, if (IN ≥ MN) & (IN ≤ MX) OUT = 0, if (IN < MN)   (IN > MX)
<b>ELMT</b>	Floating-Point Limit Detection	

### 5.3.1 MIN: 16-bit minimum value

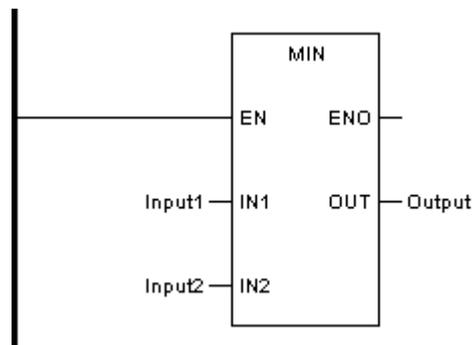
#### ◆ Function description

This function block assigns the smallest input value to the output.  
 The number of inputs can be increased to a maximum of 8.

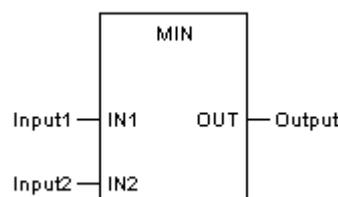
#### ◆ Formula

$$OUT = \text{MIN} \{IN1, IN2, \dots, INn\} \quad (n \leq 8)$$

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

LD	Input1
MIN	Input2
ST	Output

◆ **Representation in ST**

Output := MIN (Input1, Input2);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Input 2	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Minimum value output	INT	MW, NW

**5.3.2 DMIN: 32-bit minimum value**

◆ **Function description**

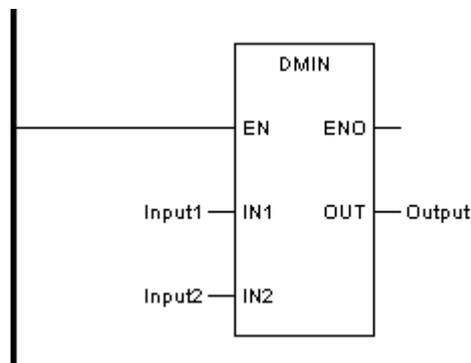
This function block assigns the smallest input value to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

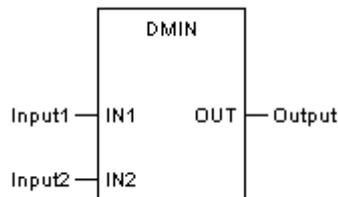
◆ **Formula**

$$OUT = \text{MIN} \{IN1, IN2, \dots, INn\} \quad (n \leq 8)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	DINT	Constant, MW, NW
IN2	Input2	Input 2	DINT	Constant, MW, NW
OUT	Output	Minimum value output	DINT	MW, NW

### 5.3.3 EMIN: Floating-point minimum value

◆ **Function description**

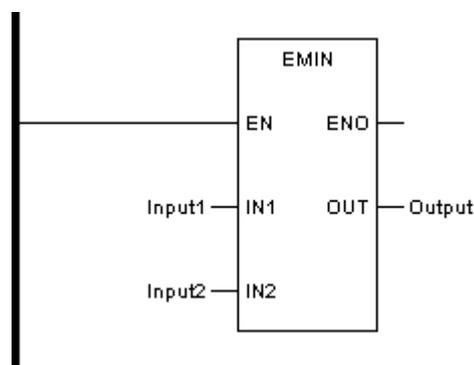
This function block assigns the smallest input value to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

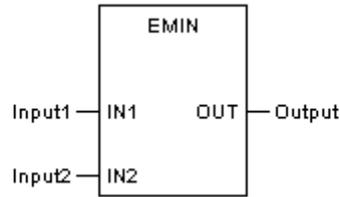
◆ **Formula**

$$OUT = \text{MIN} \{IN1, IN2, \dots, INn\} \quad (n \leq 8)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	REAL	Constant, MW, NW
IN2	Input2	Input 2	REAL	Constant, MW, NW
OUT	Output	Minimum value output	REAL	MW, NW

**5.3.4 MAX: 16-bit maximum value**

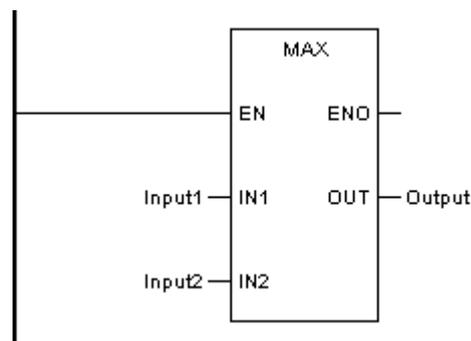
◆ **Function description**

This function block assigns the largest input value to the output.  
 The number of inputs can be increased to a maximum of 8.

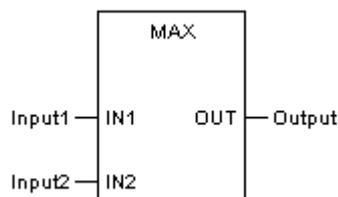
◆ **Formula**

$$OUT = \text{MAX} \{IN1, IN2, \dots, INn\} \quad (n \leq 8)$$

◆ **Representation in LD**



Representation in FBD



Representation in IL

```
LD      Input1
MAX     Input2
ST      Output
```

Representation in ST

```
Output := MAX ( Input1, Input2);
```

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Input 2	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Maximum value output	INT	MW, NW

### 5.3.5 DMAX: 32-bit maximum value

◆ **Function description**

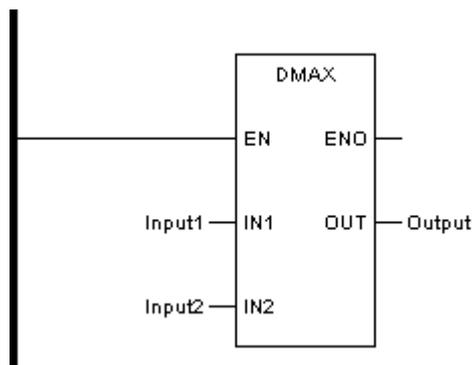
This function block assigns the largest input value to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

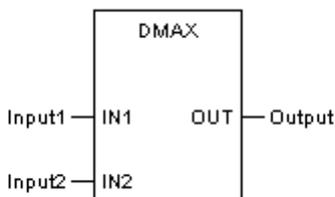
◆ **Formula**

$OUT = MAX \{IN1, IN2, \dots, INn\} \quad (n \leq 8)$

◆ **Representation in LD**



Representation in FBD



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	DINT	Constant, MW, NW
IN2	Input2	Input 2	DINT	Constant, MW, NW
OUT	Output	Maximum value output	DINT	MW, NW

**5.3.6 EMAX: Floating-point maximum value**

◆ **Function description**

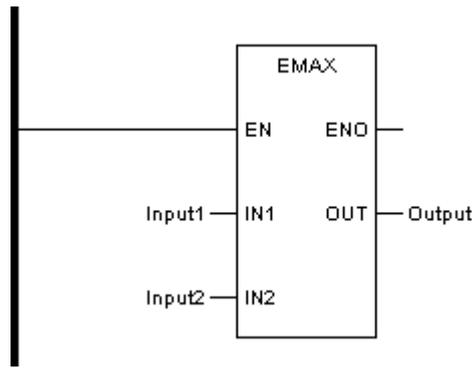
This function block assigns the largest input value to the output. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

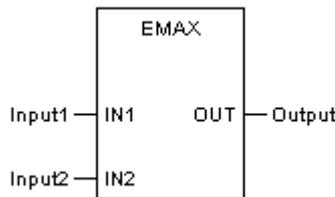
◆ **Formula**

$$OUT = \text{MAX} \{IN1, IN2, \dots, INn\} \quad (n \leq 8)$$

◆ **Representation in LD**



Representation in FBD



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	REAL	Constant, MW, NW
IN2	Input2	Input 2	REAL	Constant, MW, NW
OUT	Output	Maximum value output	REAL	MW, NW

**5.3.7 AVE: 16-bit averaging**

◆ **Function description**

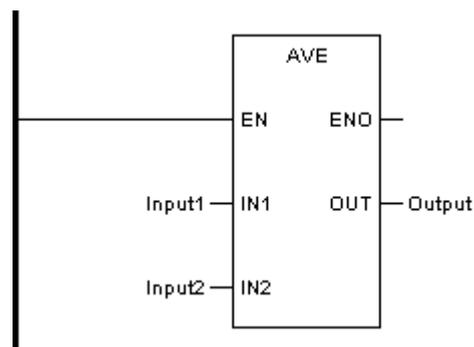
This function block calculates the average of the input values and assigns the result to the output.

The number of inputs can be increased to a maximum of 8.

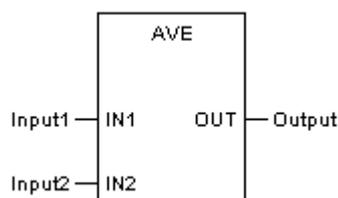
◆ **Formula**

$$OUT = \frac{IN1 + IN2 + \dots + INn}{n} \quad (n \leq 8)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

```

LD      Input1
AVE     Input2
ST      Output
    
```

◆ **Representation in ST**

```

Output := AVE (Input1, Input2);
    
```

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Input 2	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Average value output	INT	MW, NW

### 5.3.8 DAVE: 32-bit averaging

◆ **Function description**

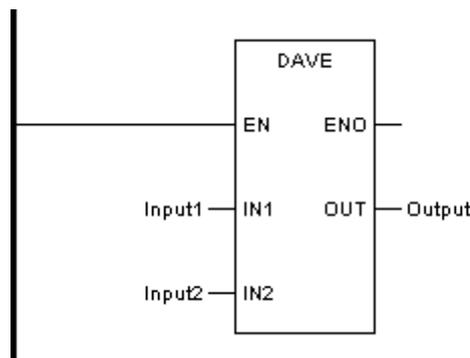
This function block calculates the average of the input values and assigns the result to the output. Input and output can only be 32-bit register, occupying 2 continuous word

registers. The serial number of word register must be odd.  
 The number of inputs can be increased to a maximum of 8.

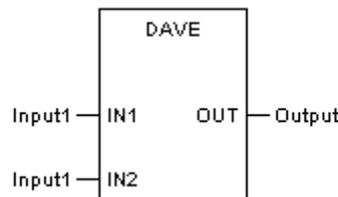
◆ **Formula**

$$OUT = \frac{IN1 + IN2 + \dots + INn}{n} \quad (n \leq 8)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	DINT	Constant, MW, NW
IN2	Input2	Input 2	DINT	Constant, MW, NW
OUT	Output	Average value output	DINT	MW, NW

### 5.3.9 EAVE: Floating-point averaging

◆ **Function description**

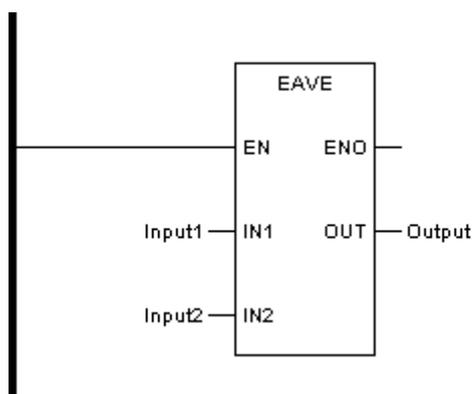
This function block calculates the average of the input values and assigns the result to the output. Input and output can only be floating-point register, occupying 2

continuous word registers. The serial number of word register must be odd.  
 The number of inputs can be increased to a maximum of 8.

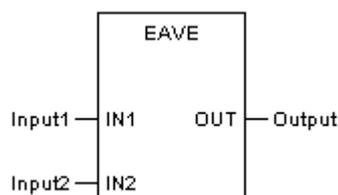
◆ **Formula**

$$OUT = \frac{IN1 + IN2 + \dots + INn}{n} \quad (n \leq 8)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	REAL	Constant, MW, NW
IN2	Input2	Input 2	REAL	Constant, MW, NW
OUT	Output	Average value output	REAL	MW, NW

**5.3.10 LIMIT: 16-bit limit**

◆ **Function description**

This function block transfers the unchanged input value (IN) to the output if the input value is not less than the minimum value (MN) and is not greater than the maximum value (MX).

If the input value (IN) is less than the minimum value (MN), the minimum value is transferred to the output.

If the input value (IN) is greater than the maximum value (MX), the maximum value is transferred to the output.

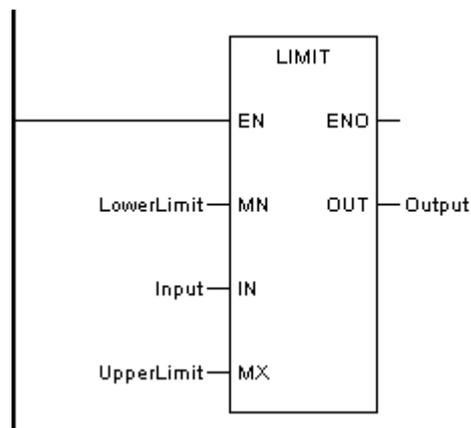
◆ **Formula**

OUT = IN, if  $(IN \geq MN) \ \& \ (IN \leq MX)$

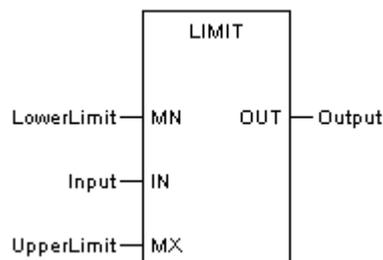
OUT = MN, if  $(IN < MN)$

OUT = MX, if  $(IN > MX)$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD	LowerLimit
LIMIT	Input, UpperLimit
ST	Output

### ◆ Representation in ST

Output := LIMIT (LowerLimit, Input, UpperLimit);

### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
MN	LowerLimit	Minimum value	INT	Constant, IW, QW, MW, NW, SW
IN	Input	Input	INT	Constant, IW, QW, MW, NW, SW
MX	UpperLimit	Maximum value	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output	INT	MW, NW

## 5.3.11 DLIMIT: 32-bit limit

### ◆ Function description

This function block transfers the unchanged input value (IN) to the output if the input value is not less than the minimum value (MN) and is not greater than the maximum value (MX).

If the input value (IN) is less than the minimum value (MN), the minimum value is transferred to the output.

If the input value (IN) is greater than the maximum value (MX), the maximum value is transferred to the output.

Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

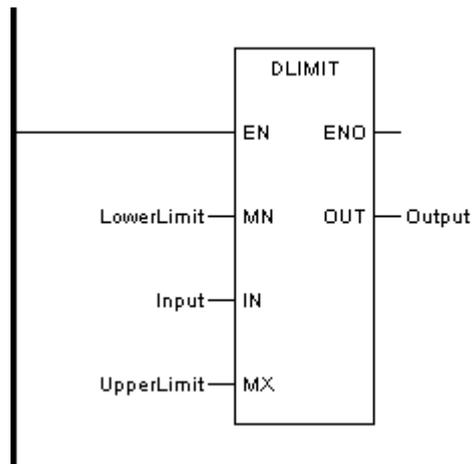
### ◆ Formula

OUT = IN, if (IN ≥ MN) & (IN ≤ MX)

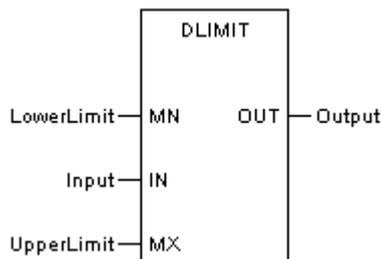
OUT = MN, if (IN < MN)

OUT = MX, if (IN > MX)

### ◆ Representation in LD



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
MN	LowerLimit	Minimum value	DINT	Constant, MW, NW
IN	Input	Input	DINT	Constant, MW, NW
MX	UpperLimit	Maximum value	DINT	Constant, MW, NW
OUT	Output	Output	DINT	MW, NW

**5.3.12 ELIMIT: Floating-point limit**

◆ **Function description**

This function block transfers the unchanged input value (IN) to the output if the input value is not less than the minimum value (MN) and is not greater than the maximum value (MX).

If the input value (IN) is less than the minimum value (MN), the minimum value is transferred to the output.

If the input value (IN) is greater than the maximum value (MX), the maximum value is transferred to the output.

Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

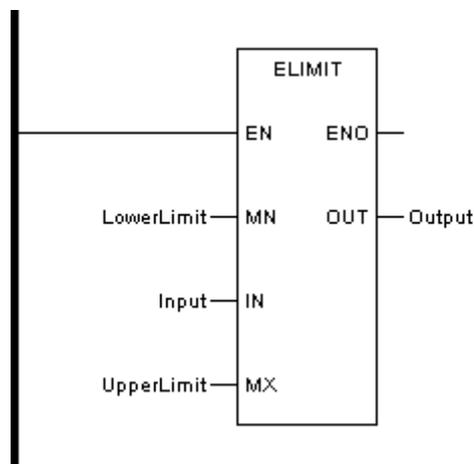
◆ **Formula**

OUT = IN, if (IN ≥ MN) & (IN ≤ MX)

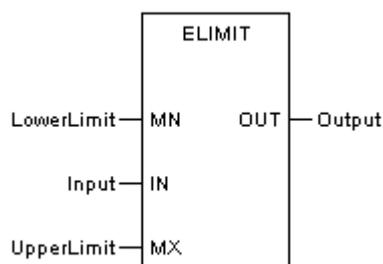
OUT = MN, if (IN < MN)

OUT = MX, if (IN > MX)

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
MN	LowerLimit	Minimum value	REAL	Constant, MW, NW
IN	Input	Input	REAL	Constant, MW, NW
MX	UpperLimit	Maximum value	REAL	Constant, MW, NW

OUT	Output	Output	REAL	MW, NW
-----	--------	--------	------	--------

### 5.3.13 SEL: 16-bit 0/1 selection

#### ◆ Function description

This function block is used for binary selection between two input values.

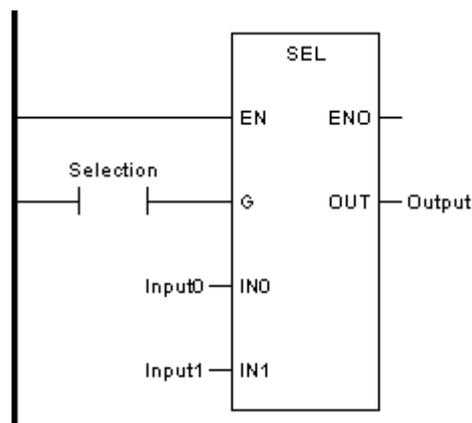
Depending on the state of the G input, either the IN0 input or IN1 input is transferred to the OUT output.

#### ◆ Formula

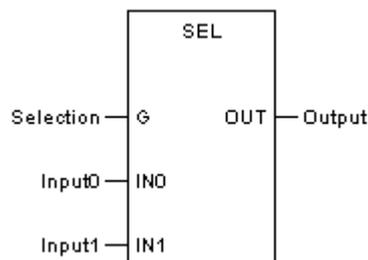
$G = 0 \rightarrow OUT = IN0$

$G = 1 \rightarrow OUT = IN1$

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

LD                    Selection  
 SEL                 Input0, Input1

ST                      Output

#### ◆ Representation in ST

Output := SEL (Selection, Input0, Input1);

#### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
G	Selection	Selection input	BOOL	Constant, I, Q, M, N, S
IN0	Input0	Input 0	INT	Constant, IW, QW, MW, NW, SW
IN1	Input1	Input 1	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output	INT	MW, NW

### 5.3.14 DSEL: 32-bit 0/1 selection

#### ◆ Function description

This function block is used for binary selection between two input values.

Depending on the state of the G input, either the IN0 input or IN1 input is transferred to the OUT output.

Input and output can only be 32-bit register, occupying 2 continuous word registers.

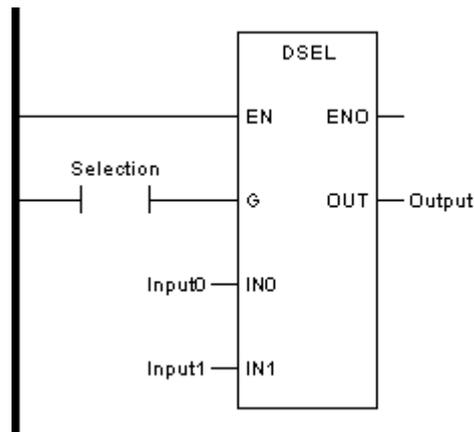
The serial number of word register must be odd.

#### ◆ Formula

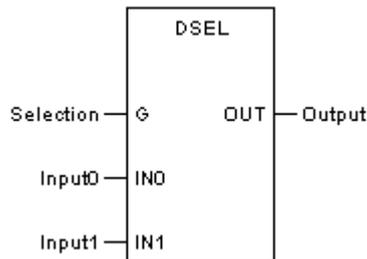
G = 0 -> OUT = IN0

G = 1 -> OUT = IN1

#### ◆ Representation in LD



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
G	Selection	Selection input	BOOL	Constant, I, Q, M, N, S
IN0	Input0	Input 0	DINT	Constant, MW, NW
IN1	Input1	Input 1	DINT	Constant, MW, NW
OUT	Output	Output	DINT	MW, NW

**5.3.15 ESEL: Floating-point 0/1 selection**

◆ **Function description**

This function block is used for binary selection between two input values.

Depending on the state of the G input, either the IN0 input or IN1 input is transferred to the OUT output.

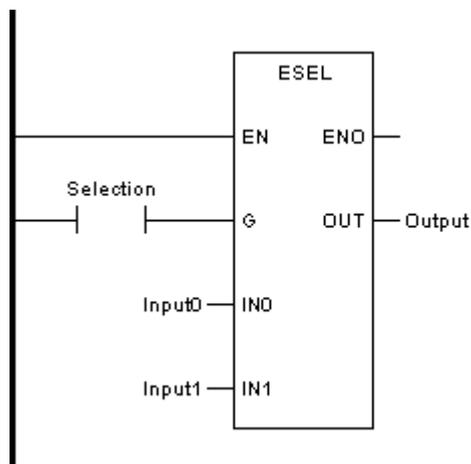
Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Formula**

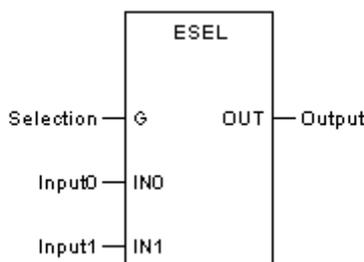
$G = 0 \rightarrow OUT = IN0$

$G = 1 \rightarrow OUT = IN1$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
G	Selection	Selection input	BOOL	Constant, I, Q, M, N, S
IN0	Input0	Input 0	REAL	Constant, MW, NW
IN1	Input1	Input 1	REAL	Constant, MW, NW
OUT	Output	Output	REAL	MW, NW

**5.3.16 MUX: 16-bit multiplexer**

◆ **Function description**

This function block transfers the respective input to the output depending on the value of K input.

The number of inputs can be increased to a maximum of 7.

◆ **Formula**

$K = 0 \rightarrow OUT = IN0$

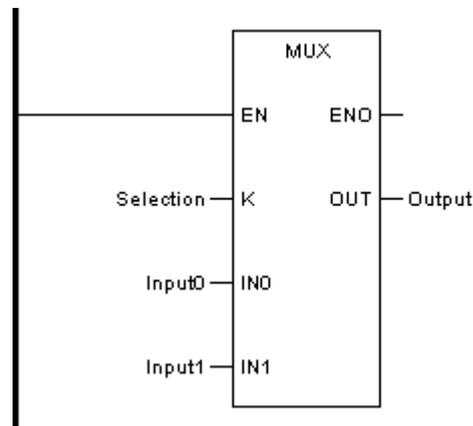
$K = 1 \rightarrow OUT = IN1$

$K = 2 \rightarrow OUT = IN2$

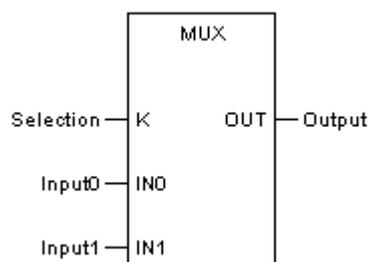
.....

$K = n \rightarrow OUT = INn \quad (n \leq 6)$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD	Selection
MUX	Input0, Input1
ST	Output

◆ **Representation in ST**

Output := MUX (Selection, Input0, Input1);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
K	Selection	Selection input	INT	Constant, IW, QW, MW, NW, SW
IN0	Input0	Input 0	INT	Constant, IW, QW, MW, NW, SW
IN1	Input1	Input 1	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output	INT	MW, NW

**5.3.17 DMUX: 32-bit multiplexer**

◆ **Function description**

This function block transfers the respective input to the output depending on the value of K input. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 7.

◆ **Formula**

$K = 0 \rightarrow OUT = IN0$

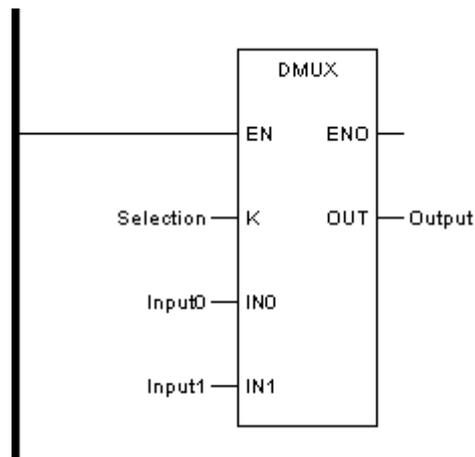
$K = 1 \rightarrow OUT = IN1$

$K = 2 \rightarrow OUT = IN2$

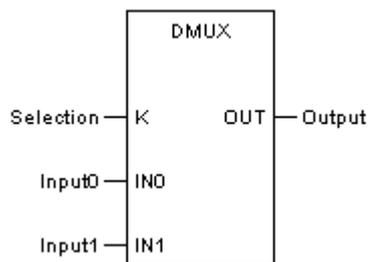
.....

$K = n \rightarrow OUT = INn \quad (n \leq 6)$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
K	Selection	Selection input	INT	Constant, IW, QW, MW, NW, SW
IN0	Input0	Input 0	DINT	Constant, MW, NW
IN1	Input1	Input 1	DINT	Constant, MW, NW
OUT	Output	Output	DINT	MW, NW

**5.3.18 EMUX: Floating-point multiplexer**

◆ **Function description**

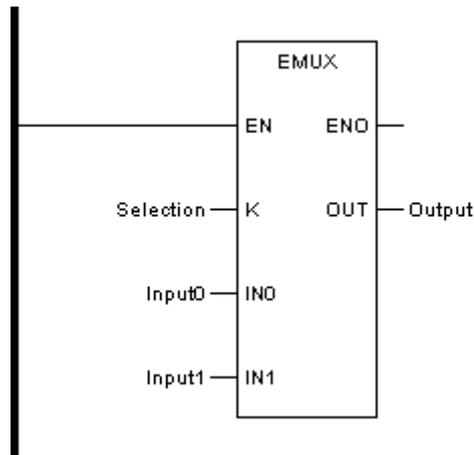
This function block transfers the respective input to the output depending on the value of K input. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 7.

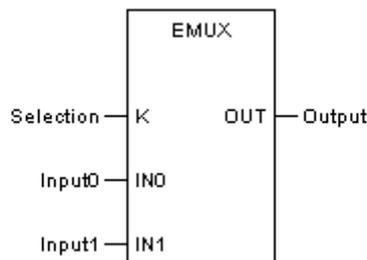
◆ **Formula**

K = 0 -> OUT = IN0  
 K = 1 -> OUT = IN1  
 K = 2 -> OUT = IN2  
 .....  
 K = n -> OUT = INn (n ≤ 6)

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
K	Selection	Selection input	INT	Constant, IW, QW, MW, NW, SW
IN0	Input0	Input 0	REAL	Constant, MW, NW
IN1	Input1	Input 1	REAL	Constant, MW, NW
OUT	Output	Output	REAL	MW, NW

### 5.3.19 LMT: 16-bit limit detection

#### ◆ Function description

The output value (OUT) is 1 if the input value (IN) is not less than the minimum value (MN) and is not greater than the maximum value (MX).

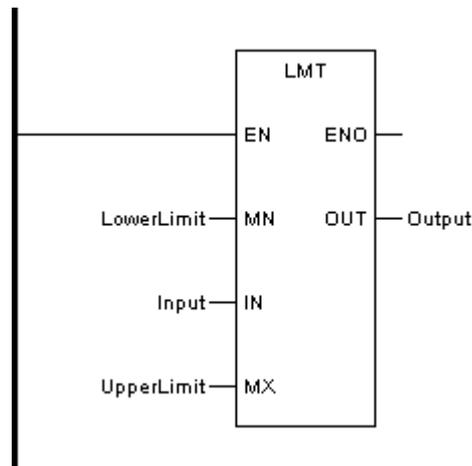
The output value (OUT) is 0 if the input value (IN) is less than the minimum value (MN) or greater than the maximum value (MX).

#### ◆ Formula

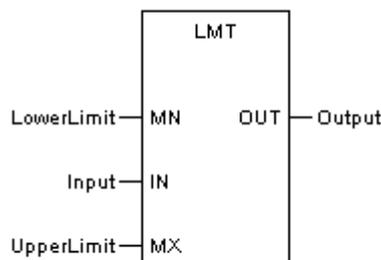
OUT = 1, if (IN ≥ MN) & (IN ≤ MX)

OUT = 0, if (IN < MN) | (IN > MX)

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

LD

LowerLimit

LMT	Input, UpperLimit
ST	Output

### ◆ Representation in ST

Output := LMT (LowerLimit, Input, UpperLimit);

### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
MN	LowerLimit	Minimum value	INT	Constant, IW, QW, MW, NW, SW
IN	Input	Input	INT	Constant, IW, QW, MW, NW, SW
MX	UpperLimit	Maximum value	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output	BOOL	M, N, Q

## 5.3.20 DLMT: 32-bit limit detection

### ◆ Function description

The output value (OUT) is 1 if the input value (IN) is not less than the minimum value (MN) and is not greater than the maximum value (MX).

The output value (OUT) is 0 if the input value (IN) is less than the minimum value (MN) or greater than the maximum value (MX).

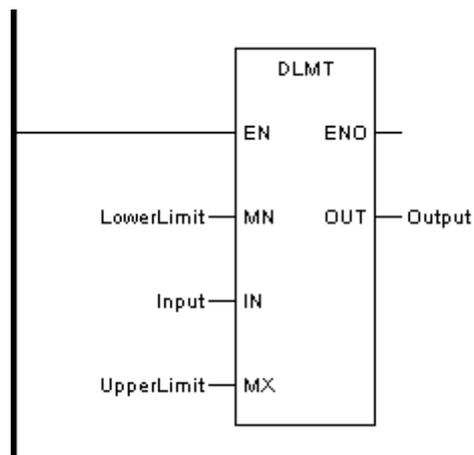
Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

### ◆ Formula

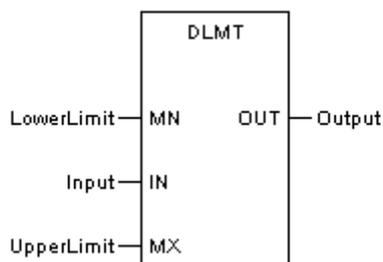
OUT = 1, if (IN ≥ MN) & (IN ≤ MX)

OUT = 0, if (IN < MN) | (IN > MX)

### ◆ Representation in LD



◆ Representation in FBD



◆ Parameter description

Icon	Parameter	Description	Data type	Point type
MN	LowerLimit	Minimum value	DINT	Constant, MW, NW
IN	Input	Input	DINT	Constant, MW, NW
MX	UpperLimit	Maximum value	DINT	Constant, MW, NW
OUT	Output	Output	BOOL	M, N, Q

**5.3.21 ELMT: Floating-point limit detection**

◆ Function description

The output value (OUT) is 1 if the input value (IN) is not less than the minimum value (MN) and is not greater than the maximum value (MX).

The output value (OUT) is 0 if the input value (IN) is less than the minimum value (MN) or greater than the maximum value (MX).

Input and output can only be floating-point register, occupying 2 continuous word

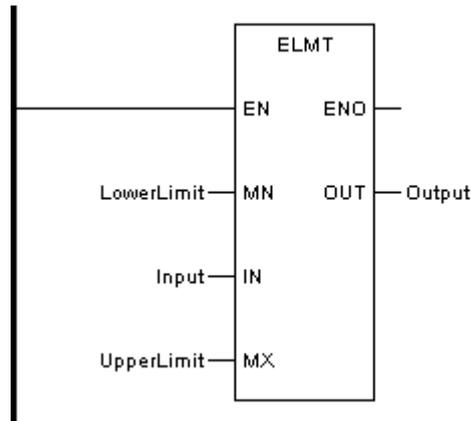
registers. The serial number of word register must be odd.

◆ **Formula**

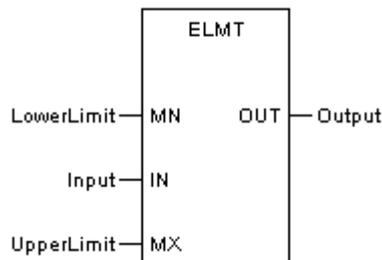
OUT = 1, if (IN ≥ MN) & (IN ≤ MX)

OUT = 0, if (IN < MN) | (IN > MX)

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
MN	LowerLimit	Minimum value	REAL	Constant, MW, NW
IN	Input	Input	REAL	Constant, MW, NW
MX	UpperLimit	Maximum value	REAL	Constant, MW, NW
OUT	Output	Output	BOOL	M, N, Q

## 5.4 Logic

This chapter describes the elementary function blocks of the Logic family.

This chapter contains the following sections.

Type	Description
<b>AND</b>	16-Bit AND
<b>DAND</b>	32-Bit AND
<b>OR</b>	16-Bit OR
<b>DOR</b>	32-Bit OR
<b>NOT</b>	16-Bit Negation
<b>DNOT</b>	32-Bit Negation
<b>XOR</b>	16-Bit Exclusive OR
<b>DXOR</b>	32-Bit Exclusive OR
<b>SHL</b>	16-Bit Shift Left
<b>DSHL</b>	32-Bit Shift Left
<b>SHR</b>	16-Bit Shift Right
<b>DSHR</b>	32-Bit Shift Right
<b>ROL</b>	16-Bit Rotate Left
<b>DROL</b>	32-Bit Rotate Left
<b>ROR</b>	16-Bit Rotate Right
<b>DROR</b>	32-Bit Rotate Right
<b>BSET</b>	Bit Set
<b>BCLR</b>	Bit Clear
<b>BTST</b>	Bit Test
<b>R_TRIG</b>	Rising Edge Detection
<b>F_TRIG</b>	Falling Edge Detection
<b>SET</b>	Set
<b>RESET</b>	Reset
<b>SR</b>	Bistable (Set Dominant)

<b>RS</b>	Bistable (Reset Dominant)
-----------	---------------------------

### 5.4.1 AND: 16-bit AND

#### ◆ Function description

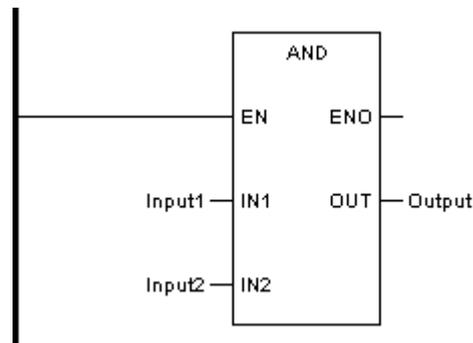
This function block for a bit-by-bit AND link of the bit sequences at the inputs and assigns the result to the output.

The number of inputs can be increased to a maximum of 8.

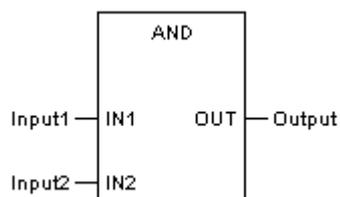
#### ◆ Formula

$$\text{OUT} = \text{IN1 AND IN2 AND } \dots \text{ AND INn}$$

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

LD	Input1
AND	Input2
ST	Output

#### ◆ Representation in ST

Output := AND (Input1, Input2);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input bit sequence	BOOL, INT	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
IN2	Input2	Input bit sequence	BOOL, INT	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
OUT	Output	Output bit sequence	BOOL, INT	Q, M, N, MW, NW

**5.4.2 DAND: 32-bit AND**

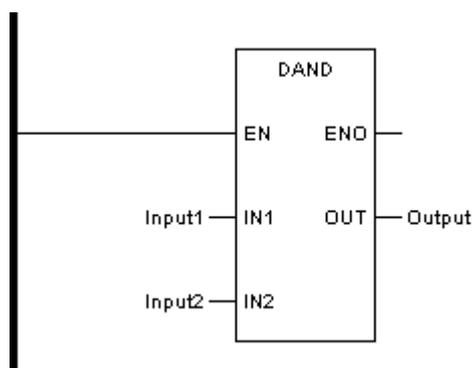
◆ **Function description**

This function block for a bit-by-bit AND link of the bit sequences at the inputs and assigns the result to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd. The number of inputs can be increased to a maximum of 8.

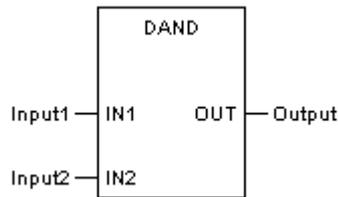
◆ **Formula**

$$OUT = IN1 \text{ AND } IN2 \text{ AND } \dots \text{ AND } INn$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input bit sequence	DINT	Constant, MW, NW
IN2	Input2	Input bit sequence	DINT	Constant, MW, NW
OUT	Output	Output bit sequence	DINT	MW, NW

### 5.4.3 OR: 16-bit OR

◆ **Function description**

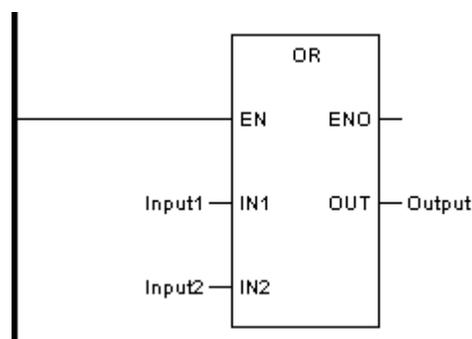
This function block for a bit-by-bit OR link of the bit sequences at the inputs and assigns the result to the output.

The number of inputs can be increased to a maximum of 8.

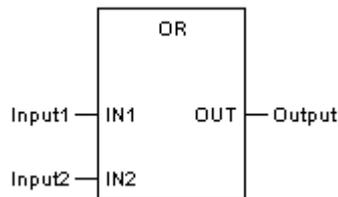
◆ **Formula**

$$OUT = IN1 \text{ OR } IN2 \text{ OR } \dots \text{ OR } INn$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input1  
 OR           Input2  
 ST           Output

◆ **Representation in ST**

Output := OR (Input1, Input2);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input bit sequence	BOOL, INT	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
IN2	Input2	Input bit sequence	BOOL, INT	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
OUT	Output	Output bit sequence	BOOL, INT	Q, M, N, MW, NW

**5.4.4 DOR: 32-bit OR**

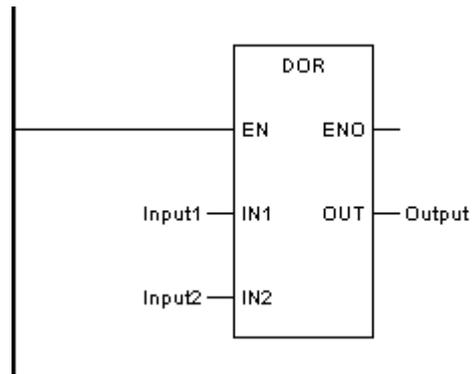
◆ **Function description**

This function block for a bit-by-bit OR link of the bit sequences at the inputs and assigns the result to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd. The number of inputs can be increased to a maximum of 8.

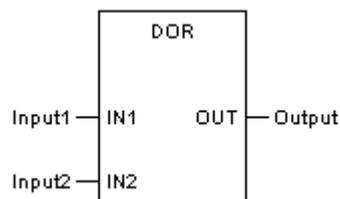
◆ **Formula**

OUT = IN1 OR IN2 OR ... OR INn

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input bit sequence	DINT	Constant, MW, NW
IN2	Input2	Input bit sequence	DINT	Constant, MW, NW
OUT	Output	Output bit sequence	DINT	MW, NW

**5.4.5 NOT: 16-bit negation**

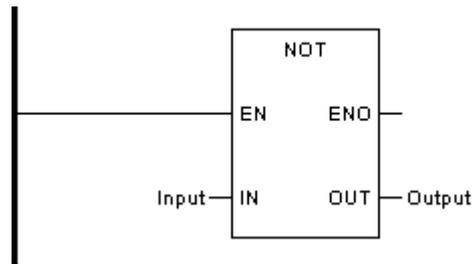
◆ **Function description**

This function block negates the input bit sequence bit-by-bit and assigns the result to the output.

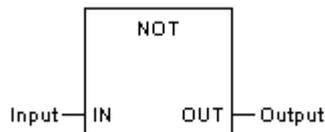
◆ **Formula**

$$OUT = NOT\ IN$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input  
 NOT  
 ST            Output

◆ **Representation in ST**

Output := NOT (Input);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input bit sequence	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output bit sequence	INT	MW, NW

**5.4.6 DNOT: 32-bit negation**

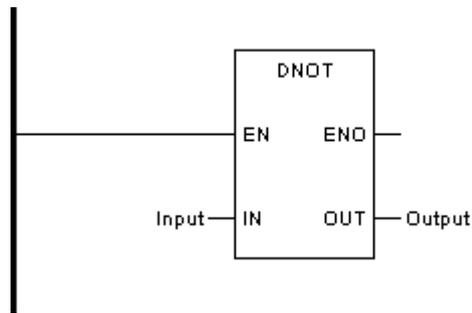
◆ **Function description**

This function block negates the input bit sequence bit-by-bit and assigns the result to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

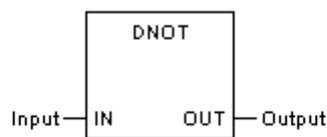
◆ **Formula**

OUT = NOT IN

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input bit sequence	DINT	Constant, MW, NW
OUT	Output	Output bit sequence	DINT	MW, NW

**5.4.7 XOR: 16-bit exclusive OR**

◆ **Function description**

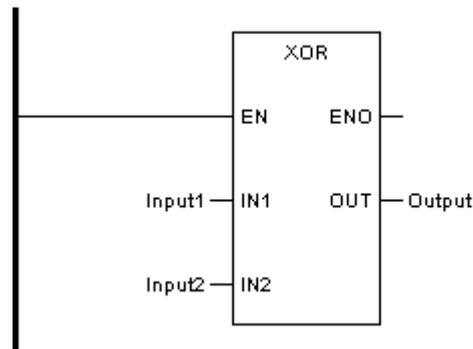
This function block for a bit-by-bit XOR link of the bit sequences at the inputs and assigns the result to the output.

The number of inputs can be increased to a maximum of 8.

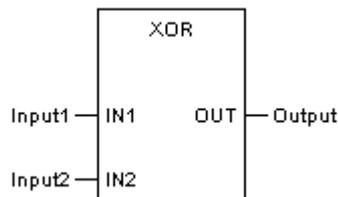
◆ **Formula**

$$OUT = IN1 \text{ XOR } IN2 \text{ XOR } \dots \text{ XOR } INn$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input1  
 XOR         Input2  
 ST           Output

◆ **Representation in ST**

Output := XOR (Input1, Input2);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input bit sequence	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Input bit sequence	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output bit sequence	INT	MW, NW

## 5.4.8 DXOR: 32-bit exclusive OR

◆ **Function description**

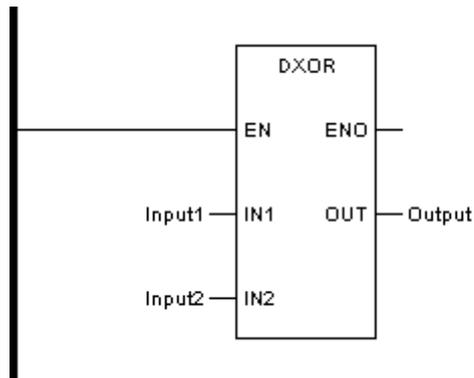
This function block for a bit-by-bit XOR link of the bit sequences at the inputs and assigns the result to the output. Input and output can only be 32-bit register, occupying

2 continuous word registers. The serial number of word register must be odd.  
The number of inputs can be increased to a maximum of 8.

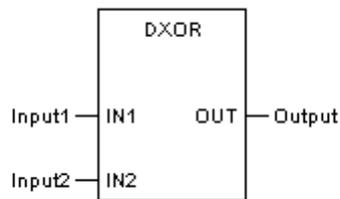
◆ **Formula**

$$\text{OUT} = \text{IN1 XOR IN2 XOR } \dots \text{ XOR INn}$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

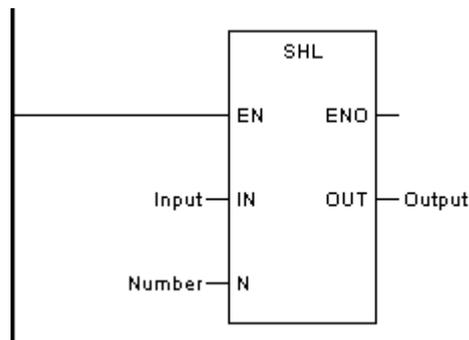
Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input bit sequence	DINT	Constant, MW, NW
IN2	Input2	Input bit sequence	DINT	Constant, , MW, NW
OUT	Output	Output bit sequence	DINT	MW, NW

### 5.4.9 SHL: 16-bit shift left

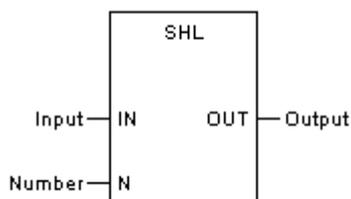
◆ **Function description**

This function block shifts the bit pattern at the IN input to the left by N bits. Zeros are filled in from the right. The result is assigned to the output.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input  
 SHL         Number  
 ST           Output

◆ **Representation in ST**

Output := SHL (Input, Number);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Bit pattern to be shifted	INT	Constant, IW, QW, MW, NW, SW
N	Number	Number to be shifted	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Bit pattern shifted	INT	MW, NW

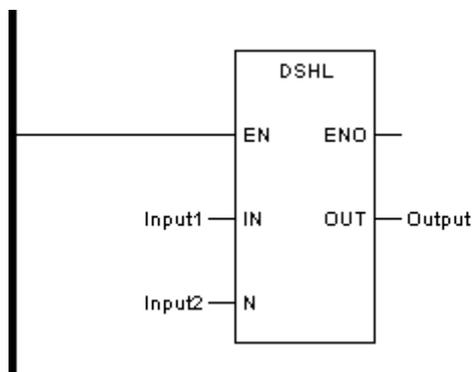
**5.4.10 DSHL: 32-bit shift left**

◆ **Function description**

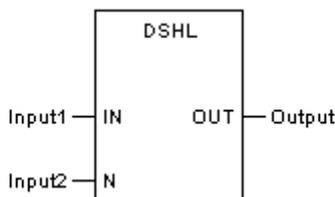
This function block shifts the bit pattern at the IN input to the left by N bits. Zeros are

filled in from the right. The result is assigned to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

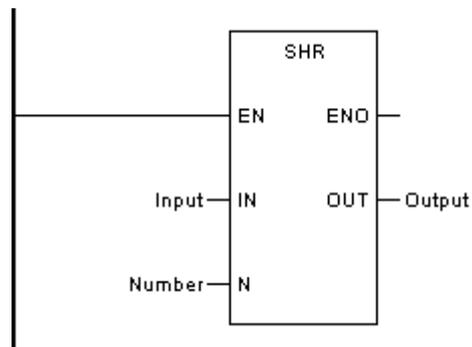
Icon	Parameter	Description	Data type	Point type
IN	Input	Bit pattern to be shifted	DINT	Constant, MW, NW
N	Number	Number to be shifted	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Bit pattern shifted	DINT	MW, NW

**5.4.11 SHR: 16-bit shift right**

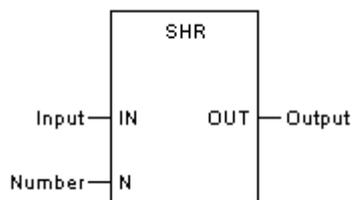
◆ **Function description**

This function block shifts the bit pattern at the IN input to the right by N bits. Zeros are filled in from the left. The result is assigned to the output.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input  
 SHR        Number  
 ST           Output

◆ **Representation in ST**

Output := SHR (Input, Number);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Bit pattern to be shifted	INT	Constant, IW, QW, MW, NW, SW
N	Number	Number to be shifted	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Bit pattern shifted	INT	MW, NW

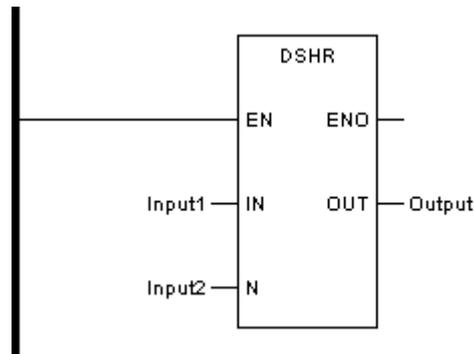
**5.4.12 DSHR: 32-bit shift right**

◆ **Function description**

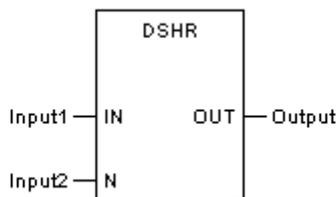
This function block shifts the bit pattern at the IN input to the right by N bits. Zeros are

filled in from the left. The result is assigned to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

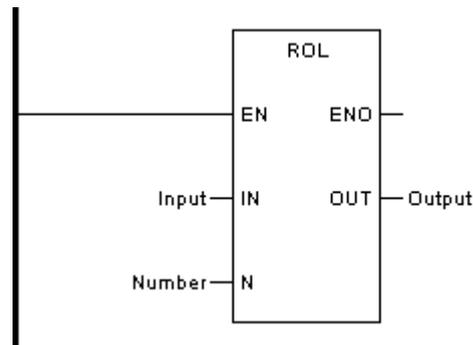
Icon	Parameter	Description	Data type	Point type
IN	Input	Bit pattern to be shifted	DINT	Constant, MW, NW
N	Number	Number to be shifted	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Bit pattern shifted	DINT	MW, NW

**5.4.13 ROL: 16-bit rotate left**

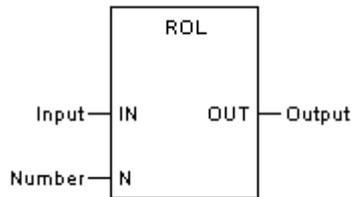
◆ **Function description**

This function block rotates the bit pattern at the IN input circularly to the left by N bits and assign the result to the output.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input  
 ROL        Number  
 ST            Output

◆ **Representation in ST**

Output := ROL (Input, Number);

◆ **Parameter description**

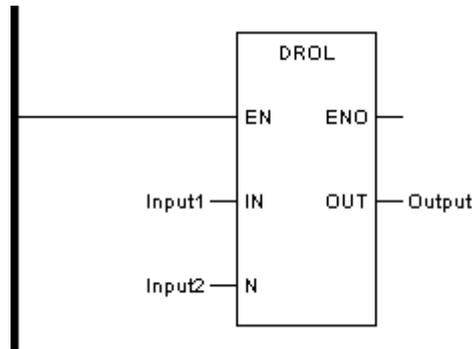
Icon	Parameter	Description	Data type	Point type
IN	Input	Bit pattern to be rotated	INT	Constant, IW, QW, MW, NW, SW
N	Number	Number to be rotated	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Bit pattern rotated	INT	MW, NW

**5.4.14 DROL: 32-bit rotate left**

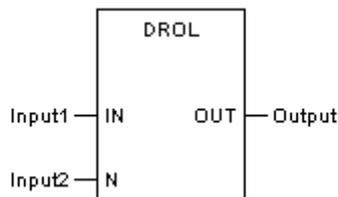
◆ **Function description**

This function block rotates the bit pattern at the IN input circularly to the left by N bits and assign the result to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

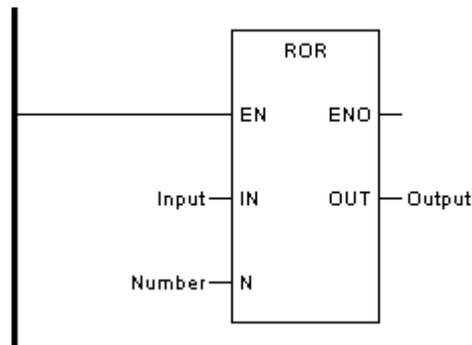
Icon	Parameter	Description	Data type	Point type
IN	Input	Bit pattern to be rotated	DINT	Constant, MW, NW
N	Number	Number to be rotated	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Bit pattern rotated	DINT	MW, NW

**5.4.15 ROR: 16-bit rotate right**

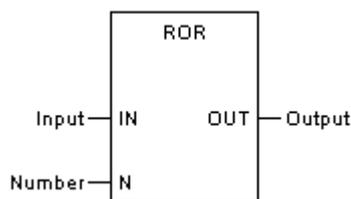
◆ **Function description**

This function block rotates the bit pattern at the IN input circularly to the right by N bits and assign the result to the output.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input  
 ROR        Number  
 ST            Output

◆ **Representation in ST**

Output := ROR (Input, Number);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Bit pattern to be rotated	INT	Constant, IW, QW, MW, NW, SW
N	Number	Number to be rotated	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Bit pattern rotated	INT	MW, NW

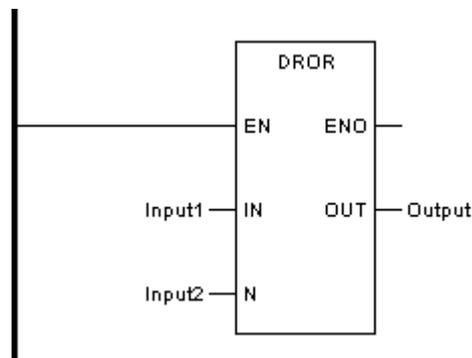
**5.4.16 DROR: 32-bit rotate right**

◆ **Function description**

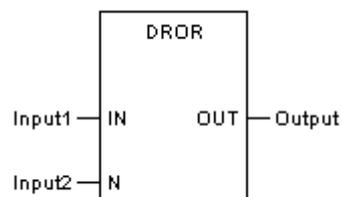
This function block rotates the bit pattern at the IN input circularly to the right by N bits

and assign the result to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

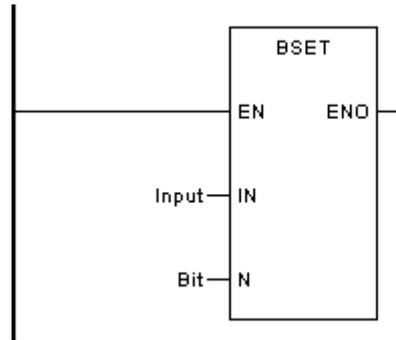
Icon	Parameter	Description	Data type	Point type
IN	Input	Bit pattern to be rotated	DINT	Constant, MW, NW
N	Number	Number to be rotated	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Bit pattern rotated	DINT	MW, NW

**5.4.17 BSET: Bit set**

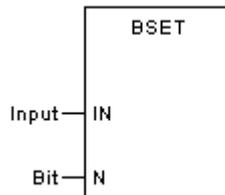
◆ **Function description**

This function block sets the Nth bit of the bit sequence at the IN input to “1”.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL BSET (IN:=Input, N:=Bit)

◆ **Representation in ST**

BSET (IN:=Input, N:=Bit);

◆ **Parameter description**

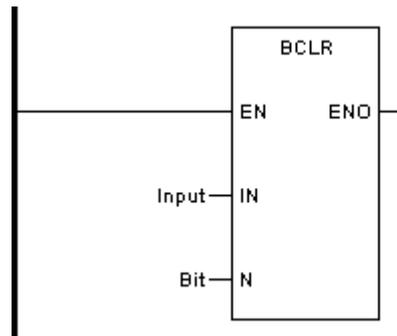
Icon	Parameter	Description	Data type	Point type
IN	Input	Input bit sequence	BOOL, INT	Q, M, MW, N, NW
N	Bit	Bit number	INT	Constant, IW, QW, MW, NW, SW

### 5.4.18 BCLR: Bit clear

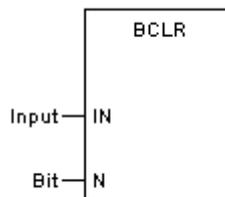
◆ **Function description**

This function block sets the Nth bit of the bit sequence at the IN input to “0”.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL BCLR (IN:=Input, N:=Bit)

◆ **Representation in ST**

BCLR (IN:=Input, N:=Bit);

◆ **Parameter description**

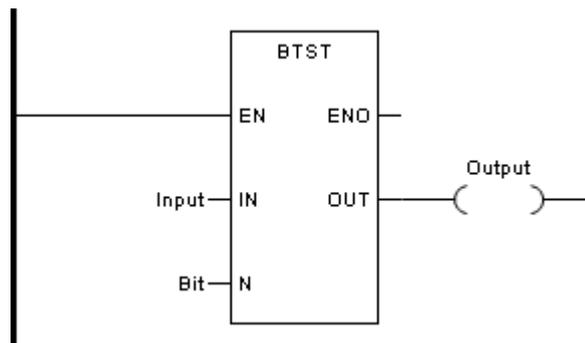
Icon	Parameter	Description	Data type	Point type
IN	Input	Input bit sequence	BOOL, INT	Q, M, MW, N, NW
N	Bit	Bit number	INT	Constant, IW, QW, MW, NW, SW

**5.4.19 BTST: Bit test**

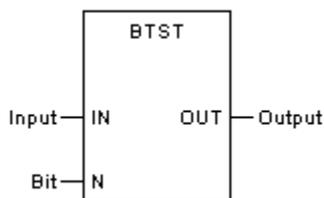
◆ **Function description**

This function block tests the Nth bit of the bit sequence at the IN input whether “1” or “0”. If the result is “1”, the output becomes “1”. If the result is “0”, the output becomes “0”.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL BTST (IN:=Input, N:=Bit, OUT=>Output)

◆ **Representation in ST**

BTST (IN:=Input, N:=Bit, OUT=>Output);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input bit sequence	BOOL, INT	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
N	Bit	Bit number	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output	BOOL	Q, M, N

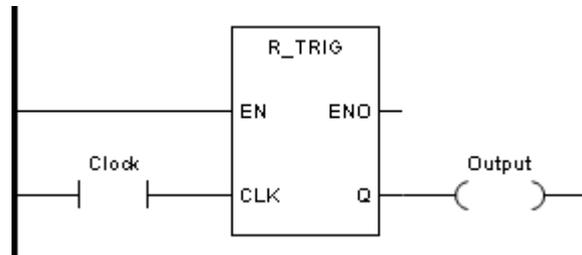
**5.4.20 R\_TRIG: Rising edge detection**

◆ **Function description**

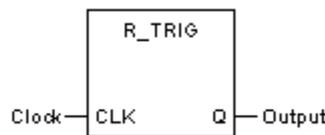
This function block is used for the detection of rising edges 0->1. Output Q becomes “1” if there is a transition from “0” to “1” at the CLK input. The output remains at “1” from one function block execution to the next (one cycle); the output subsequently

returns to “0”.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL R\_TRIG (CLK:=Clock, Q=>Output)

◆ **Representation in ST**

R\_TRIG (CLK:=Clock, Q=>Output);

◆ **Parameter description**

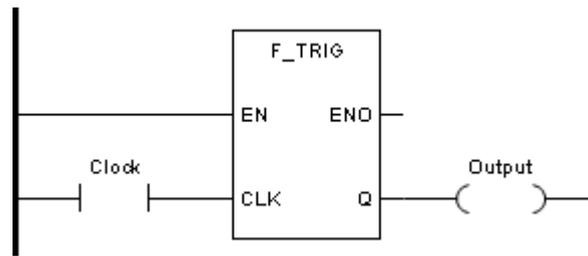
Icon	Parameter	Description	Data type	Point type
CLK	Clock	Clock input	BOOL	Constant, I, Q, M, N, S
Q	Output	Output	BOOL	Q, M, N

### 5.4.21 F\_TRIG: Falling edge detection

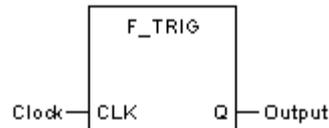
◆ **Function description**

This function block is used for the detection of falling edges 1->0. Output Q becomes “1” if there is a transition from “1” to “0” at the CLK input. The output remains at “1” from one function block execution to the next (one cycle); the output subsequently returns to “0”.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL F\_TRIG (CLK:=Clock, Q=>Output)

◆ **Representation in ST**

F\_TRIG (CLK:=Clock, Q=>Output);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
CLK	Clock	Clock input	BOOL	Constant, I, Q, M, N, S
Q	Output	Output	BOOL	Q, M, N

### 5.4.22 SET: Set

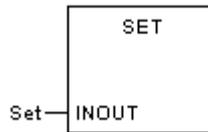
◆ **Function description**

This function block sets the bit associated with it to “1”. Its function is same as Set Coil -( S )-.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL SET (Set)

◆ **Representation in ST**

SET (Set);

◆ **Parameter description**

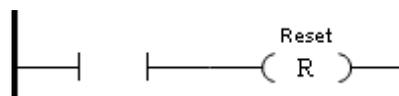
Icon	Parameter	Description	Data type	Point type
INOUT	Set	Bit to be set	BOOL	Q, M, N

### 5.4.23 RESET: Reset

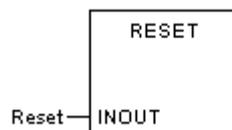
◆ **Function description**

This function block sets the bit associated with it to “0”. Its function is same as Reset Coil -( R )-.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL RESET (Reset )

◆ **Representation in ST**

RESET (Reset);

◆ **Parameter description**

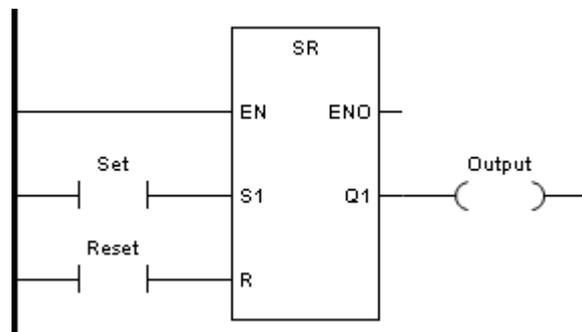
Icon	Parameter	Description	Data type	Point type
INOUT	Reset	Bit to be reset	BOOL	Q, M, N

**5.4.24 SR: Bistable (set dominant)**

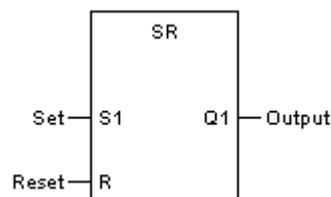
◆ **Function description**

This function block is used as SR memory with the property “Set dominant”. Output Q1 becomes “1” when the S1 input becomes “1”. This state remains even if input S1 reverts back to “0”. Output Q1 changes back to “0” when input R becomes “1”. If the inputs S1 and R are both “1” simultaneously, the dominating input S1 will set the output Q1 to “1”.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL SR (S1:=Set, R:=Reset, Q1=>Output)

◆ **Representation in ST**

SR (S1:=Set, R:=Reset, Q1=>Output);

◆ **Parameter description**

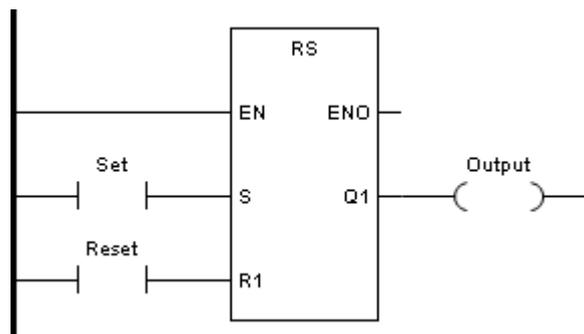
Icon	Parameter	Description	Data type	Point type
S1	Set	Set (dominant)	BOOL	Constant, I, Q, M, N, S
R	Reset	Reset	BOOL	Constant, I, Q, M, N, S
Q1	Output	Output	BOOL	Q, M, N

**5.4.25 RS: Bistable (reset dominant)**

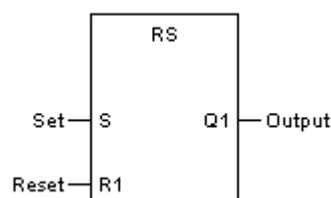
◆ **Function description**

This function block is used as RS memory with the property “Reset dominant”. Output Q1 becomes “1” when the S input becomes “1”. This state remains even if input S reverts back to “0”. Output Q1 changes back to “0” when input R1 becomes “1”. If the inputs S and R1 are both “1” simultaneously, the dominating input R1 will set the output Q1 to “0”.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL RS (S:=Set, R1:=Reset, Q1=>Output)

◆ **Representation in ST**

RS (S:=Set, R1:=Reset, Q1=>Output);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
S	Set	Set	BOOL	Constant, I, Q, M, N, S
R1	Reset	Reset (dominant)	BOOL	Constant, I, Q, M, N, S
Q1	Output	Output	BOOL	Q, M, N

## 5.5 Comparison

This chapter describes the elementary function blocks of the Comparison family.

This chapter contains the following sections.

Type	Description	Formula
<b>EQ</b>	16-Bit Equal to	OUT = (IN1 = IN2)
<b>DEQ</b>	32-Bit Equal to	
<b>EEQ</b>	Floating-Point Equal to	
<b>NE</b>	16-Bit Not Equal to	OUT = (IN1 <> IN2)
<b>DNE</b>	32-Bit Not Equal to	
<b>ENE</b>	Floating-Point Not Equal to	
<b>GT</b>	16-Bit Greater than	OUT = (IN1 > IN2)
<b>DGT</b>	32-Bit Greater than	
<b>EGT</b>	Floating-Point Greater than	
<b>GE</b>	16-Bit Greater than or Equal to	OUT = (IN1 ≥ IN2)
<b>DGE</b>	32-Bit Greater than or Equal to	
<b>EGE</b>	Floating-Point Greater than or Equal to	

<b>LT</b>	16-Bit Less than	OUT = (IN1 < IN2)
<b>DLT</b>	32-Bit Less than	
<b>ELT</b>	Floating-Point Less than	
<b>LE</b>	16-Bit Less than or Equal to	OUT = (IN1 ≤ IN2)
<b>DLE</b>	32-Bit Less than or Equal to	
<b>ELE</b>	Floating-Point Less than or Equal to	

### 5.5.1 EQ: 16-bit equal to

#### ◆ Function description

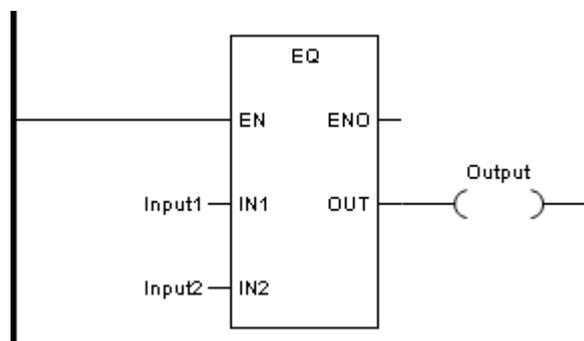
This function block checks the inputs for equality, i.e. the output becomes “1” if there is equality at all inputs; otherwise, the output remains at “0”.

The number of inputs can be increased to a maximum of 8.

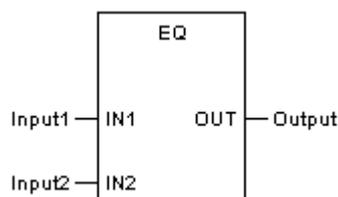
#### ◆ Formula

OUT = 1, if (IN1 = IN2) AND (IN2 = IN3) AND ... AND (INn-1 = INn)

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

LD                      Input1

EQ	Input2
ST	Output

◆ **Representation in ST**

Output := EQ (Input1, Input2);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Input 2	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output	BOOL	Q, M, N

**5.5.2 DEQ: 32-bit equal to**

◆ **Function description**

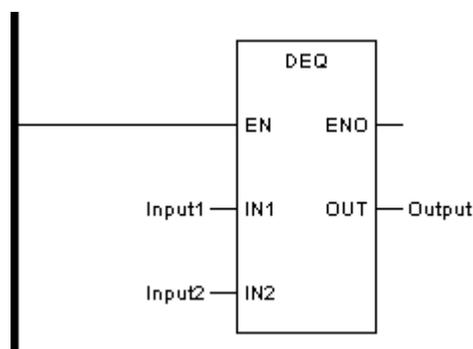
This function block checks the inputs for equality, i.e. the output becomes “1” if there is equality at all inputs; otherwise, the output remains at “0”. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

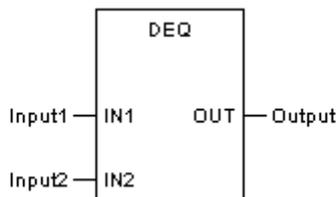
◆ **Formula**

$$OUT = 1, \text{ if } (IN1 = IN2) \text{ AND } (IN2 = IN3) \text{ AND } \dots \text{ AND } (IN_{n-1} = IN_n)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	DINT	Constant, MW, NW
IN2	Input2	Input 2	DINT	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

### 5.5.3 EEQ: Floating-point equal to

◆ **Function description**

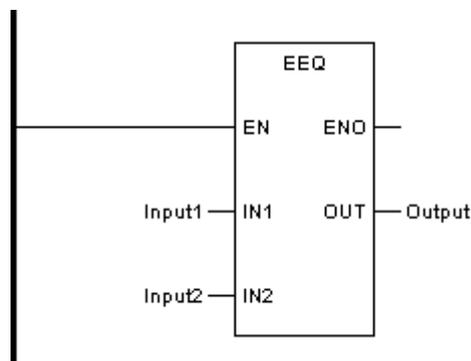
This function block checks the inputs for equality, i.e. the output becomes “1” if there is equality at all inputs; otherwise, the output remains at “0”. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

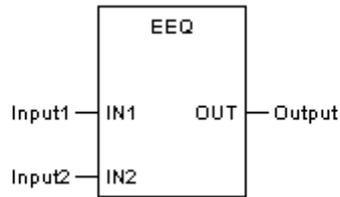
◆ **Formula**

$$OUT = 1, \text{ if } (IN1 = IN2) \text{ AND } (IN2 = IN3) \text{ AND } \dots \text{ AND } (IN_{n-1} = IN_n)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	REAL	Constant, MW, NW
IN2	Input2	Input 2	REAL	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

### 5.5.4 NE: 16-bit not equal to

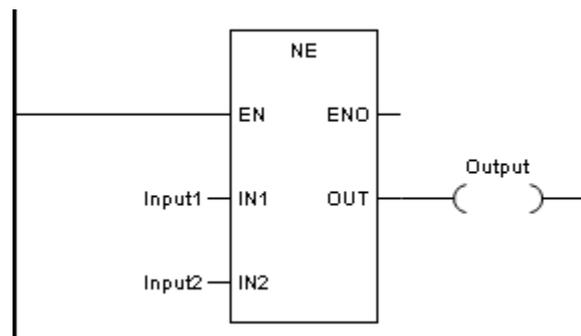
◆ **Function description**

This function block checks the input values for inequality.

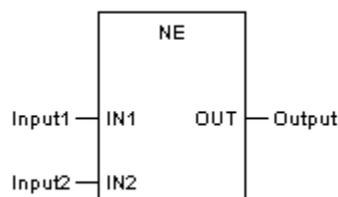
◆ **Formula**

OUT = 1, if IN1 <> IN2

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

```
LD      Input1
NE      Input2
ST      Output
```

◆ **Representation in ST**

```
Output := NE (Input1, Input2);
```

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Input 2	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output	BOOL	Q, M, N

### 5.5.5 DNE: 32-bit not equal to

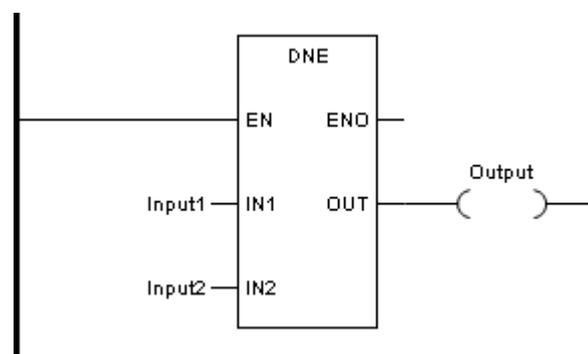
◆ **Function description**

This function block checks the input values for inequality. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

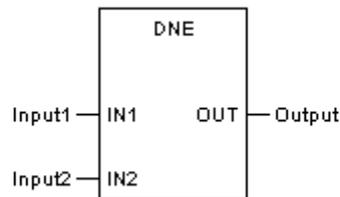
◆ **Formula**

OUT = 1, if IN1<>IN2

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	DINT	Constant, MW, NW
IN2	Input2	Input 2	DINT	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

**5.5.6 ENE: Floating-point not equal to**

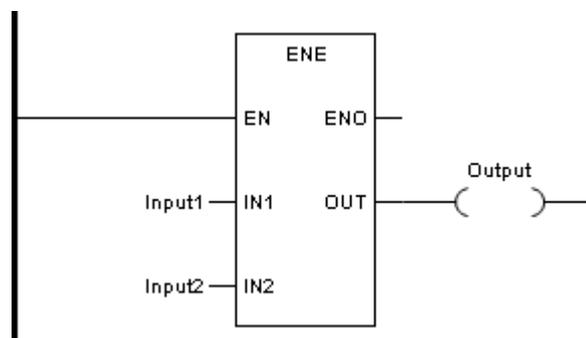
◆ **Function description**

This function block checks the input values for inequality. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

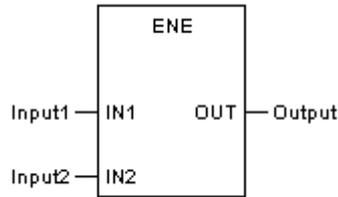
◆ **Formula**

$$OUT = 1, \text{ if } IN1 \neq IN2$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	REAL	Constant, MW, NW
IN2	Input2	Input 2	REAL	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

### 5.5.7 GT: 16-bit greater than

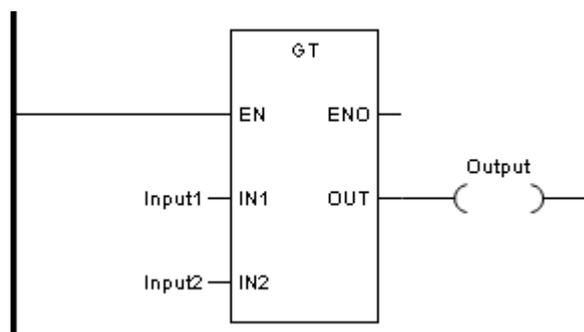
◆ **Function description**

This function block checks the values of successive inputs for a decreasing sequence. The number of inputs can be increased to a maximum of 8.

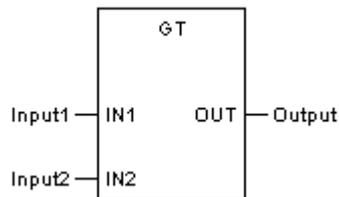
◆ **Formula**

$$OUT = 1, \text{ if } (IN1 > IN2) \text{ AND } (IN2 > IN3) \text{ AND } \dots \text{ AND } (IN_{n-1} > IN_n)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input1  
 GT            Input2  
 ST            Output

◆ **Representation in ST**

Output := GT (Input1, Input2);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Input 2	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output	BOOL	Q, M, N

**5.5.8 DGT: 32-bit greater than**

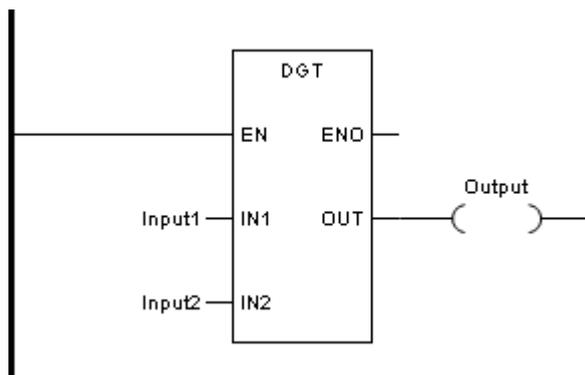
◆ **Function description**

This function block checks the values of successive inputs for a decreasing sequence. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd. The number of inputs can be increased to a maximum of 8.

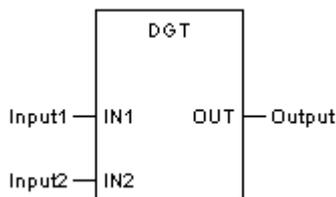
◆ **Formula**

OUT = 1, if (IN1>IN2) AND (IN2>IN3) AND ... AND (INn-1>INn)

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	DINT	Constant, MW, NW
IN2	Input2	Input 2	DINT	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

**5.5.9 EGT: Floating-point greater than**

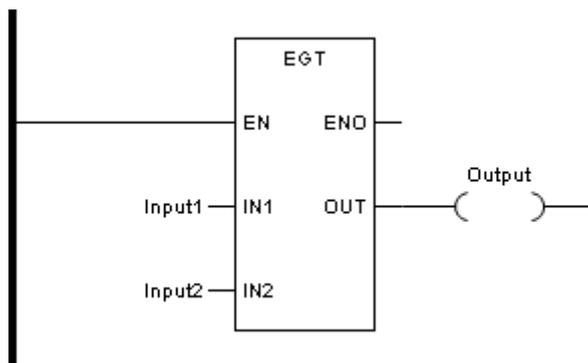
◆ **Function description**

This function block checks the values of successive inputs for a decreasing sequence. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd. The number of inputs can be increased to a maximum of 8.

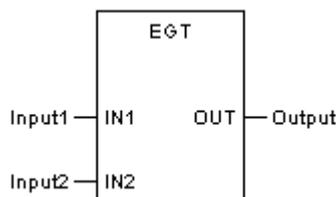
◆ **Formula**

$$OUT = 1, \text{ if } (IN1 > IN2) \text{ AND } (IN2 > IN3) \text{ AND } \dots \text{ AND } (IN_{n-1} > IN_n)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	REAL	Constant, MW, NW
IN2	Input2	Input 2	REAL	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

**5.5.10 GE: 16-bit greater than or equal to**

◆ **Function description**

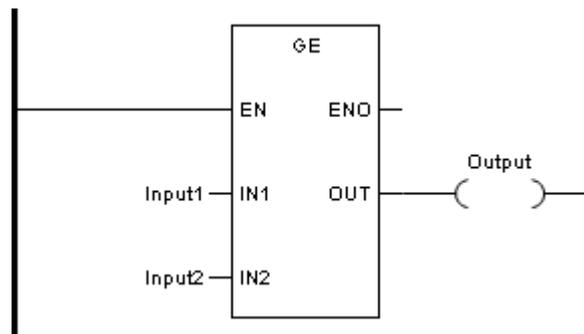
This function block checks the values of successive inputs for a decreasing sequence or equality.

The number of inputs can be increased to a maximum of 8.

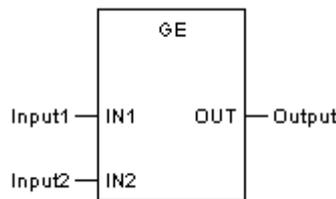
◆ **Formula**

$$OUT = 1, \text{ if } (IN1 \geq IN2) \text{ AND } (IN2 \geq IN3) \text{ AND } \dots \text{ AND } (IN_{n-1} \geq IN_n)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

```

LD      Input1
GE      Input2
ST      Output
    
```

◆ **Representation in ST**

```

Output := GE (Input1, Input2);
    
```

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Input 2	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output	BOOL	Q, M, N

### 5.5.11 DGE: 32-bit greater than or equal to

◆ **Function description**

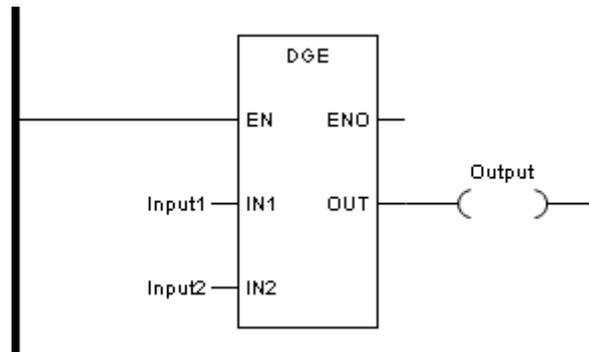
This function block checks the values of successive inputs for a decreasing sequence or equality. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

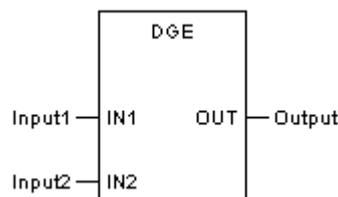
◆ **Formula**

OUT = 1, if (IN1 ≥ IN2) AND (IN2 ≥ IN3) AND ... AND (INn-1 ≥ INn)

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	DINT	Constant, MW, NW
IN2	Input2	Input 2	DINT	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

### 5.5.12 EGE: Floating-point greater than or equal to

◆ **Function description**

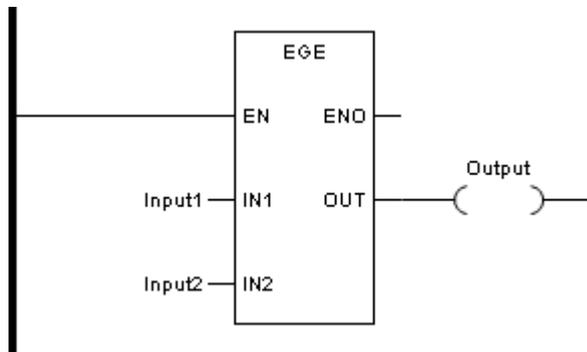
This function block checks the values of successive inputs for a decreasing sequence or equality. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

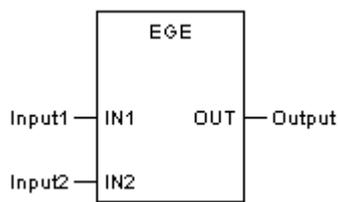
◆ **Formula**

OUT = 1, if (IN1 ≥ IN2) AND (IN2 ≥ IN3) AND ... AND (INn-1 ≥ INn)

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	REAL	Constant, MW, NW
IN2	Input2	Input 2	REAL	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

**5.5.13 LT: 16-bit less than**

◆ **Function description**

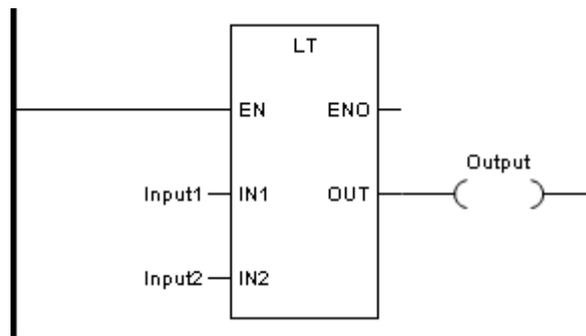
This function block checks the values of successive inputs for an increasing sequence.

The number of inputs can be increased to a maximum of 8.

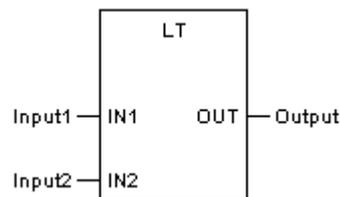
◆ **Formula**

OUT = 1, if (IN1 < IN2) AND (IN2 < IN3) AND ... AND (INn-1 < INn)

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input1  
 LT            Input2  
 ST            Output

◆ **Representation in ST**

Output := LT (Input1, Input2);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Input 2	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output	BOOL	Q, M, N

**5.5.14 DLT: 32-bit less than**

◆ **Function description**

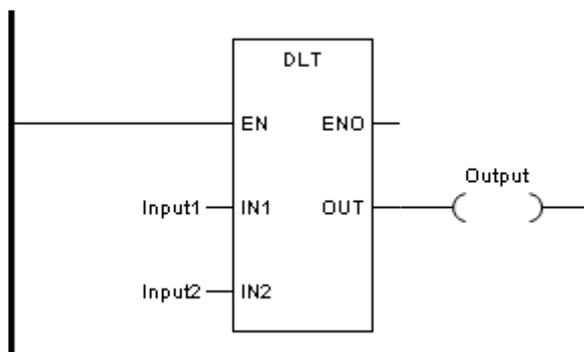
This function block checks the values of successive inputs for an increasing sequence.

Input and output can only be 32-bit register, occupying 2 continuous word registers.  
 The serial number of word register must be odd.  
 The number of inputs can be increased to a maximum of 8.

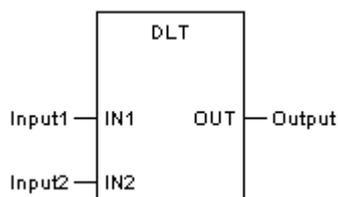
◆ **Formula**

$$OUT = 1, \text{ if } (IN1 < IN2) \text{ AND } (IN2 < IN3) \text{ AND } \dots \text{ AND } (IN_{n-1} < IN_n)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	DINT	Constant, MW, NW
IN2	Input2	Input 2	DINT	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

### 5.5.15 ELT: Floating-point less than

◆ **Function description**

This function block checks the values of successive inputs for an increasing sequence.

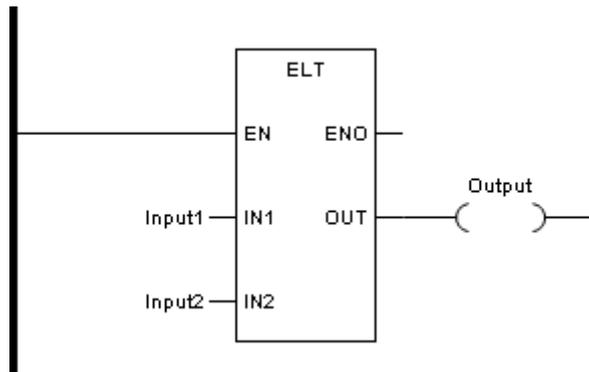
Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

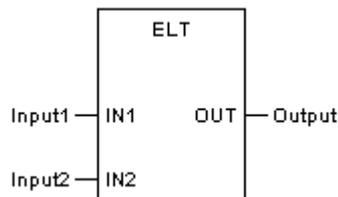
◆ **Formula**

$$OUT = 1, \text{ if } (IN1 < IN2) \text{ AND } (IN2 < IN3) \text{ AND } \dots \text{ AND } (IN_{n-1} < IN_n)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	REAL	Constant, MW, NW
IN2	Input2	Input 2	REAL	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

### 5.5.16 LE: 16-bit less than or equal to

◆ **Function description**

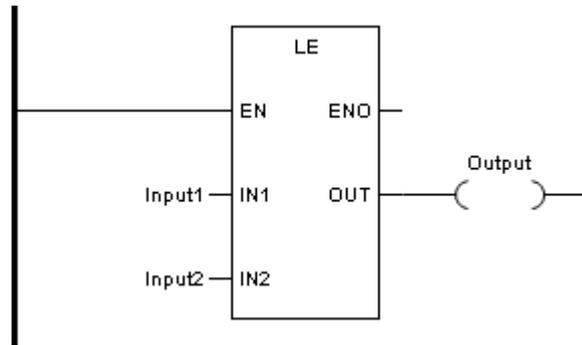
This function block checks the values of successive inputs for an increasing sequence or equality.

The number of inputs can be increased to a maximum of 8.

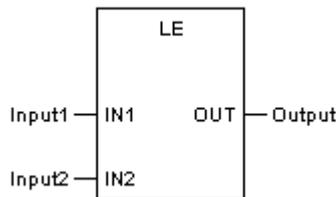
◆ **Formula**

OUT = 1, if (IN1≤IN2) AND (IN2≤IN3) AND ... AND (INn-1≤INn)

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input1  
LE            Input2  
ST            Output

◆ **Representation in ST**

Output := LE (Input1, Input2);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	INT	Constant, IW, QW, MW, NW, SW
IN2	Input2	Input 2	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output	BOOL	Q, M, N

## 5.5.17 DLE: 32-bit less than or equal to

### ◆ Function description

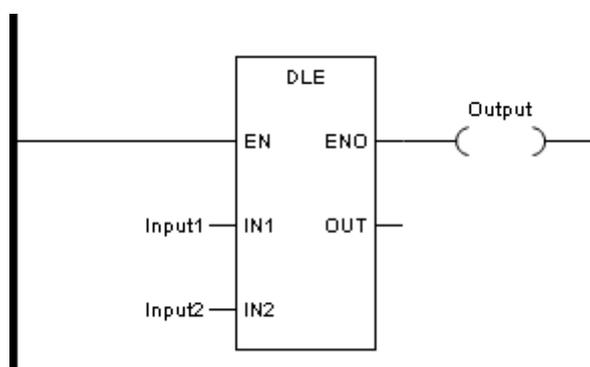
This function block checks the values of successive inputs for an increasing sequence or equality. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

The number of inputs can be increased to a maximum of 8.

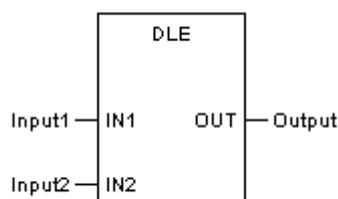
### ◆ Formula

$OUT = 1$ , if  $(IN1 \leq IN2)$  AND  $(IN2 \leq IN3)$  AND ... AND  $(IN_{n-1} \leq IN_n)$

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	DINT	Constant, MW, NW
IN2	Input2	Input 2	DINT	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

### 5.5.18 ELE: Floating-point less than or equal to

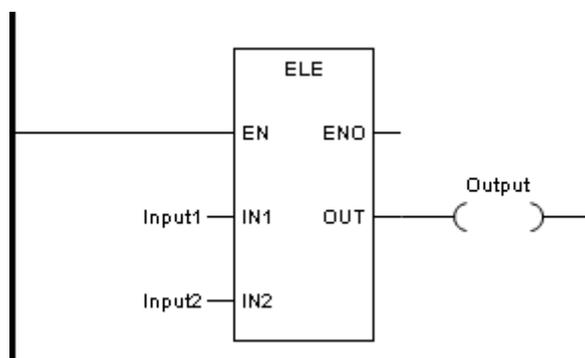
◆ **Function description**

This function block checks the values of successive inputs for an increasing sequence or equality. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd. The number of inputs can be increased to a maximum of 8.

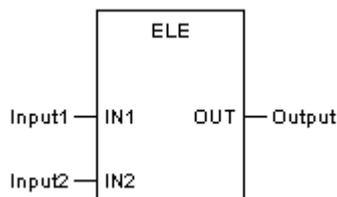
◆ **Formula**

$$OUT = 1, \text{ if } (IN1 \leq IN2) \text{ AND } (IN2 \leq IN3) \text{ AND } \dots \text{ AND } (IN_{n-1} \leq IN_n)$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Input 1	REAL	Constant, MW, NW
IN2	Input2	Input 2	REAL	Constant, MW, NW
OUT	Output	Output	BOOL	Q, M, N

## 5.6 Conversion

This chapter describes the elementary function blocks of the Conversion family.

This chapter contains the following sections.

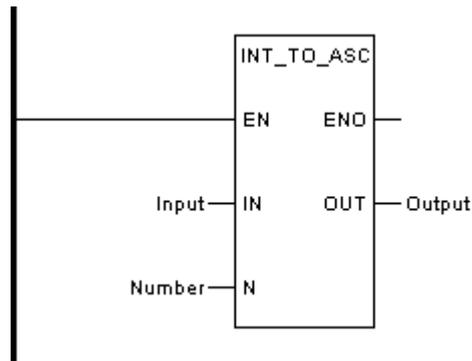
Type	Description
<b>INT_TO_ASC</b>	Integer to ASCII Code
<b>ASC_TO_INT</b>	ASCII Code to Integer
<b>INT_TO_BCD</b>	Integer to BCD Code
<b>BCD_TO_INT</b>	BCD Code to Integer
<b>INT_TO_GRY</b>	16-Bit Integer to Gray Code
<b>GRY_TO_INT</b>	Gray Code to 16-Bit Integer
<b>DIT_TO_GRY</b>	32-Bit Integer to Gray Code
<b>GRY_TO_DIT</b>	Gray Code to 32-Bit Integer
<b>INT_TO_DIT</b>	16-Bit Integer to 32-Bit Integer
<b>INT_TO_RAL</b>	16-Bit Integer to Floating-Point
<b>RAL_TO_INT</b>	Floating-Point to 16-Bit Integer
<b>DIT_TO_RAL</b>	32-Bit Integer to Floating-Point
<b>RAL_TO_DIT</b>	Floating-Point to 32-Bit Integer

### 5.6.1 INT\_TO\_ASC: Integer to ASCII code

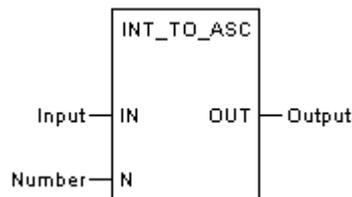
#### ◆ Function description

This function block converts the input integer into ASCII code.

#### ◆ Representation in LD



◆ **Representation in FBD**



◆ **Representation in IL**

CAL INT\_TO\_ASC (IN:=Input, N:=Number, OUT=>Output)

◆ **Representation in ST**

INT\_TO\_ASC (IN:=Input, N:=Number, OUT=>Output);

◆ **Parameter description**

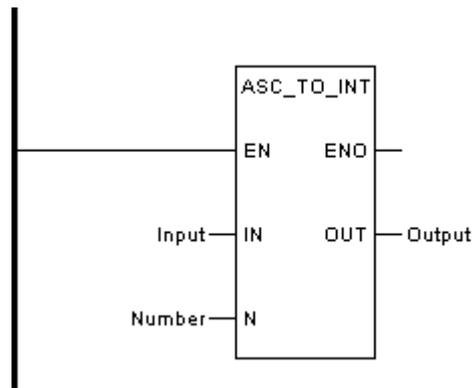
Icon	Parameter	Description	Data type	Point type
IN	Input	Integer	INT	MW, NW
N	Number	Number of integers	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	ASCII code	INT	MW, NW

## 5.6.2 ASC\_TO\_INT: ASCII code to integer

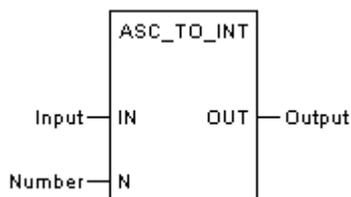
◆ **Function description**

This function block converts the ASCII code into integer.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL ASC\_TO\_INT (IN:=Input, N:=Number, OUT=>Output)

◆ **Representation in ST**

ASC\_TO\_INT (IN:=Input, N:=Number, OUT=>Output);

◆ **Parameter description**

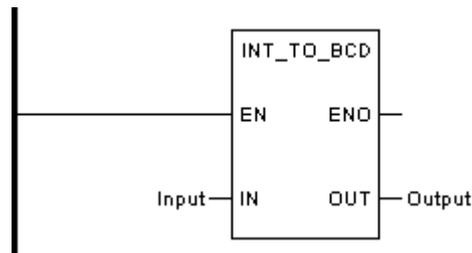
Icon	Parameter	Description	Data type	Point type
IN	Input	ASCII code	INT	MW, NW
N	Number	Number of ASCII codes	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Integer	INT	MW, NW

**5.6.3 INT\_TO\_BCD: Integer to BCD code**

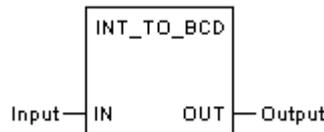
◆ **Function description**

This function block converts a binary coded integer into a Binary Coded Decimal (BCD) integer.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

```

LD      Input
INT_TO_BCD
ST      Output
    
```

◆ **Representation in ST**

```

Output := INT_TO_BCD (Input);
    
```

◆ **Parameter description**

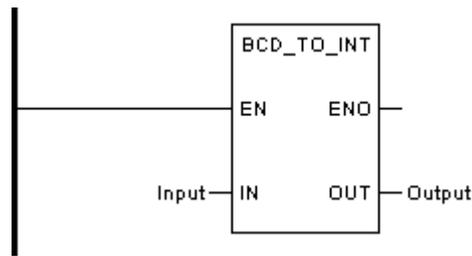
Icon	Parameter	Description	Data type	Point type
IN	Input	Binary coded integer	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	BCD integer	INT	MW, NW

### 5.6.4 BCD\_TO\_INT: BCD code to integer

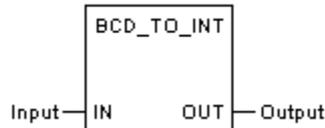
◆ **Function description**

This function block converts a Binary Coded Decimal (BCD) integer into a binary coded integer.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

```
LD          Input
BCD_TO_INT
ST          Output
```

◆ **Representation in ST**

```
Output := BCD_TO_INT (Input);
```

◆ **Parameter description**

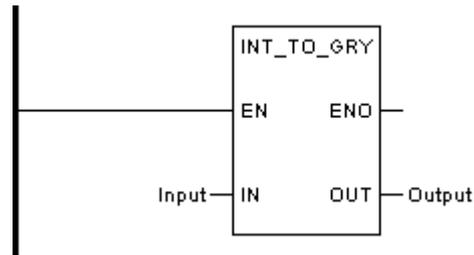
Icon	Parameter	Description	Data type	Point type
IN	Input	BCD integer	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Binary coded integer	INT	MW, NW

### 5.6.5 INT\_TO\_GRY: 16-bit integer to Gray code

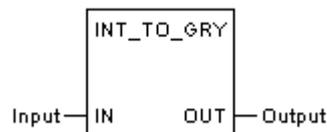
◆ **Function description**

This function block converts a 16-bit binary coded integer into a Gray code integer.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

```
LD      Input
INT_TO_GRY
ST      Output
```

◆ **Representation in ST**

```
Output := INT_TO_GRY (Input);
```

◆ **Parameter description**

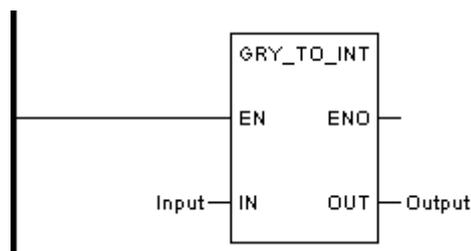
Icon	Parameter	Description	Data type	Point type
IN	Input	Binary coded integer	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Gray code integer	INT	MW, NW

### 5.6.6 GRY\_TO\_INT: Gray code to 16-bit integer

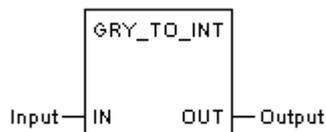
◆ **Function description**

This function block converts a Gray code integer into a 16-bit binary coded integer.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

```
LD      Input
GRY_TO_INT
ST      Output
```

◆ **Representation in ST**

```
Output := GRY_TO_INT (Input);
```

◆ **Parameter description**

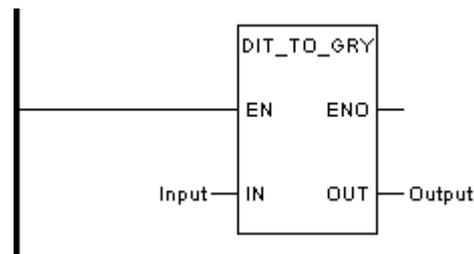
Icon	Parameter	Description	Data type	Point type
IN	Input	Gray code integer	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Binary coded integer	INT	MW, NW

**5.6.7 DIT\_TO\_GRY: 32-bit integer to Gray code**

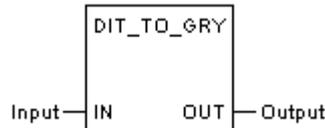
◆ **Function description**

This function block converts a 32-bit binary coded integer into a Gray code integer. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

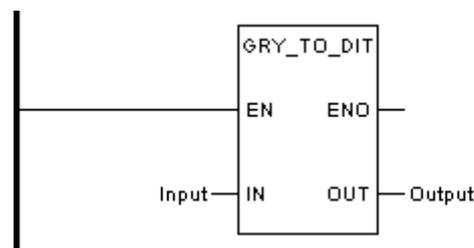
Icon	Parameter	Description	Data type	Point type
IN	Input	Binary coded integer	DINT	Constant, MW, NW
OUT	Output	Gray code integer	DINT	MW, NW

### 5.6.8 GRY\_TO\_DIT: Gray code to 32-bit integer

◆ **Function description**

This function block converts a Gray code integer into a 32-bit binary coded integer. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

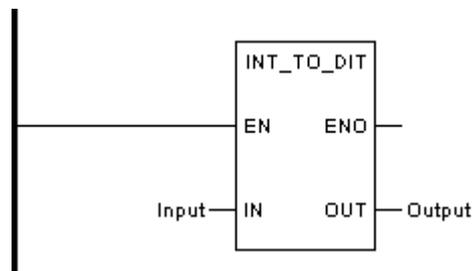
Icon	Parameter	Description	Data type	Point type
IN	Input	Gray code integer	DINT	Constant, MW, NW
OUT	Output	Binary coded integer	DINT	MW, NW

### 5.6.9 INT\_TO\_DIT: 16-bit integer to 32-bit integer

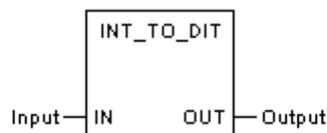
◆ **Function description**

This function block converts a 16-bit integer into a 32-bit integer. Output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	16-bit integer input	INT	MW, NW

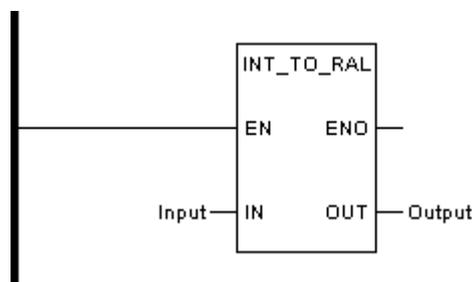
OUT	Output	32-bit integer output	DINT	MW, NW
-----	--------	-----------------------	------	--------

### 5.6.10 INT\_TO\_RAL: 16-bit integer to floating-point

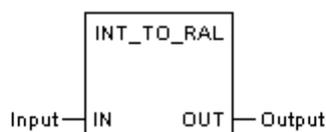
◆ **Function description**

This function block converts a 16-bit integer into a floating-point. Output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

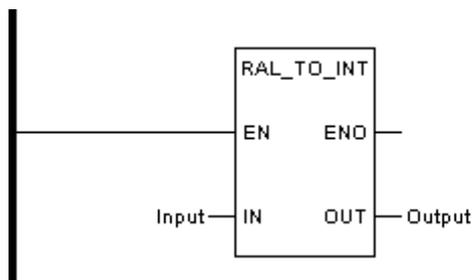
Icon	Parameter	Description	Data type	Point type
IN	Input	16-bit integer input	INT	MW, NW
OUT	Output	Floating-point output	REAL	MW, NW

### 5.6.11 RAL\_TO\_INT: Floating-point to 16-bit integer

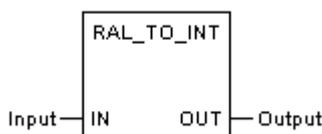
◆ **Function description**

This function block converts a floating-point into a 16-bit integer. Input can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

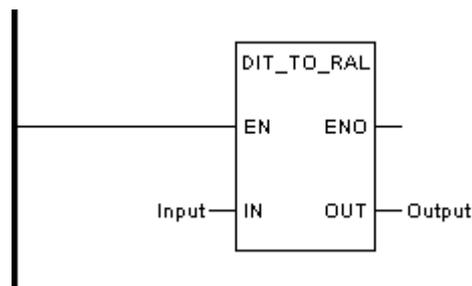
Icon	Parameter	Description	Data type	Point type
IN	Input	Floating-point input	REAL	MW, NW
OUT	Output	16-bit integer output	INT	MW, NW

**5.6.12 DIT\_TO\_RAL: 32-bit integer to floating-point**

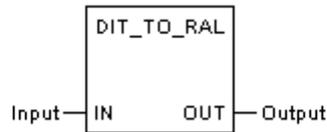
◆ **Function description**

This function block converts a 32-bit integer into a floating-point. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

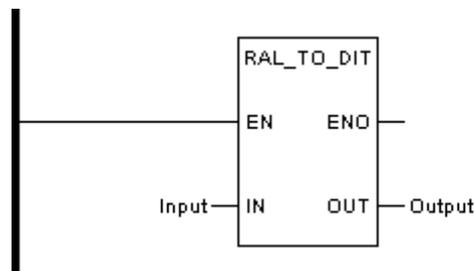
Icon	Parameter	Description	Data type	Point type
IN	Input	32-bit integer input	DINT	MW, NW
OUT	Output	Floating-point output	REAL	MW, NW

### 5.6.13 RAL\_TO\_DIT: Floating-point to 32-bit integer

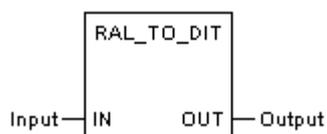
◆ **Function description**

This function block converts a floating-point into a 32-bit integer. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Floating-point input	REAL	MW, NW

OUT	Output	32-bit integer output	DINT	MW, NW
-----	--------	-----------------------	------	--------

## 5.7 Data move

This chapter describes the elementary function blocks of the Data Move family.

This chapter contains the following sections.

Type	Description
<b>MOVE</b>	Assignment
<b>DMOVE</b>	32-Bit Assignment
<b>EMOVE</b>	Floating-Point Assignment
<b>BLKMOV</b>	Block Data Move
<b>BLKCLR</b>	Block Data Clear
<b>SWAP</b>	High/Low Byte Swap
<b>SFL</b>	Bit Shift Left
<b>SFR</b>	Bit Shift Right
<b>BSFL</b>	Byte Shift Left
<b>BSFR</b>	Byte Shift Right
<b>WSFL</b>	Word Shift Left
<b>WSFR</b>	Word Shift Right
<b>XCH</b>	16-Bit Data Exchange
<b>DXCH</b>	32-Bit Data Exchange
<b>EXCH</b>	Floating-Point Data Exchange

### 5.7.1 MOVE: Assignment

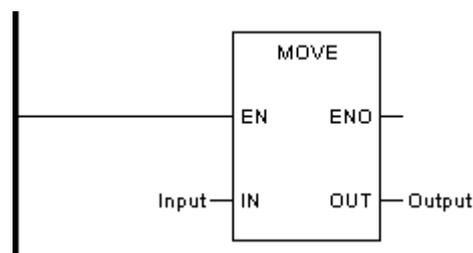
#### ◆ Function description

This function block assigns the input value to the output.

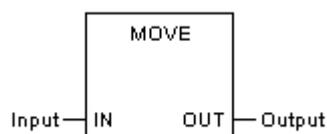
#### ◆ Formula

OUT = IN

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

LD            Input  
ST            Output

◆ **Representation in ST**

Output := Input;

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input value	BOOL, INT	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
OUT	Output	Output value	BOOL, INT	Q, M, MW, N, NW

## 5.7.2 DMOVE: 32-bit assignment

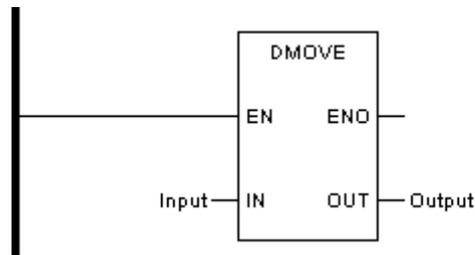
◆ **Function description**

This function block assigns the input value to the output. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

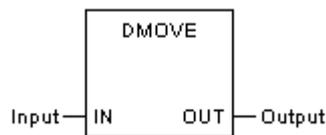
◆ **Formula**

OUT = IN

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input value	DINT	Constant, MW, NW
OUT	Output	Output value	DINT	MW, NW

### 5.7.3 EMOVE: Floating-point assignment

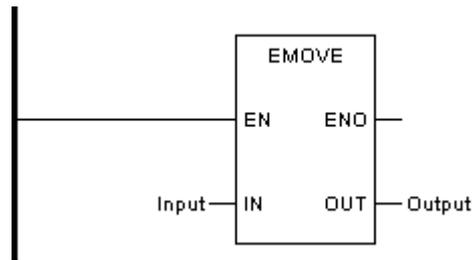
◆ **Function description**

This function block assigns the input value to the output. Input and output can only be floating-point, occupying 2 continuous word registers. The serial number of word register must be odd.

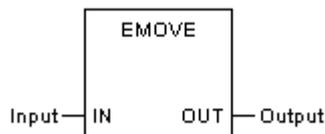
◆ **Formula**

OUT = IN

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

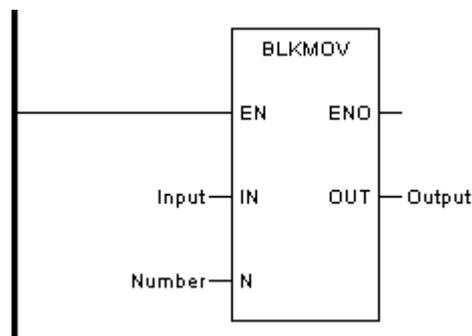
Icon	Parameter	Description	Data type	Point type
IN	Input	Input value	REAL	Constant, MW, NW
OUT	Output	Output value	REAL	MW, NW

**5.7.4 BLKMOV: Block data move**

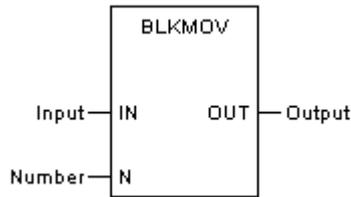
◆ **Function description**

This function block copies a block of input data to output data.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL BLKMOV (IN:=Input, N:=Number, OUT=>Output)

◆ **Representation in ST**

BLKMOV (IN:=Input, N:=Number, OUT=>Output);

◆ **Parameter description**

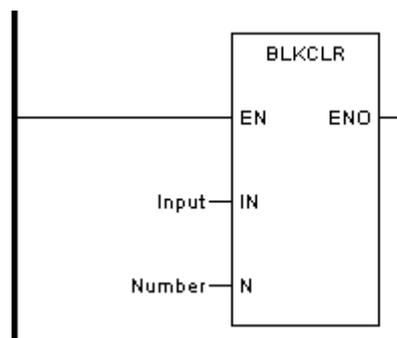
Icon	Parameter	Description	Data type	Point type
IN	Input	Input data	BOOL, INT	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
N	Number	Number of input data to be copied	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output data	BOOL, INT	M, MW, N, NW, Q

### 5.7.5 BLKCLR: Block data clear

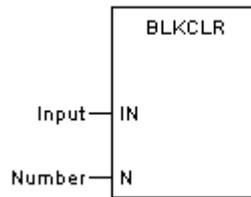
◆ **Function description**

This function block fills a specified block of input data with zeros.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL BLKCLR (IN:=Input, N:=Number)

◆ **Representation in ST**

BLKCLR (IN:=Input, N:=Number);

◆ **Parameter description**

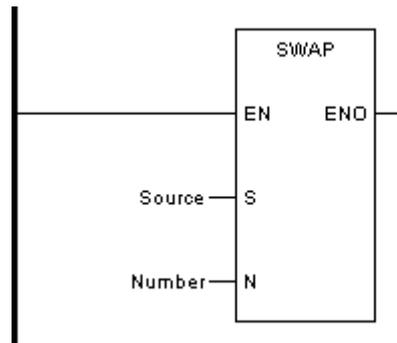
Icon	Parameter	Description	Data type	Point type
IN	Input	Input data	BOOL, INT	Q, M, MW, N, NW
N	Number	Number of input data to be cleared	INT	Constant, IW, QW, MW, NW, SW

### 5.7.6 SWAP: High/low byte swap

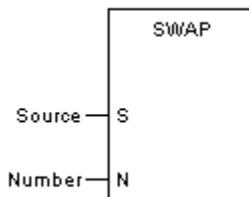
◆ **Function description**

This function block swaps the higher byte and the lower byte of every word register designated at S.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL SWAP (S:=Source, N:=Number)

◆ **Representation in ST**

SWAP (S:=Source, N:=Number);

◆ **Parameter description**

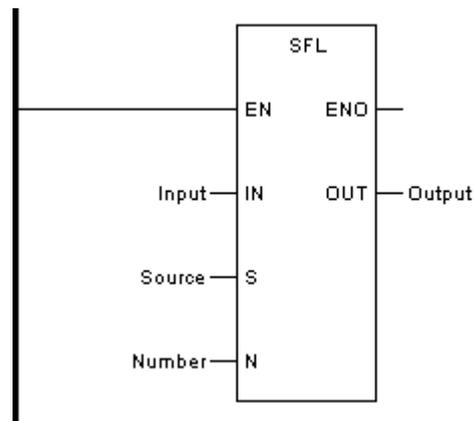
Icon	Parameter	Description	Data type	Point type
S	Source	Word register to be swapped	INT	MW, NW
N	Number	Number of word registers	INT	Constant, IW, QW, MW, NW, SW

**5.7.7 SFL: Bit shift left**

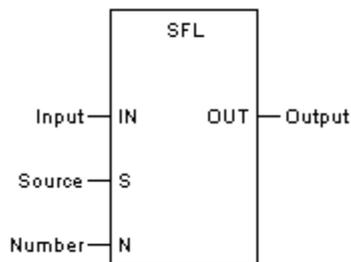
◆ **Function description**

This function block causes a shift to the left by 1 bit of N continuous data designated at S. The bit to be shifted in is designated at IN, the bit shifted out is at OUT.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL SFL (IN:=Input, S:= Source, N:=Number, OUT=>Output)

◆ **Representation in ST**

SFL (IN:=Input, S:= Source, N:=Number, OUT=>Output);

◆ **Parameter description**

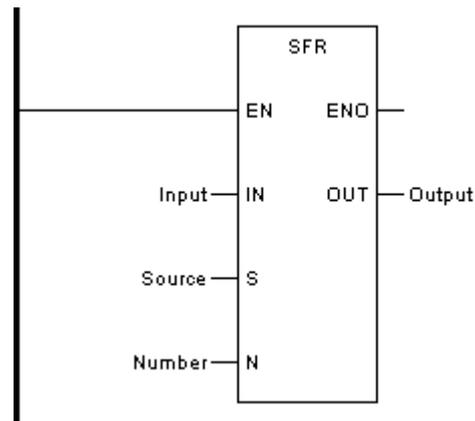
Icon	Parameter	Description	Data type	Point type
IN	Input	Bit to be shifted in	BOOL	Constant, I, Q, M, N, S
S	Source	Data to be shifted	BOOL, INT	M, N, MW, NW
N	Number	Number of data	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Bit shifted out	BOOL	Q, M, N

## 5.7.8 SFR: Bit shift right

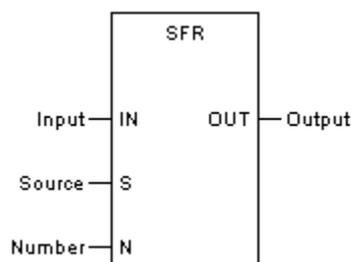
### ◆ Function description

This function block causes a shift to the right by 1 bit of N continuous data designated at S. The bit to be shifted in is designated at IN, the bit shifted out is at OUT.

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Representation in IL

```
CAL SFR (IN:=Input, S:= Source, N:=Number, OUT=>Output)
```

### ◆ Representation in ST

```
SFR(IN:=Input, S:= Source, N:=Number, OUT=>Output);
```

### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
IN	Input	Bit to be shifted in	BOOL	Constant, I, Q, M, N, S

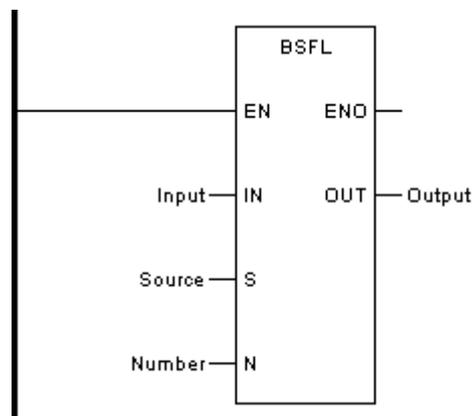
S	Source	Data to be shifted	BOOL, INT	M, N, MW, NW
N	Number	Number of data	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Bit shifted out	BOOL	Q, M, N

### 5.7.9 BSFL: Byte shift left

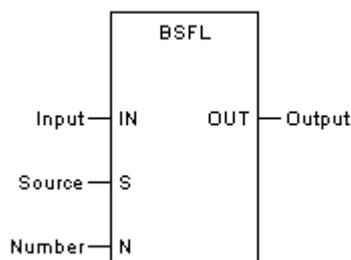
#### ◆ Function description

This function block causes a shift to the left by 1 byte of N continuous data designated at S. The byte to be shifted in is designated at IN, the byte shifted out is at OUT.

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

CAL BSFL (IN:=Input, S:= Source, N:=Number, OUT=>Output)

#### ◆ Representation in ST

BSFL (IN:=Input, S:= Source, N:=Number, OUT=>Output);

◆ **Parameter description**

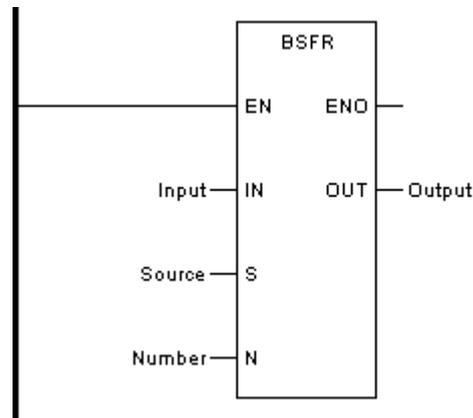
Icon	Parameter	Description	Data type	Point type
IN	Input	Byte to be shifted in	INT	Constant, IW, QW, MW, NW, SW
S	Source	Data to be shifted	INT	MW, NW
N	Number	Number of data	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Byte shifted out	INT	MW, NW

**5.7.10 BSFR: Byte shift right**

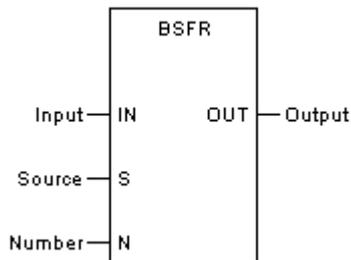
◆ **Function description**

This function block causes a shift to the right by 1 byte of N continuous data designated at S. The byte to be shifted in is designated at IN, the byte shifted out is at OUT.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL BSFR (IN:=Input, S:= Source, N:=Number, OUT=>Output)

◆ **Representation in ST**

BSFR (IN:=Input, S:= Source, N:=Number, OUT=>Output);

◆ **Parameter description**

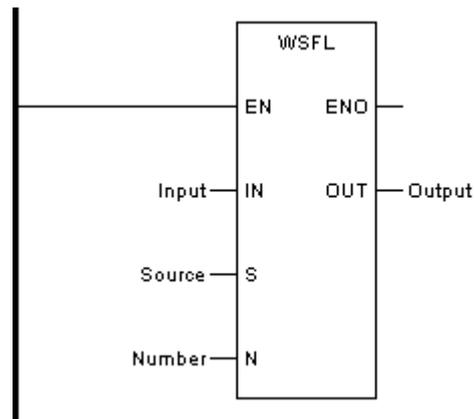
Icon	Parameter	Description	Data type	Point type
IN	Input	Byte to be shifted in	INT	Constant, IW, QW, MW, NW, SW
S	Source	Data to be shifted	INT	MW, NW
N	Number	Number of data	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Byte shifted out	INT	MW, NW

**5.7.11 WSFL: Word shift left**

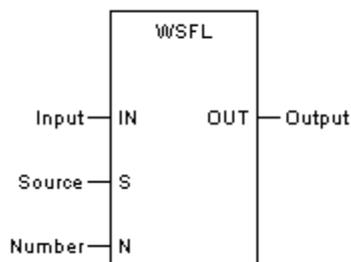
◆ **Function description**

This function block causes a shift to the left by 1 word of N continuous data designated at S. The word to be shifted in is designated at IN, the word shifted out is at OUT.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL WSFL (IN:=Input, S:= Source, N:=Number, OUT=>Output)

◆ **Representation in ST**

WSFL (IN:=Input, S:= Source, N:=Number, OUT=>Output);

◆ **Parameter description**

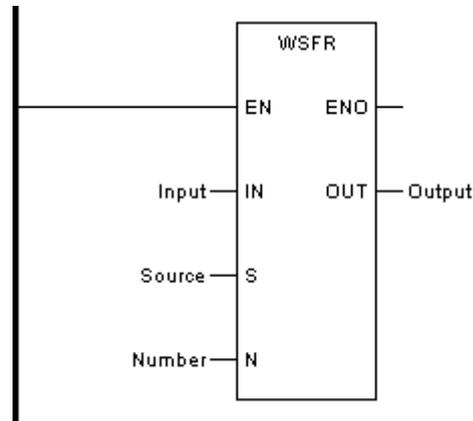
Icon	Parameter	Description	Data type	Point type
IN	Input	Word to be shifted in	INT	Constant, IW, QW, MW, NW, SW
S	Source	Data to be shifted	INT	MW, NW
N	Number	Number of data	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Word shifted out	INT	MW, NW

## 5.7.12 WSFR: Word shift right

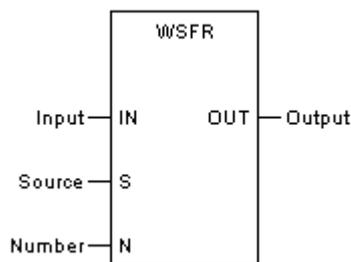
◆ **Function description**

This function block causes a shift to the right by 1 word of N continuous data designated at S. The word to be shifted in is designated at IN, the word shifted out is at OUT.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL WSFR (IN:=Input, S:= Source, N:=Number, OUT=>Output)

◆ **Representation in ST**

WSFR (IN:=Input, S:= Source, N:=Number, OUT=>Output);

◆ **Parameter description**

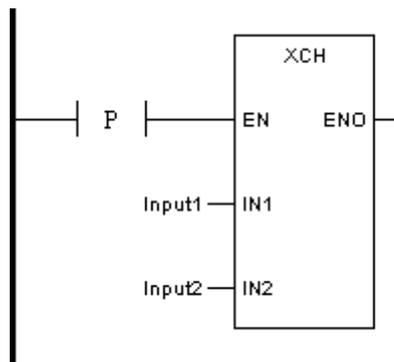
Icon	Parameter	Description	Data type	Point type
IN	Input	Word to be shifted in	INT	Constant, IW, QW, MW, NW, SW
S	Source	Data to be shifted	INT	MW, NW
N	Number	Number of data	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Word shifted out	INT	MW, NW

### 5.7.13 XCH: 16-bit data exchange

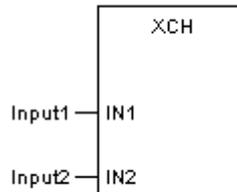
#### ◆ Function description

This function block exchanges the 16-bit data of IN1 and IN2.

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

```
CAL XCH (IN1:=Input1, IN2:=Input2)
```

#### ◆ Representation in ST

```
XCH (IN1:=Input1, IN2:=Input2);
```

#### ◆ Parameter description

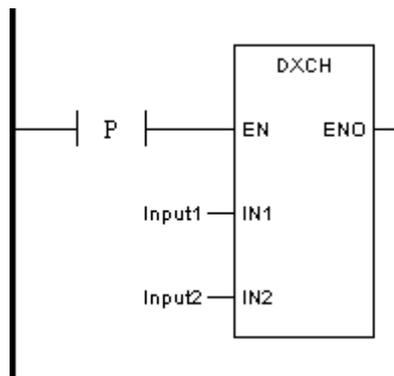
Icon	Parameter	Description	Data type	Point type
IN1	Input1	Data to be exchanged	INT	MW, NW
IN2	Input2	Data to be exchanged	INT	MW, NW

### 5.7.14 DXCH: 32-bit data exchange

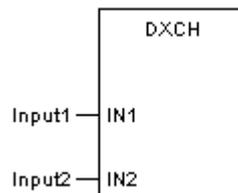
◆ **Function description**

This function block exchanges the 32-bit data of IN1 and IN2. Input can only be 32-bit, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL DXCH (IN1:=Input1, IN2:=Input2)

◆ **Representation in ST**

DXCH (IN1:=Input1, IN2:=Input2);

◆ **Parameter description**

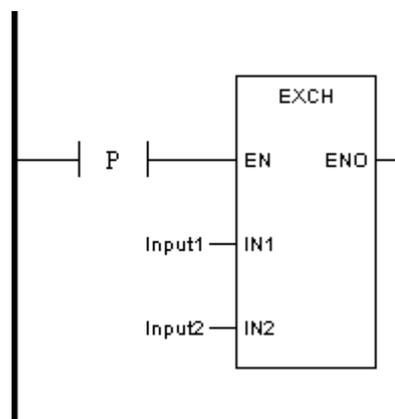
Icon	Parameter	Description	Data type	Point type
IN1	Input1	Data to be exchanged	DINT	MW, NW
IN2	Input2	Data to be exchanged	DINT	MW, NW

## 5.7.15 EXCH: Floating-point data exchange

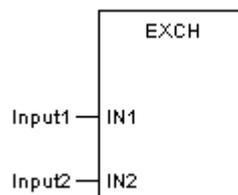
### ◆ Function description

This function block exchanges the floating-point data of IN1 and IN2. Input can only be floating-point, occupying 2 continuous word registers. The serial number of word register must be odd.

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Representation in IL

```
CAL    EXCH (IN1:=Input1, IN2:=Input2)
```

### ◆ Representation in ST

```
EXCH (IN1:=Input1, IN2:=Input2);
```

### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
IN1	Input1	Data to be exchanged	REAL	MW, NW

IN2	Input2	Data to be exchanged	REAL	MW, NW
-----	--------	----------------------	------	--------

## 5.8 Timer

This chapter describes the elementary function blocks of the Timer family.

This chapter contains the following sections.

Type	Description
<b>TON</b>	On Delay Timer (Second)
<b>TOF</b>	Off Delay Timer (Second)
<b>TP</b>	Pulse Timer (Second)
<b>TON_MS</b>	On Delay Timer (Millisecond)
<b>TOF_MS</b>	Off Delay Timer (Millisecond)
<b>TP_MS</b>	Pulse Timer (Millisecond)
<b>TON_M</b>	On Delay Timer (Minute)
<b>TOF_M</b>	Off Delay Timer (Minute)
<b>TP_M</b>	Pulse Timer (Minute)
<b>TON_H</b>	On Delay Timer (Hour)
<b>TOF_H</b>	Off Delay Timer (Hour)
<b>TP_H</b>	Pulse Timer (Hour)

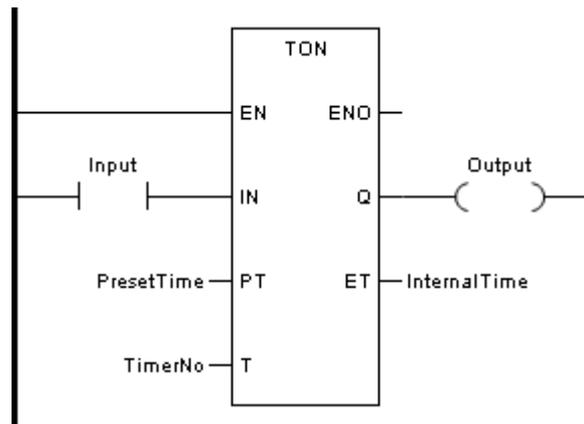
### 5.8.1 TON: On delay timer

#### ◆ Function description

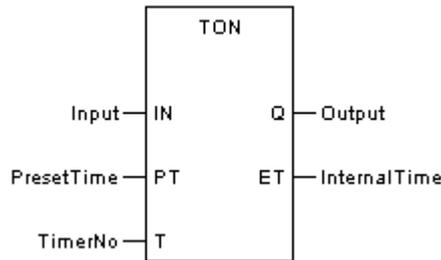
This function block is used as the On delay. When the function block is called for the first time, the initial state of ET is "0".

TON, TON\_MS, TON\_M and TON\_H are identical except for their units.

#### ◆ Representation in LD



◆ **Representation in FBD**



◆ **Representation in IL**

CAL TON (IN:=Input, PT:=PresetTime, T:=TimerNo, Q=>Output,  
ET=>InternalTime) (ET can be omitted)

◆ **Representation in ST**

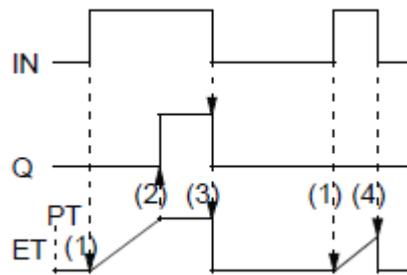
TON (IN:=Input, PT:=PresetTime, T:=TimerNo, Q=>Output, ET=>InternalTime);  
(ET can be omitted)

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Start delay	BOOL	Constant, I, Q, M, N, S
PT	PresetTime	Preset delay time	INT	Constant, IW, QW, MW, NW, SW
T	TimerNo	Timer No.		T
Q	Output	Output	BOOL	Q, M, N
ET	InternalTime	Internal time	INT	MW, NW

◆ **Timing Diagram**

Representation of the On delay TON:



- (1) If IN becomes “1”, the internal time (ET) starts.
- (2) If the internal time reaches the value of PT, Q becomes “1”.
- (3) If IN becomes “0”, Q becomes “0” and the internal time stops/resets.
- (4) If IN becomes “0” before the internal time has reached the value of PT, the internal time stops/resets without Q going to “1”.

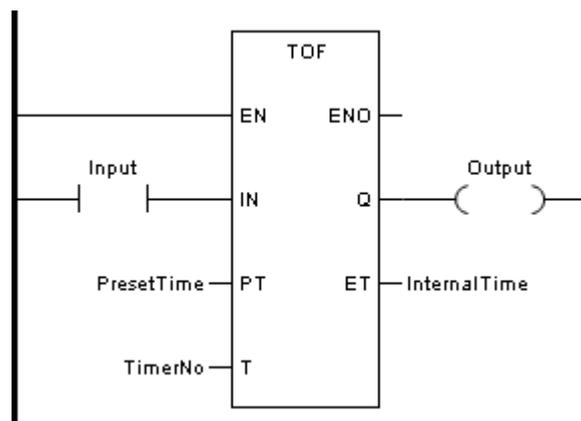
## 5.8.2 TOF: Off delay timer

### ◆ Function description

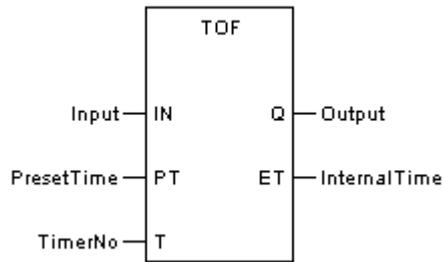
This function block is used as the Off delay. When the function block is called for the first time, the initial state of ET is “0”.

TOF, TOF\_MS, TOF\_M and TOF\_H are identical except for their units.

### ◆ Representation in LD



### ◆ Representation in FBD



◆ **Representation in IL**

CAL TOF (IN:=Input, PT:=PresetTime, T:=TimerNo, Q=>Output,  
ET=>InternalTime) (ET can be omitted)

◆ **Representation in ST**

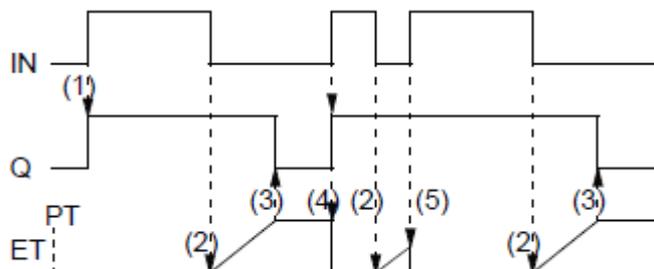
TOF (IN:=Input, PT:=PresetTime, T:=TimerNo, Q=>Output, ET=>InternalTime);  
(ET can be omitted)

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Start delay	BOOL	Constant, I, Q, M, N, S
PT	PresetTime	Preset delay time	INT	Constant, IW, QW, MW, NW, SW
T	TimerNo	Timer No.		T
Q	Output	Output	BOOL	Q, M, N
ET	InternalTime	Internal time	INT	MW, NW

◆ **Timing Diagram**

Representation of the Off delay TOF:



- (1) If IN becomes "1", Q becomes "1".
- (2) If IN becomes "0", the internal time (ET) starts.

- (3) If the internal time reaches the value of PT, Q becomes “0”.
- (4) If IN becomes “1”, Q becomes “1” and the internal time stops/resets.
- (5) If IN becomes “1” before the internal time has reached the value of PT, the internal time stops/resets without Q being set back to “0”.

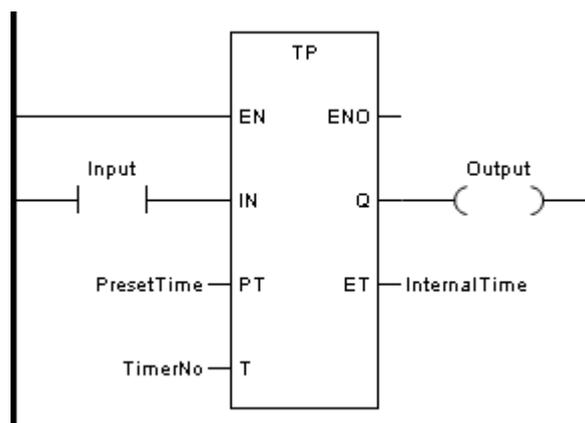
### 5.8.3 TP: Pulse timer

#### ◆ Function description

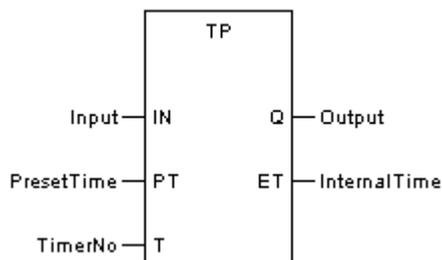
This function block is used for the generation of a pulse with defined duration. When the function block is called for the first time, the initial state of ET is “0”.

TP, TP\_MS, TP\_M and TP\_H are identical except for their units.

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

```
CAL TP(IN:=Input, PT:=PresetTime, T:=TimerNo, Q=>Output,
      ET=>InternalTime) (ET can be omitted)
```

◆ **Representation in ST**

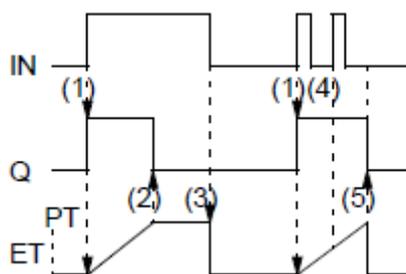
TP (IN:=Input, PT:=PresetTime, T:=TimerNo, Q=>Output, ET=>InternalTime);  
 (ET can be omitted)

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Trigger pulse	BOOL	Constant, I, Q, M, N, S
PT	PresetTime	Preset pulse duration	INT	Constant, IW, QW, MW, NW, SW
T	TimerNo	Timer No.		T
Q	Output	Output	BOOL	Q, M, N
ET	InternalTime	Internal time	INT	MW, NW

◆ **Timing Diagram**

Representation of the TP pulse:



- (1) If IN becomes "1", Q becomes "1" and the internal time (ET) starts.
- (2) If the internal time reaches the value of PT, Q becomes "0" (Independent of IN).
- (3) The internal time stops/resets if IN becomes "0".
- (4) If the internal time has not reached the value of PT yet, the internal time is not affected by a clock at IN.
- (5) If the internal time has reached the value of PT and IN is "0", the internal time stops/resets and Q becomes "0".

**5.9 Counter**

This chapter describes the elementary function blocks of the Counter family. This chapter contains the following sections.

Type	Description
<b>CTU</b>	Up Counter
<b>CTD</b>	Down Counter
<b>CTUD</b>	Up/Down Counter

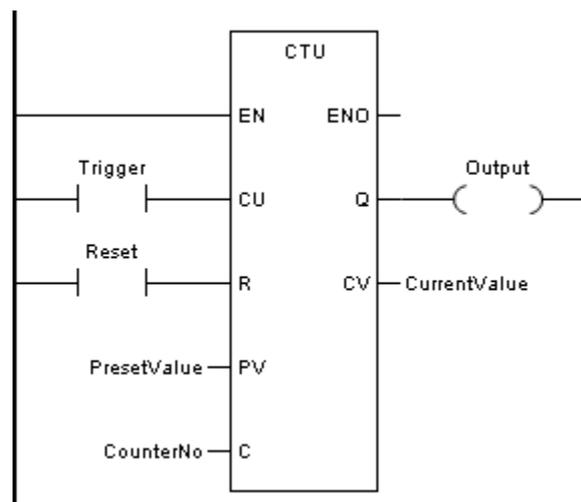
### 5.9.1 CTU: Up counter

#### ◆ Function description

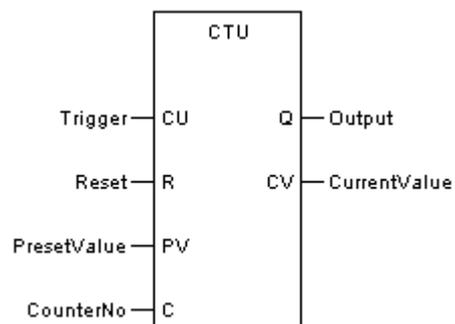
This function block is used for upwards counting.

A “1” signal at the R input caused the value “0” to be assigned to the CV output. With each transition from “0” to “1” at the CU input, the value of CV is incremented by 1. When  $CV \geq PV$ , the Q output is set to “1”.

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

CAL CTU (CU:=Trigger, R:=Reset, PV:=PresetValue, C:=CounterNo, Q=>Output, CV=>CurrentValue) (CV can be omitted)

◆ **Representation in ST**

CTU (CU:=Trigger, R:=Reset, PV:=PresetValue, C:=CounterNo, Q=>Output, CV=>CurrentValue); (CV can be omitted)

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
CU	Trigger	Trigger input	BOOL	Constant, I, Q, M, N, S
R	Reset	Reset	BOOL	Constant, I, Q, M, N, S
PV	PresetValue	Preset value	INT	Constant, IW, QW, MW, NW, SW
C	CounterNo	Counter No.		C
Q	Output	Output	BOOL	Q, M, N
CV	CurrentValue	Current value	INT	MW, NW

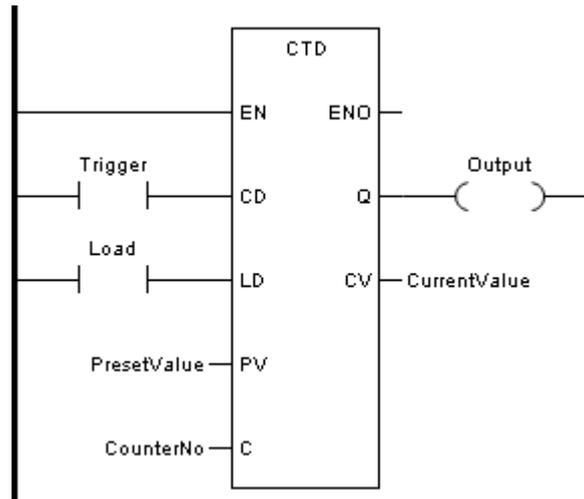
**5.9.2 CTD: Down counter**

◆ **Function description**

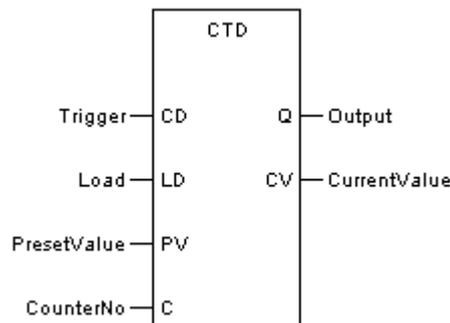
This function block is used for downwards counting.

A “1” signal at the LD input caused the value of the PV input to be allocated to the CV output. With each transition from “0” to “1” at the CD input, the value of CV is decremented by 1. When  $CV \leq 0$ , the Q output is set to “1”.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL CTD (CD:=Trigger, LD:=Load, PV:=PresetValue, C:=CounterNo, Q=>Output, CV=>CurrentValue) (CV can be omitted)

◆ **Representation in ST**

CTD (CD:=Trigger, LD:=Load, PV:=PresetValue, C:=CounterNo, Q=>Output, CV=>CurrentValue); (CV can be omitted)

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
CD	Trigger	Trigger input	BOOL	Constant, I, Q, M, N, S
LD	Load	Load	BOOL	Constant, I, Q, M, N, S
PV	PresetValue	Preset value	INT	Constant, IW, QW, MW, NW, SW
C	CounterNo	Counter No.		C

Q	Output	Output	BOOL	Q, M, N
CV	CurrentValue	Current value	INT	MW, NW

### 5.9.3 CTUD: Up/down counter

#### ◆ Function description

This function block is used for upwards and downwards counting.

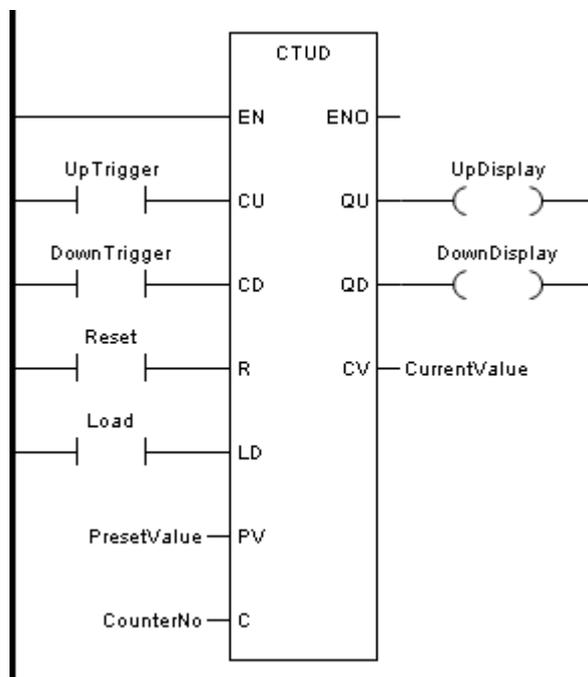
A “1” signal at the R input caused the value “0” to be assigned to the CV output. A “1” signal at the LD input caused the value of the PV input to be allocated to the CV output. With each transition from “0” to “1” at the CU input, the value of CV is incremented by 1. With each transition from “0” to “1” at the CD input, the value of CV is decremented by 1.

If there is a simultaneous “1” signal at inputs R and LD, input R has precedence.

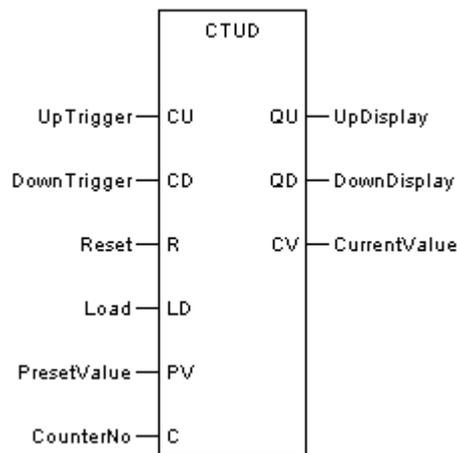
When  $CV \geq PV$ , the QU output is set to “1”.

When  $CV \leq 0$ , the QD output is set to “1”.

#### ◆ Representation in LD



#### ◆ Representation in FBD



◆ **Representation in IL**

CAL CTUD (CU:=UpTrigger, CD:=DownTrigger, R:=Reset, LD:=Load, PV:=PresetValue, C:=CounterNo, QU=>UpDisplay, QD=>DownDisplay, CV=>CurrentValue) (CV can be omitted)

◆ **Representation in ST**

CTUD (CU:=UpTrigger, CD:=DownTrigger, R:=Reset, LD:=Load, PV:=PresetValue, C:=CounterNo, QU=>UpDisplay, QD=>DownDisplay, CV=>CurrentValue); (CV can be omitted)

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
CU	UpTrigger	Up counter trigger input	BOOL	Constant, I, Q, M, N, S
CD	DownTrigger	Down counter trigger input	BOOL	Constant, I, Q, M, N, S
R	Reset	Reset	BOOL	Constant, I, Q, M, N, S
LD	Load	Load	BOOL	Constant, I, Q, M, N, S
PV	PresetValue	Preset value	INT	Constant, IW, QW, MW, NW, SW
C	CounterNo	Counter No.		C
QU	UpDisplay	Up display	BOOL	Q, M, N
QD	DownDisplay	Down display	BOOL	Q, M, N
CV	CurrentValue	Current value	INT	MW, NW

## 5.10 Control

This chapter describes the elementary function blocks of the Control family. This chapter contains the following sections.

Type	Description
<b>CALL</b>	Call Program

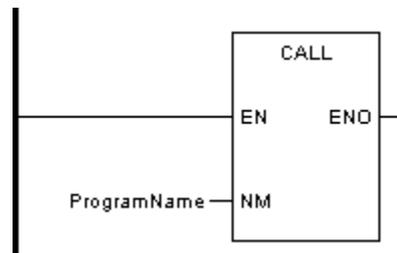
### 5.10.1 CALL: Call program

#### ◆ Function description

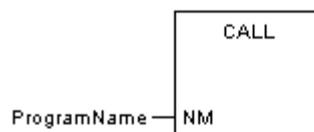
This function block calls the corresponding subroutine (other LD, FBD, IL, ST program).

When the function block is called, it causes the scan to go immediately to the designated subroutine and execute it. After the subroutine execution is complete, control returns to the rung in the logic immediately following the CALL function block.

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

CAL    ProgramName

#### ◆ Representation in ST

ProgramName ();

### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
NM	ProgramName	Program name		

## 5.11 PLC

This chapter describes the elementary function blocks of the PLC family.

This chapter contains the following sections.

Type	Description
<b>PULSE</b>	Pulse Digital Output
<b>AOUT</b>	Analog Output
<b>FORCE</b>	I/O Force
<b>UNFORCE</b>	I/O Unforce
<b>RESH</b>	I/O Refresh
<b>ENI</b>	Interrupt Enable
<b>DISI</b>	Interrupt Disable
<b>ATCH</b>	Interrupt Attach
<b>DTCH</b>	Interrupt Disattach
<b>PWM</b>	Pulse Width Modulation Output
<b>PWM1</b>	Pulse Width Modulation Output 1
<b>PLSY</b>	High Speed Pulse Output
<b>PLSY1</b>	High Speed Pulse Output 1
<b>PLSR</b>	Acceleration-Deceleration High Speed Pulse Output
<b>PLSR1</b>	Acceleration-Deceleration High Speed Pulse Output 1
<b>PLSR2</b>	Acceleration-Deceleration High Speed Pulse Output 2
<b>ORG</b>	Origin Search
<b>READ</b>	Special Data Read

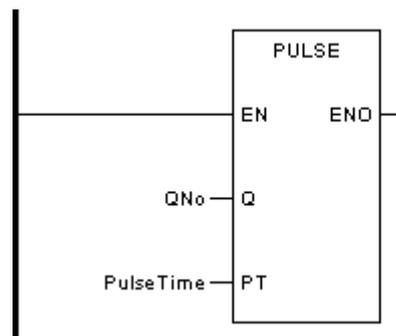
<b>WRITE</b>	Special Data Write
<b>XMT</b>	Free Port Transmit
<b>RCV</b>	Free Port Receive
<b>FLASH</b>	Flash Data Read/Write
<b>RTC</b>	Real-Time Clock
<b>MODRW</b>	Modbus Data Read/Write

### 5.11.1 PULSE: Pulse digital output

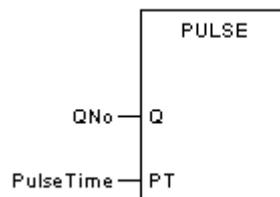
#### ◆ Function description

This function block sets a digital output point to “1” and holds for specified time and then turns to “0” automatically.

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

CAL PULSE (Q:=QNo, PT:=PulseTime)

#### ◆ Representation in ST

PULSE (Q:=QNo, PT:=PulseTime);

#### ◆ Parameter description

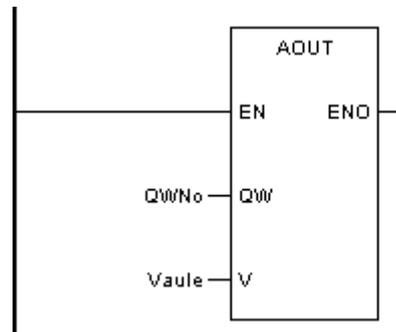
Icon	Parameter	Description	Data type	Point type
Q	QNo	Pulse digital output point		Q
PT	PulseTime	Pulse time, unit: ms, 1~30000	INT	Constant, IW, QW, MW, NW, SW

### 5.11.2 AOUT: Analog output

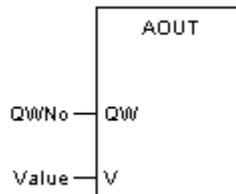
#### ◆ Function description

This function block sets an analog output point to a specified value.

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

CAL AOUT (QW:=QWNo:=Value)

#### ◆ Representation in ST

AOUT (QW:=QWNo:=Value);

#### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
------	-----------	-------------	-----------	------------

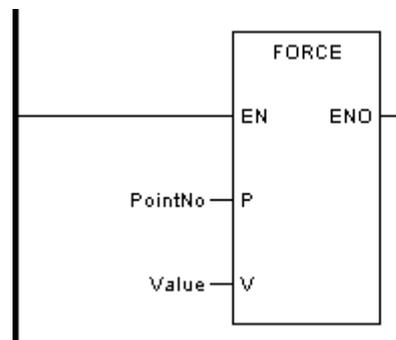
QW	QWNo	Analog output point		QW
V	Value	Specified value	INT	Constant, IW, QW, MW, NW, SW

### 5.11.3 FORCE: I/O force

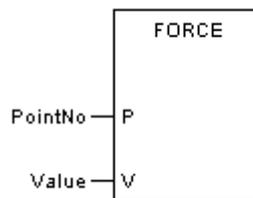
#### ◆ Function description

This function block forces a point (I, Q, IW, QW) to a specified value.

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

```
CAL FORCE (P:=PointNo, V:=Value)
```

#### ◆ Representation in ST

```
FORCE (P:=PointNo, V:=Value);
```

#### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
P	PointNo	Point to be forced		I, Q, IW, QW

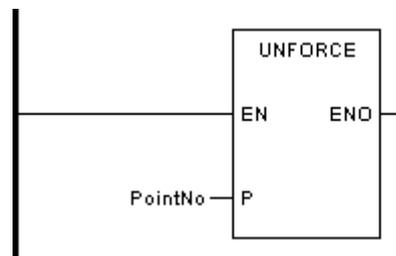
V	Value	Value to be forced	BOOL, INT	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
---	-------	--------------------	-----------	---------------------------------------------

### 5.11.4 UNFORCE: I/O unforce

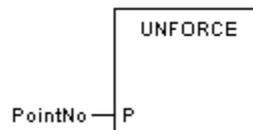
#### ◆ Function description

This function block unforces a point (I, Q, IW, QW).

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

```
CAL UNFORCE (PointNo)
```

#### ◆ Representation in ST

```
UNFORCE (PointNo);
```

#### ◆ Parameter description

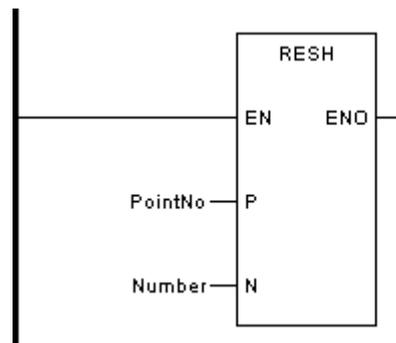
Icon	Parameter	Description	Data type	Point type
P	PointNo	Point to be unforced		I, Q, IW, QW

### 5.11.5 RESH: I/O refresh

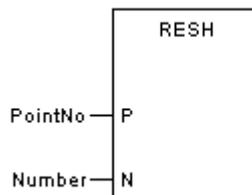
◆ **Function description**

This function block refreshes the state of digital input and digital output immediately.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL RESH (P:=PointNo, N:=Number)

◆ **Representation in ST**

RESH (P:=PointNo, N:=Number);

◆ **Parameter description**

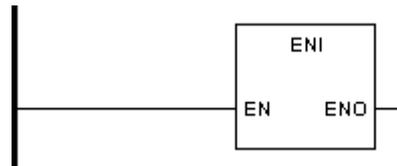
Icon	Parameter	Description	Data type	Point type
P	PointNo	Point to be refreshed		I, Q
N	Number	Number of points	INT	Constant, IW, QW, MW, NW, SW

## 5.11.6 ENI: Interrupt enable

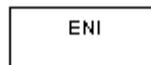
### ◆ Function description

This function block enables PLC interrupt. Once the function block is called, when the interrupt occurs, the condition to call the corresponding interrupt program is met.

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Representation in IL

```
CAL ENI ()
```

### ◆ Representation in ST

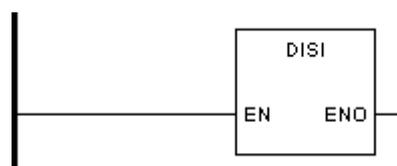
```
ENI ();
```

## 5.11.7 DISI: Interrupt disable

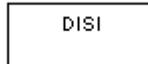
### ◆ Function description

This function block disables PLC interrupt. Once the function block is called, when the interrupt occurs, the condition to call the corresponding interrupt program is not met.

### ◆ Representation in LD



### ◆ Representation in FBD



◆ **Representation in IL**

CAL DISI ()

◆ **Representation in ST**

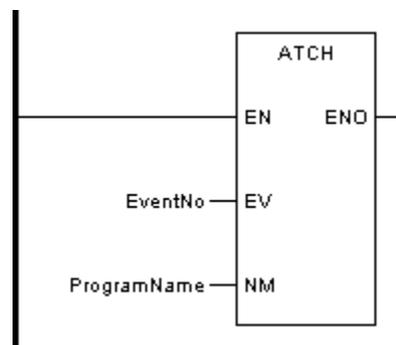
DISI ();

### 5.11.8 ATCH: Interrupt attach

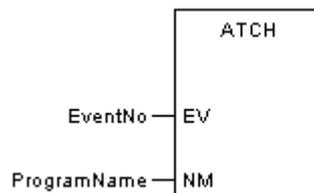
◆ **Function description**

This function block attaches an interrupt program to an interrupt event. When the interrupt occurs, it causes the scan to go immediately to the attached program and execute it.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL ATCH (EV:=EventNo, NM:=ProgramName)

◆ **Representation in ST**

ATCH (EV:=EventNo, NM:=ProgramName);

◆ **Parameter description**

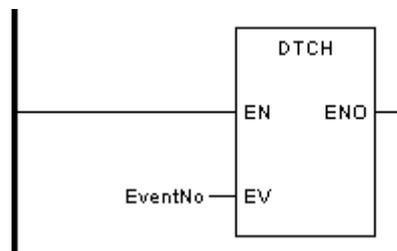
Icon	Parameter	Description	Data type	Point type
EV	EventNo	Interrupt event No.	INT	Constant, IW, QW, MW, NW, SW
NM	ProgramName	Interrupt program name		

**5.11.9 DTCH: Interrupt disattach**

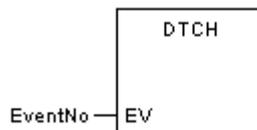
◆ **Function description**

This function block disattaches the attached program to the corresponding interrupt event.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL DTCH (EventNo)

◆ **Representation in ST**

DTCH (EventNo);

◆ **Parameter description**

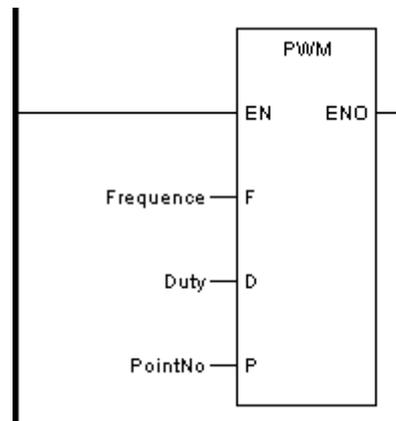
Icon	Parameter	Description	Data type	Point type
EV	EventNo	Interrupt event No.	INT	Constant, IW, QW, MW, NW, SW

**5.11.10 PWM: Pulse width modulation output**

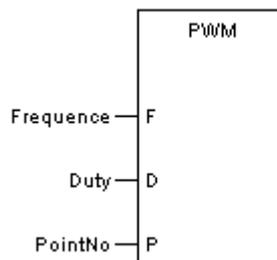
◆ **Function description**

This function block outputs the pulse of frequency specified in F and duty ratio specified in D for point specified in P.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL    PWM (IN:=Enable, F:=Frequency, D:=Duty, P:=PointNo)

◆ **Representation in ST**

PWM (IN:=Enable, F:=Frequency, D:=Duty, P:=PointNo);

◆ **Parameter description**

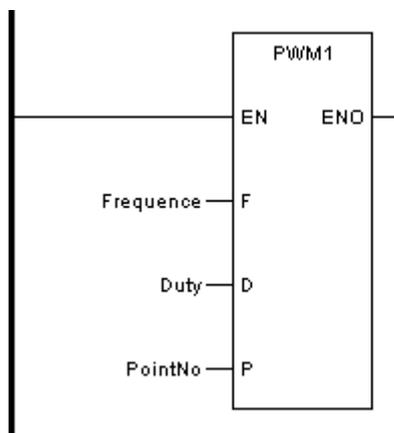
Icon	Parameter	Description	Data type	Point type
F	Frequence	Pulse output frequency, unit: 10Hz, 2~10000	INT	Constant, IW, QW, MW, NW, SW
D	Duty	Pulse output duty ratio, 0~100	INT	Constant, IW, QW, MW, NW, SW
P	PointNo	Point No.		Q

**5.11.11 PWM1: Pulse width modulation output 1**

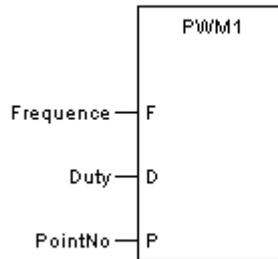
◆ **Function description**

This function block outputs the pulse of frequency specified in F and duty ratio specified in D for point specified in P.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL PWM1 (IN:=Enable, F:=Frequence, D:=Duty, P:=PointNo)

◆ **Representation in ST**

PWM1 (IN:=Enable, F:=Frequence, D:=Duty, P:=PointNo);

◆ **Parameter description**

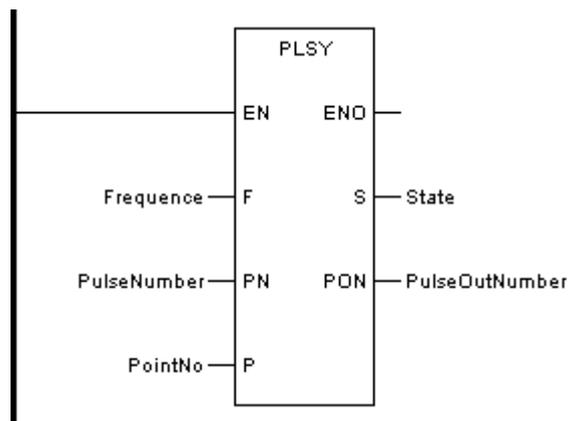
Icon	Parameter	Description	Data type	Point type
F	Frequence	Pulse output frequency, unit: Hz, 12~100000	INT	Constant, IW, QW, MW, NW, SW
D	Duty	Pulse output duty ratio, 0~100	INT	Constant, IW, QW, MW, NW, SW
P	PointNo	Point No.		Q

### 5.11.12 PLSY: High speed pulse output

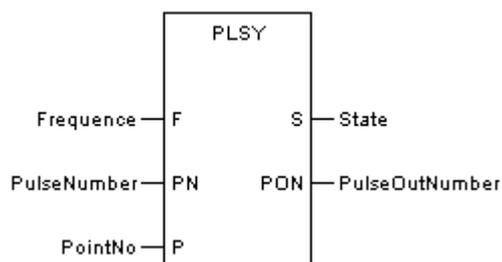
◆ **Function description**

This function block outputs the pulse of frequency specified in F and number specified in PN for point specified in P.

◆ **Representation in LD**



◆ Representation in FBD



◆ Representation in IL

CAL PLSY (IN:=Enable, F:=Frequency, PN:=PulseNumber, P:=PointNo, S=>State, PON=>PulseOutNumber)

◆ Representation in ST

PLSY (IN:=Enable, F:=Frequency, PN:=PulseNumber, P:=PointNo, S=>State, PON=>PulseOutNumber);

◆ Parameter description

Icon	Parameter	Description	Data type	Point type
F	Frequence	Pulse output frequency, unit: 10Hz, 1~20000	INT	Constant, IW, QW, MW, NW, SW
PN	PulseNumber	Pulse number	DINT	Constant, MW, NW
P	PointNo	Point No.		Q
S	State	Pulse output state	BOOL	Q, M, N

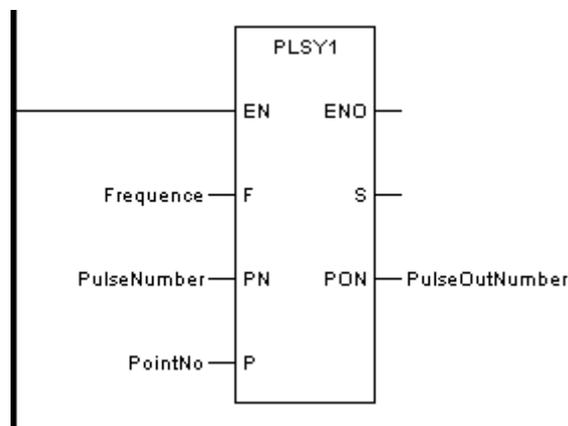
PON	PulseOutNumber	Outputted pulse number	DINT	MW, NW
-----	----------------	------------------------	------	--------

### 5.11.13 PLSY1: High speed pulse output 1

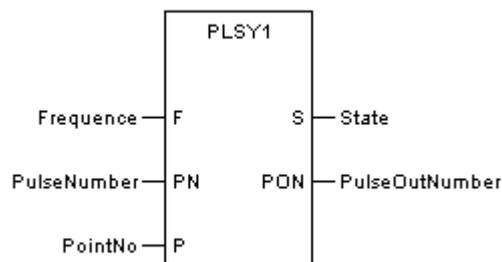
#### ◆ Function description

This function block outputs the pulse of frequency specified in F and number specified in PN for point specified in P.

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

CAL PLSY1 (IN:=Enable, F:=Frequency, PN:=PulseNumber, P:=PointNo, S=>State, PON=>PulseOutNumber)

#### ◆ Representation in ST

PLSY1 (IN:=Enable, F:=Frequency, PN:=PulseNumber, P:=PointNo, S=>State, PON=>PulseOutNumber);

◆ **Parameter description**

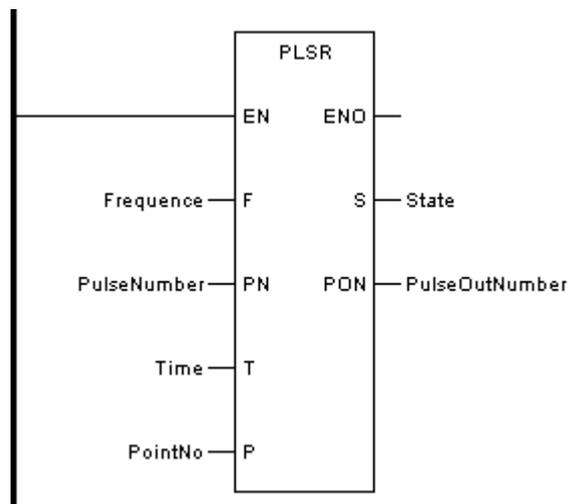
Icon	Parameter	Description	Data type	Point type
F	Frequence	Pulse output frequency, unit: Hz, 6~200000	INT	Constant, IW, QW, MW, NW, SW
PN	PulseNumber	Pulse number	DINT	Constant, MW, NW
P	PointNo	Point No.		Q
S	State	Pulse output state	BOOL	Q, M, N
PON	PulseOutNumber	Outputted pulse number	DINT	MW, NW

**5.11.14 PLSR: Acceleration-deceleration high speed pulse output**

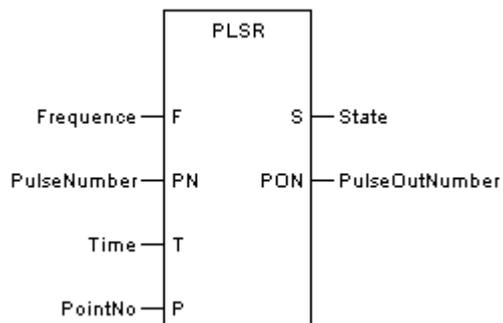
◆ **Function description**

This function block outputs the pulse of variable frequency and number specified in PN for point specified in P. The highest frequency is specified in F, the acceleration time and the deceleration time is specified in T.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL PLSR (IN:=Enable, F:=Frequency, PN:=PulseNumber, T:=Time, P:=PointNo, S=>State, PON=>PulseOutNumber)

◆ **Representation in ST**

PLSR (IN:=Enable, F:=Frequency, PN:=PulseNumber, T:=Time, P:=PointNo, S=>State, PON=>PulseOutNumber);

◆ **Parameter description**

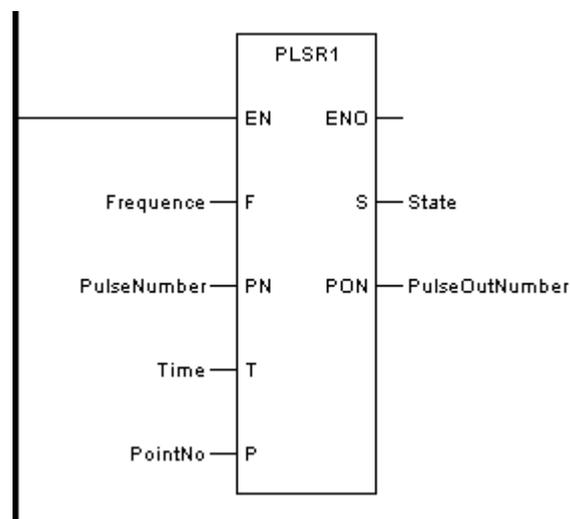
Icon	Parameter	Description	Data type	Point type
F	Frequency	Pulse output frequency, unit: 10Hz, 1~20000	INT	Constant, IW, QW, MW, NW, SW
PN	PulseNumber	Pulse number	DINT	Constant, MW, NW
T	Time	Acceleration and deceleration time, unit: ms, 5~5000	INT	Constant, IW, QW, MW, NW, SW
P	PointNo	Point No.		Q
S	State	Pulse output state	BOOL	Q, M, N
PON	PulseOutNumber	Outputted pulse number	DINT	MW, NW

## 5.11.15 PLSR1: Acceleration-deceleration high speed pulse output 1

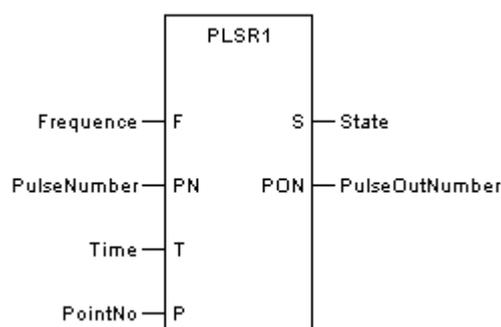
### ◆ Function description

This function block outputs the pulse of variable frequency and number specified in PN for point specified in P. The highest frequency is specified in F, the acceleration time and the deceleration time is specified in T.

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Representation in IL

CAL PLSR1 (IN:=Enable, F:=Frequency, PN:=PulseNumber, T:=Time, P:=PointNo, S=>State, PON=>PulseOutNumber)

### ◆ Representation in ST

PLSR1 (IN:=Enable, F:=Frequency, PN:=PulseNumber, T:=Time, P:=PointNo,  
S=>State, PON=>PulseOutNumber);

### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
F	Frequency	Pulse output frequency, unit: Hz, 6~200000	INT	Constant, IW, QW, MW, NW, SW
PN	PulseNumber	Pulse number	DINT	Constant, MW, NW
T	Time	Acceleration and deceleration time, unit: ms, 5~5000	INT	Constant, IW, QW, MW, NW, SW
P	PointNo	Point No.		Q
S	State	Pulse output state	BOOL	Q, M, N
PON	PulseOutNumber	Outputted pulse number	DINT	MW, NW

## 5.11.16 PLSR2: Acceleration-deceleration high speed pulse output 2

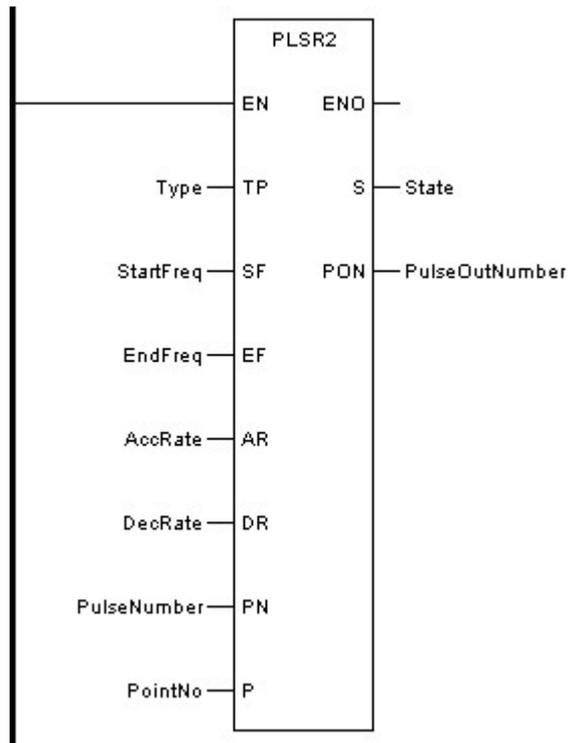
### ◆ Function description

This function block outputs the pulse of startup frequency specified in SF, type specified in TP and number specified in PN for point specified in P.

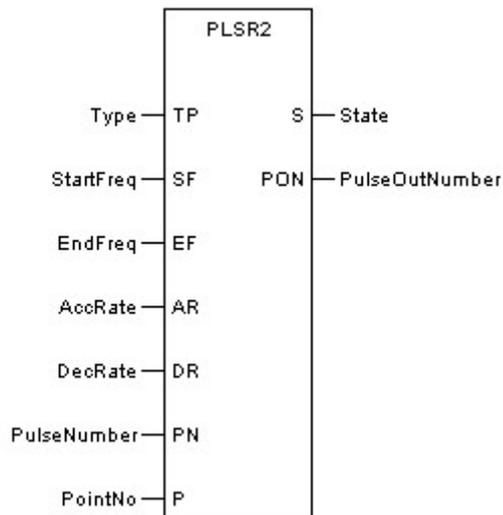
In every pulse control cycle (4ms), the frequency increases in the specified acceleration rate AR, until it reaches the target frequency specified in EF. After the frequency reaches the target, it will stop accelerating and continue to pulse output in a uniform velocity.

When the number of pulse output reaches the point of deceleration, the frequency decreases in the specified deceleration rate DR until it reaches the startup frequency when the output stops.

### ◆ Representation in LD



◆ **Representation in FBD**



◆ **Representation in IL**

CAL PLSR2 (IN:=Enable, TP:=Type, SF:=StartFreq, EF:=EndFreq,  
 AR:=AccRate, DR:=DecRate, PN:=PulseNumber, P:=PointNo,  
 S=>State, PON=>PulseOutNumber)

◆ **Representation in ST**

PLSR2 (IN:=Enable, TP:=Type, SF:=StartFreq, EF:=EndFreq, AR:=AccRate,  
DR:=DecRate, PN:=PulseNumber, P:=PointNo, S=>State,  
PON=>PulseOutNumber);

### ◆ Parameter description

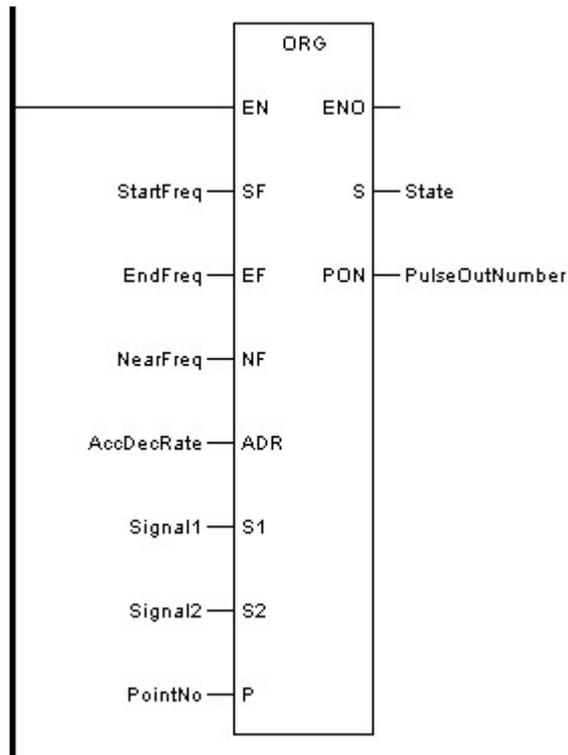
Icon	Parameter	Description	Data type	Point type
TP	Type	Pulse output type	INT	Constant, IW, QW, MW, NW, SW
SF	StartFreq	Startup frequency, unit: 10Hz, 1~20000	INT	Constant, IW, QW, MW, NW, SW
EF	EndFreq	Target frequency, unit: 10Hz, 1~20000	INT	Constant, IW, QW, MW, NW, SW
AR	AccRate	Accelerate rate	INT	Constant, IW, QW, MW, NW, SW
DR	DecRate	Decelerate rate	INT	Constant, IW, QW, MW, NW, SW
PN	PulseNumber	Pulse number	DINT	Constant, MW, NW
P	PointNo	Point No.		Q
S	State	Pulse output state	BOOL	Q, M, N
PON	PulseOutNumber	Outputted pulse number	DINT	MW, NW

## 5.11.17 ORG: Origin search

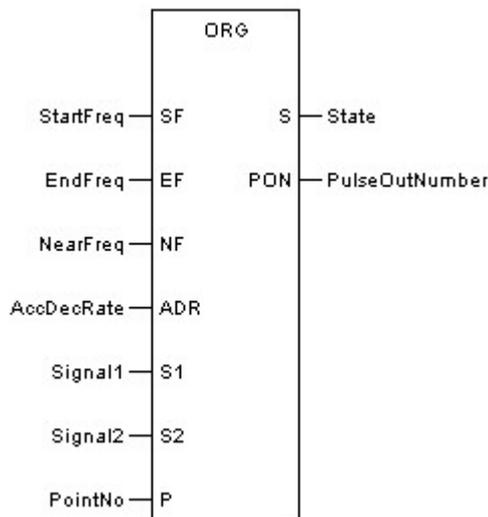
### ◆ Function description

This function block is used to search the origin according to the specified mode.

### ◆ Representation in LD



◆ **Representation in FBD**



◆ **Representation in IL**

CAL    ORG (IN:=Enable, SF:=StartFreq, EF:=EndFreq, NF:=NearFreq,  
          ADR:=AccDecRate, S1:=Signal1, S2:=Signal2, P:=PointNo, S=>State,  
          PON=>PulseOutNumber)

◆ **Representation in ST**

ORG (IN:=Enable, SF:=StartFreq, EF:=EndFreq, NF:=NearFreq,  
 ADR:=AccDecRate, S1:=Signal1, S2:=Signal2, P:=PointNo, S=>State,  
 PON=>PulseOutNumber);

◆ **Parameter description**

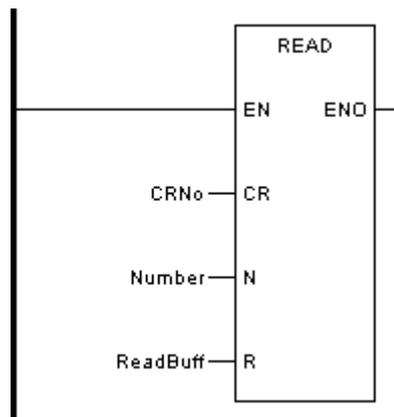
Icon	Parameter	Description	Data type	Point type
SF	StartFreq	Startup frequency, unit: 10Hz, 1~20000	INT	Constant, IW, QW, MW, NW, SW
EF	EndFreq	Target frequency, unit: 10Hz, 1~20000	INT	Constant, IW, QW, MW, NW, SW
NF	NearFreq	Near origin frequency, unit: 10Hz, 1~20000	INT	Constant, IW, QW, MW, NW, SW
ADR	AccDecRate	Accelerate and decelerate rate	INT	Constant, IW, QW, MW, NW, SW
S1	Signal1	External signal 1	BOOL	Constant, I, Q, M, N, S
S2	Signal2	External signal 2	BOOL	Constant, I, Q, M, N, S
P	PointNo	Point No.		Q
S	State	Pulse output state	BOOL	Q, M, N
PON	PulseOutNumber	Outputted pulse number	DINT	MW, NW

**5.11.18 READ: Special data read**

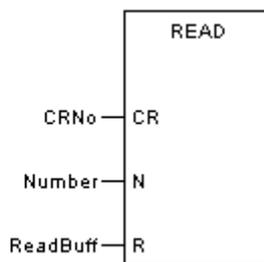
◆ **Function description**

This function block reads special CR data, such as high speed counter.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL READ (CR:=CRNo, N:=Number, R:=ReadBuff)

◆ **Representation in ST**

READ (CR:=CRNo, N:=Number, R:=ReadBuff);

◆ **Parameter description**

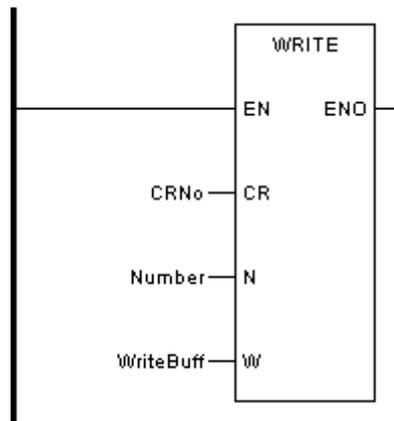
Icon	Parameter	Description	Data type	Point type
CR	CRNo	Start No. of CR	INT	Constant, IW, QW, MW, NW, SW
N	Number	Number of CR	INT	Constant, IW, QW, MW, NW, SW
R	ReadBuff	Data buffer	INT	MW, NW

**5.11.19 WRITE: Special data write**

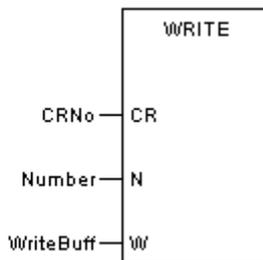
◆ **Function description**

This function block writes special CR data, such as high speed counter.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL WRITE (CR:=CRNo, N:=Number, W:=WriteBuff)

◆ **Representation in ST**

WRITE (CR:=CRNo, N:=Number, W:=WriteBuff);

◆ **Parameter description**

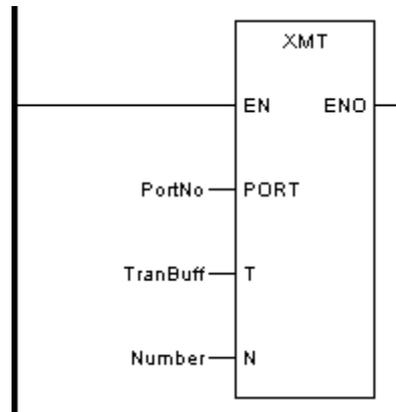
Icon	Parameter	Description	Data type	Point type
CR	CRNo	Start No. of CR	INT	Constant, IW, QW, MW, NW, SW
N	Number	Number of CR	INT	Constant, IW, QW, MW, NW, SW
W	WriteBuff	Data buffer	INT	MW, NW

**5.11.20 XMT: Free port transmit**

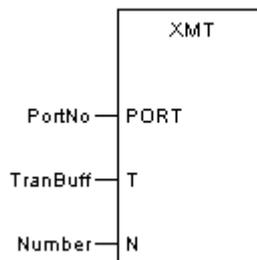
◆ **Function description**

This function block is used to transmit data in free port mode of COM1/COM2.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL XMT (PORT:=PortNo, T:=TranBuff, N:=Number)

◆ **Representation in ST**

XMT (PORT:=PortNo, T:=TranBuff, N:=Number);

◆ **Parameter description**

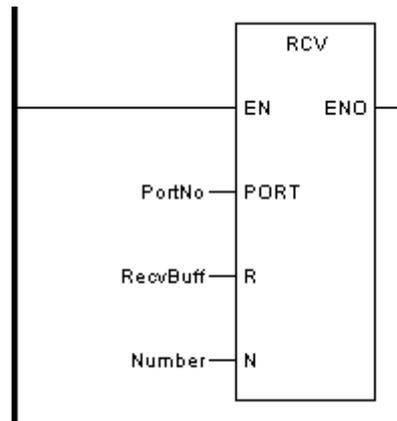
Icon	Parameter	Description	Data type	Point type
PORT	PortNo	COM No. (1 or 2)	INT	Constant, IW, QW, MW, NW, SW
T	TranBuff	Data buffer to be sent	INT	MW, NW
N	Number	Number of bytes to be sent	INT	Constant, IW, QW, MW, NW, SW

### 5.11.21 RCV: Free port receive

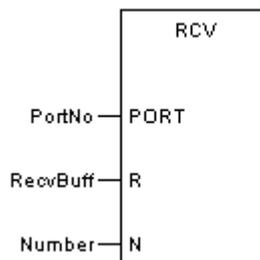
◆ **Function description**

This function block is used to receive data in free port mode of COM1/COM2.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL RCV (PORT:=PortNo, R:=RecvBuff, N:=Number)

◆ **Representation in ST**

RCV (PORT:=PortNo, R:=RecvBuff, N:=Number);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
PORT	PortNo	COM No. (1 or 2)	INT	Constant, IW, QW, MW, NW, SW
R	RecvBuff	Data buffer to be	INT	MW, NW

		received		
N	Number	Number of bytes to be received	INT	Constant, IW, QW, MW, NW, SW

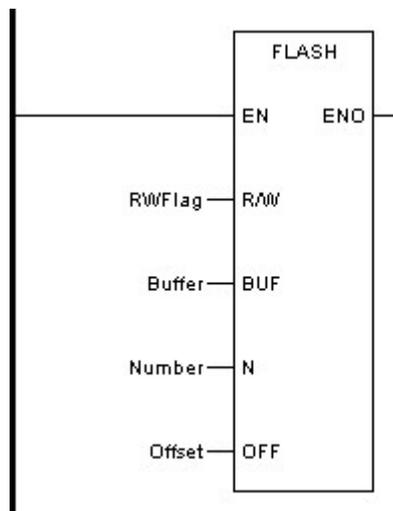
### 5.11.22 FLASH: Flash data read/write

#### ◆ Function description

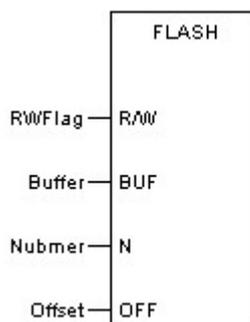
This function block is used to read or write FLASH data.

This function block can not be called frequently, prevent damage to FLASH.

#### ◆ Representation in LD



#### ◆ Representation in FBD



#### ◆ Representation in IL

CAL FLASH (R/W:=RWFlag, BUF:=Buffer, N:=Number, OFF:=Offset)

◆ **Representation in ST**

FLASH (R/W:=RWFlag, BUF:=Buffer, N:=Number, OFF:=Offset);

◆ **Parameter description**

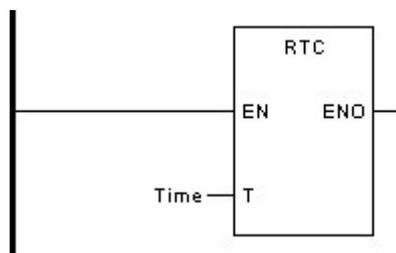
Icon	Parameter	Description	Data type	Point type
R/W	RWFlag	Read or write flag, 1: write, 0: read	BOOL	Constant, I, Q, M, N, S
BUF	Buffer	Data buffer	INT	MW, NW
N	Number	Number of words to be read or written, unit: word, 1~256	INT	Constant, IW, QW, MW, NW, SW
OFF	Offset	FLASH offset, unit: word, 1~32767	INT	Constant, IW, QW, MW, NW, SW

**5.11.23 RTC: Real-time clock**

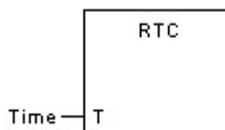
◆ **Function description**

This function block is used to set real-time clock. When this function block is called, 6 consecutive words of date and time (start in T) are written to the PLC.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL RTC (Time)

◆ **Representation in ST**

RTC (Time);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
T	Time	Time buffer, 6 words, T: year, T+1: month, T+2: day, T+3: hour, T+4: minute, T+5: second	INT	MW, NW

**5.11.24 MODRW: Modbus data read/write**

◆ **Function description**

This function block is used to read or write data by MODBUS master protocol.

Communication state:

SW21 (COM1 send state), SW23 (COM2 send state)

0: sending

1: send successfully

SW22 (COM1 receive state), SW24 (COM2 receive state)

0: receiving

1: receive successfully

2: COM 1/2 failure

3: receive timeout

4: exceed maximum interval between characters

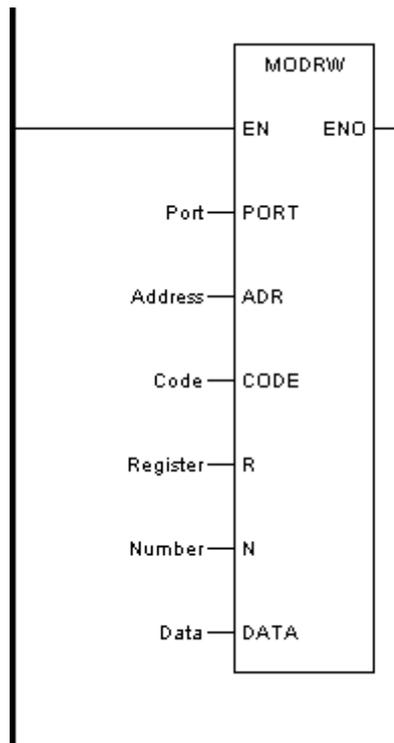
5: exceed maximum character number

7: response message error

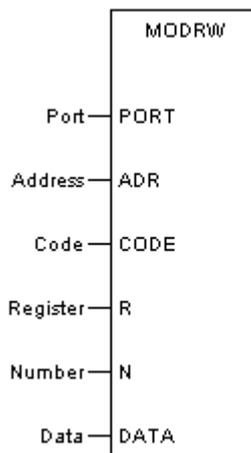
8: request message error

9: CRC16 check error

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL     MODRW (PORT:=Port, ADR:=Address, CODE:=Code, R:=Register,  
           N:=Number, DATA:=Data)

◆ **Representation in ST**

MODRW (PORT:=Port, ADR:=Address, CODE:=Code, R:=Register,  
        N:=Number, DATA:=Data);

## ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
PORT	PortNo	COM No. (1 or 2)	INT	Constant, IW, QW, MW, NW, SW
ADR	Address	Modbus slave address, 1~255	INT	Constant, IW, QW, MW, NW, SW
CODE	Code	Modbus protocol function code: 01, 02, 03, 04, 05, 06, 15, 16	INT	Constant, IW, QW, MW, NW, SW
R	Register	Start address of register	INT	Constant, IW, QW, MW, NW, SW
N	Number	Number of registers	INT	Constant, IW, QW, MW, NW, SW
DATA	Data	Data buffer	BOOL, INT	IW, QW, MW, NW, SW, I, Q, M, N, S

## 5.12 Others

This chapter describes the elementary function blocks of the Others family.

This chapter contains the following sections.

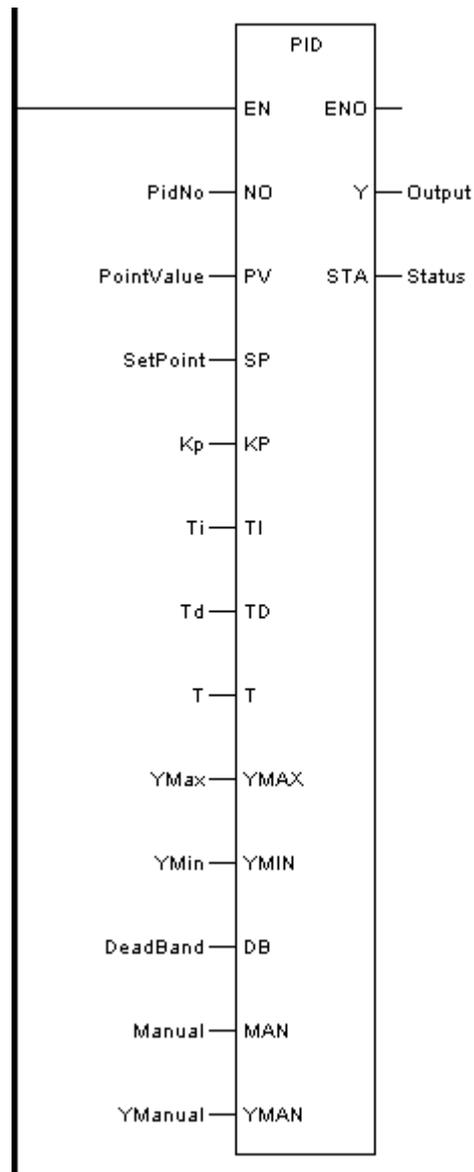
Type	Description
<b>PID</b>	PID Control
<b>LRC</b>	LRC Check
<b>CRC</b>	CRC Check
<b>LC</b>	16-Bit Linear Change
<b>DLC</b>	32-Bit Linear Change
<b>ELC</b>	Floating-Point Linear Change
<b>SORT</b>	16-Bit Data Sort
<b>DSORT</b>	32-Bit Data Sort
<b>ESORT</b>	Floating-Point Data Sort

## 5.12.1 PID: PID control

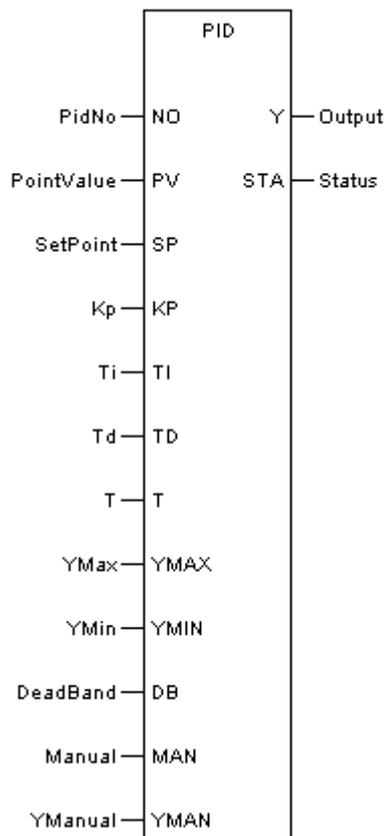
### ◆ Function description

This function block produces a PID controller.

### ◆ Representation in LD



### ◆ Representation in FBD



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
NO	PidNo	PID serial No.	INT	Constant, IW, QW, MW, NW, SW
PV	PointValue	Measure value of controlled variable	INT	IW, MW, NW
SP	SetPoint	Setpoint	INT	Constant, IW, QW, MW, NW, SW
KP	Kp	Proportional gain, unit: 0.01	INT	Constant, IW, QW, MW, NW, SW
TI	Ti	Integral time, unit: 0.01s	INT	Constant, IW, QW, MW, NW, SW
TD	Td	Derivative time, unit: 0.01s	INT	Constant, IW, QW, MW, NW, SW
T	T	Sample time, unit: 0.01s, 1~2000	INT	Constant, IW, QW, MW, NW, SW
YMAX	YMax	Upper limit of output	INT	Constant, IW, QW, MW, NW, SW
YMIN	YMin	Lower limit of output	INT	Constant, IW, QW, MW, NW, SW
DB	DeadBand	Dead band	INT	Constant, IW, QW, MW, NW, SW

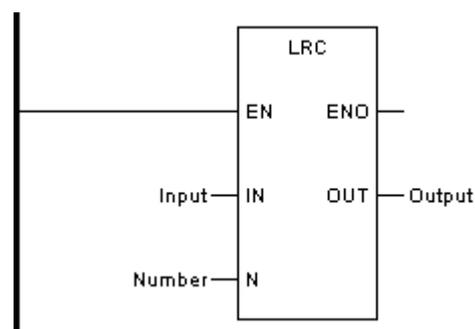
MAN	Manual	Manual/automatic mode 1: Manual, 0: Automatic	BOOL	Constant, I, Q, M, N, S
YMAN	YManual	Manual output value	INT	Constant, IW, QW, MW, NW, SW
Y	Output	Output	INT	MW, NW
STA	Status	State: 1: Initialization error 2: PID ON 3: PID OFF 4: Manual 5: Output > YMAX 6: Output < YMIN 7: YMAX < YMIN	INT	MW, NW

### 5.12.2 LRC: LRC check

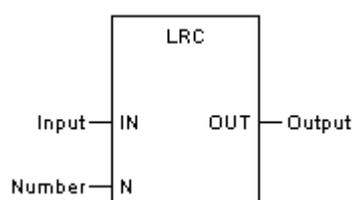
#### ◆ Function description

This function block completes LRC check.

#### ◆ Representation in LD



#### ◆ Representation in FBD



◆ **Representation in IL**

CAL LRC (IN:=Input, N:=Number, OUT=>Output)

◆ **Representation in ST**

LRC (IN:=Input, N:=Number, OUT=>Output);

◆ **Parameter description**

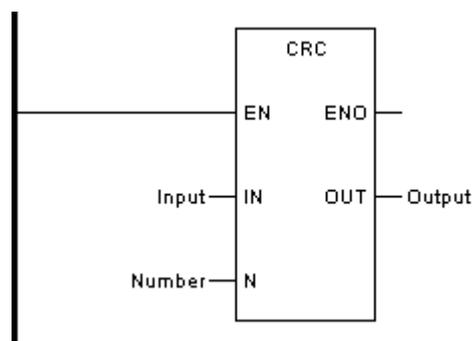
Icon	Parameter	Description	Data type	Point type
IN	Input	Input data	INT	MW, NW
N	Number	Number of bytes	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	LRC output	INT	MW, NW

### 5.12.3 CRC: CRC check

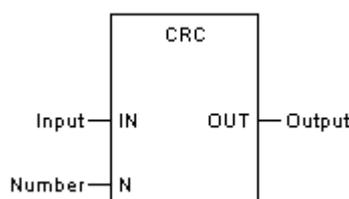
◆ **Function description**

This function block completes CRC check.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL CRC (IN:=Input, N:=Number, OUT=>Output)

◆ **Representation in ST**

CRC (IN:=Input, N:=Number, OUT=>Output);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input data	INT	MW, NW
N	Number	Number of bytes	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	CRC output	INT	MW, NW

### 5.12.4 LC: 16-bit linear change

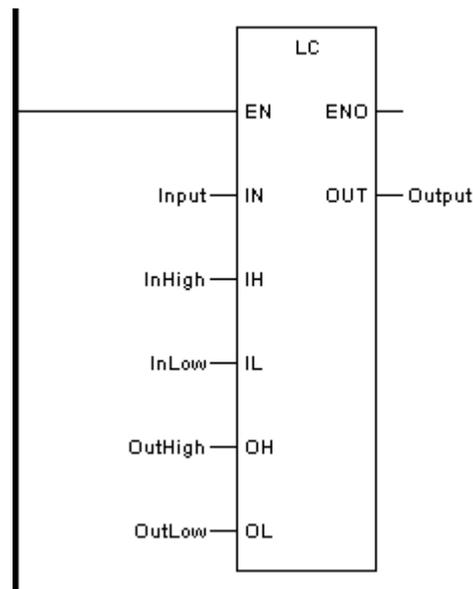
◆ **Function description**

This function block makes linear change according to upper limit and lower limit.

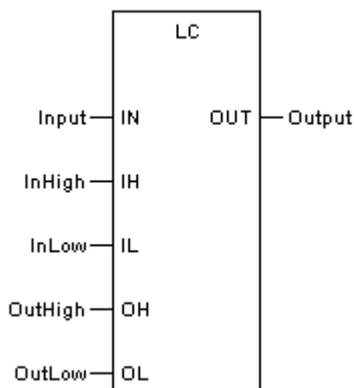
◆ **Formula**

$$OUT = ((IN - IL) * (OH - OL)) / (IH - IL) + OL$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL LC (IN:=Input, IH:=InHigh, IL:=InLow, OH:=OutHigh, OL:=OutLow, OUT=>Output)

◆ **Representation in ST**

LC (IN:=Input, IH:=InHigh, IL:=InLow, OH:=OutHigh, OL:=OutLow, OUT=>Output);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input data	INT	Constant, IW, QW, MW, NW, SW
IH	InHigh	Upper limit of input	INT	Constant, IW, QW, MW, NW, SW
IL	InLow	Lower limit of input	INT	Constant, IW, QW, MW, NW, SW
OH	OutHigh	Upper limit of output	INT	Constant, IW, QW, MW, NW, SW
OL	OutLow	Lower limit of output	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output data	INT	MW, NW

**5.12.5 DLC: 32-bit linear change**

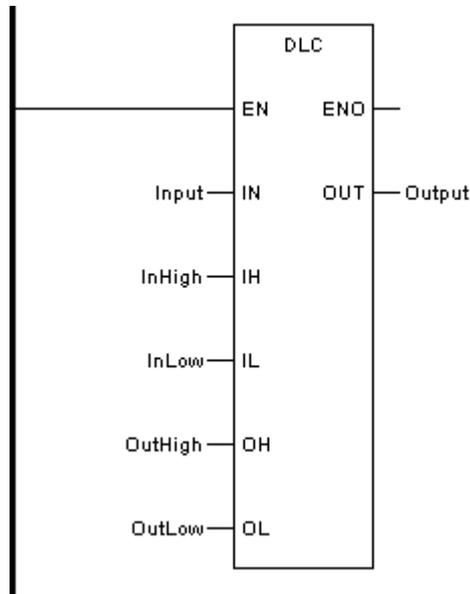
◆ **Function description**

This function block makes 32-bit linear change according to upper limit and lower limit. Input and output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

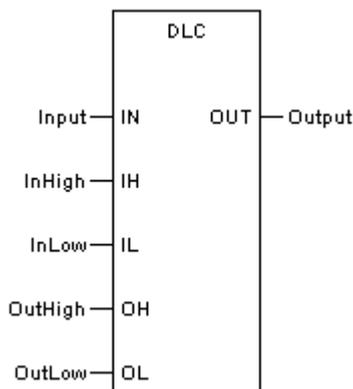
◆ **Formula**

$$OUT = ((IN - IL) * (OH - OL)) / (IH - IL) + OL$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input data	DINT	Constant, MW, NW
IH	InHigh	Upper limit of input	DINT	Constant, MW, NW
IL	InLow	Lower limit of input	DINT	Constant, MW, NW
OH	OutHigh	Upper limit of output	DINT	Constant, MW, NW

OL	OutLow	Lower limit of output	DINT	Constant, MW, NW
OUT	Output	Output data	DINT	MW, NW

### 5.12.6 ELC: Floating-point linear change

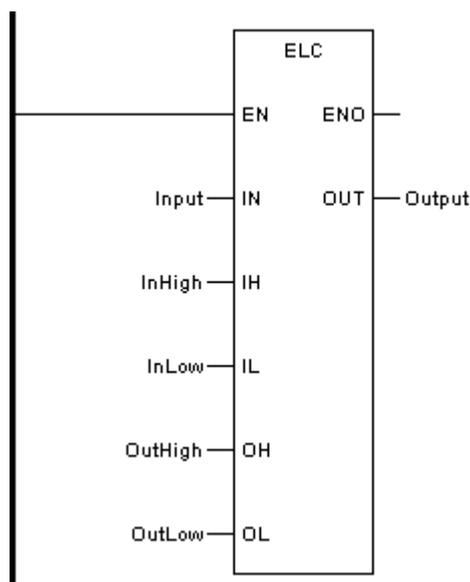
◆ **Function description**

This function block makes floating-point linear change according to upper limit and lower limit. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

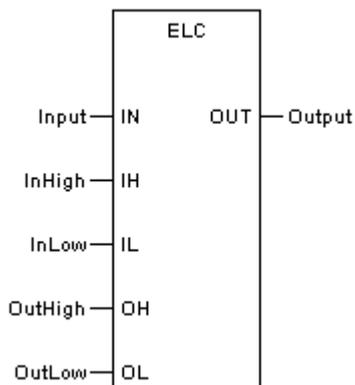
◆ **Formula**

$$OUT = ((IN - IL) * (OH - OL)) / (IH - IL) + OL$$

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Parameter description**

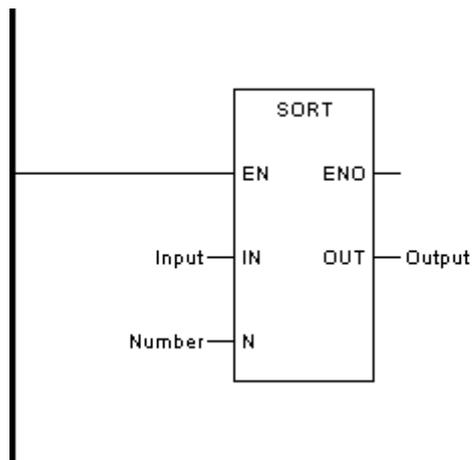
Icon	Parameter	Description	Data type	Point type
IN	Input	Input data	REAL	Constant, MW, NW
IH	InHigh	Upper limit of input	REAL	Constant, MW, NW
IL	InLow	Lower limit of input	REAL	Constant, MW, NW
OH	OutHigh	Upper limit of output	REAL	Constant, MW, NW
OL	OutLow	Lower limit of output	REAL	Constant, MW, NW
OUT	Output	Output data	REAL	MW, NW

**5.12.7 SORT: 16-bit data sort**

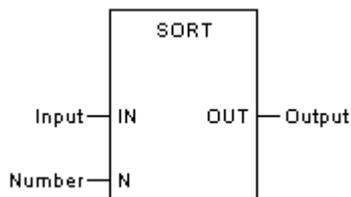
◆ **Function description**

This function block sorts the input data according to the ascending order.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL SORT (IN:=Input, N:=Number, OUT=>Output)

◆ **Representation in ST**

SORT (IN:=Input, N:=Number, OUT=>Output);

◆ **Parameter description**

Icon	Parameter	Description	Data type	Point type
IN	Input	Input data	INT	MW, NW
N	Number	Number of input data to be sorted	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output data	INT	MW, NW

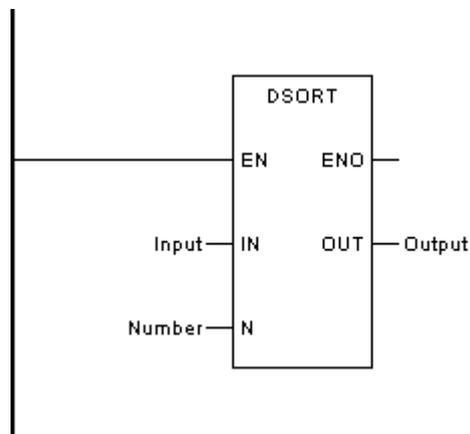
**5.12.8 DSORT: 32-bit data sort**

◆ **Function description**

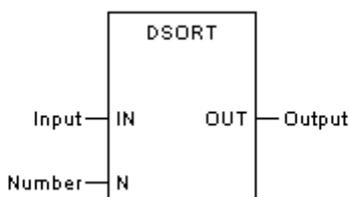
This function block sorts the input data according to the ascending order. Input and

output can only be 32-bit register, occupying 2 continuous word registers. The serial number of word register must be odd.

◆ **Representation in LD**



◆ **Representation in FBD**



◆ **Representation in IL**

CAL DSORT (IN:=Input, N:=Number, OUT=>Output)

◆ **Representation in ST**

DSORT (IN:=Input, N:=Number, OUT=>Output);

◆ **Parameter description**

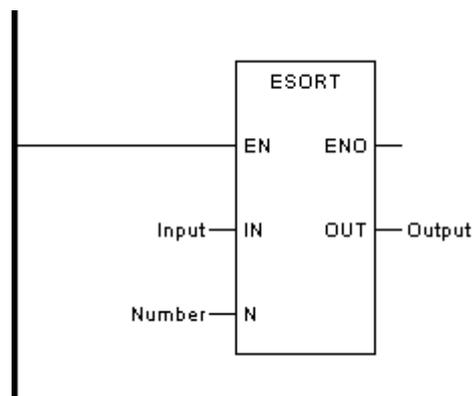
Icon	Parameter	Description	Data type	Point type
IN	Input	Input data	DINT	MW, NW
N	Number	Number of input data to be sorted	INT	Constant, IW, QW, MW, NW, SW
OUT	Output	Output data	DINT	MW, NW

## 5.12.9 ESORT: Floating-point data sort

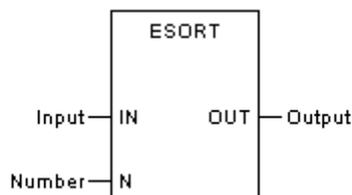
### ◆ Function description

This function block sorts the input data according to the ascending order. Input and output can only be floating-point register, occupying 2 continuous word registers. The serial number of word register must be odd.

### ◆ Representation in LD



### ◆ Representation in FBD



### ◆ Representation in IL

```
CAL    ESORT (IN:=Input, N:=Number, OUT=>Output)
```

### ◆ Representation in ST

```
ESORT (IN:=Input, N:=Number, OUT=>Output);
```

### ◆ Parameter description

Icon	Parameter	Description	Data type	Point type
IN	Input	Input data	REAL	MW, NW
N	Number	Number of input data to	INT	Constant, IW, QW, MW, NW, SW

		be sorted		
OUT	Output	Output data	REAL	MW, NW

## Chapter 6 LD Programming

Ladder diagram (LD) is one kind of graphic programming language, whose instruction and syntax are similar with the circuit diagram. By using LD, the data and current flow can be tracked online.

LD programming language was first designed to suit the habit of traditional circuit designers, so there are many points of similarity between LD and the hardware circuit composed of relays and electronic devices, and the control programs written by LD are very intuitive. After a period of using, users have found that the LD programming language is easy to learn and convenient to use, so the LD language was widely applied. With the continuous development and expanding some complex computing functions such as timer, counter, and math computing, the LD programming language has become one of the main programming languages in PLC and other automation control devices.

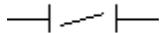
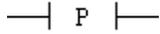
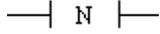
Many instructions and function blocks of the LD programming language are very similar to the control devices of local circuit. For example, normally open contact (-| |-) in LD is equivalent to a normally open contact in site, and normally open coil (-( )-) in LD is equivalent to a relay coil in site.

The LD programming language has many types of instructions and function blocks, rich Data types and the convenient addressing modes required by users, so the acquaintance with the function blocks and programming modes is the precondition for writing concise and efficient LD programs.

### 6.1 Contact, coil and function block

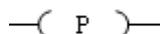
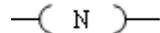
#### 6.1.1 Contact

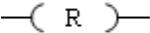
Contact is used to monitor the state of a given point. The data of the given point can only be BOOL type, only 0 or 1. When the contact ON condition is met, the current can be conducted through the contact. The contact will turn red when reflected on the LD at online state, otherwise the contact turns green. The contact ON condition depends on the state of given point and the contact type.

Contact Type	Description	Icon
Normally open contact	If the given point of the normally open contact is 1, the current can be conducted to the right.	
Normally closed contact	If the given point of the normally closed contact is 0, the current can be conducted to the right.	
Positive transition-sensing contact	If the given point of positive transition-sensing contact has a bit change from 0 to 1, the current can be conducted to the right and maintain one cycle. When the contact is scanned in the next cycle, the current cannot be conducted until the bit change from 0 to 1 appears again.	
Negative transition-sensing contact	If the given point of negative transition-sensing contact has a bit change from 1 to 0, the current can be conducted to the right and maintain one cycle. When the contact is scanned in the next cycle, the current cannot be conducted until the bit change from 1 to 0 appears again.	

## 6.1.2 Coil

Coil is used to control a given point which is also BOOL Data type. The coil is controlled by some conditional logic. Only when the left condition of coil is met, and the current is conducted, then the coil has effect. There are also many types of coil.

Coil	Description	Icon
Coil	When the current of coil is conducted, the given point is set to 1. Coil is non-retentive, so the coil will be set to 0 when the current is not conducted.	
Negated coil	When the current of negated coil is not conducted, the given point is set to 1; when the current is conducted, the given point is set to 0. Negated coil is also non-retentive.	
Positive transition-sensing coil	When the current input of positive transition-sensing coil has a bit change from 0 to 1, the given point of coil is set to 1, and maintains a scan cycle. In the next scan to this part of LD, the given point is set back to 0.	
Negative transition-sensing coil	When the current input of negative transition-sensing coil has a bit change from 1 to 0, the given point of coil	

coil	is set to 1, and maintains a scan cycle. In the next scan to this part of LD, the given point is set back to 0.	
Set coil	When the current of set coil is conducted, the given point is set to 1. The set coil is retentive, only encountering the reset coil of the same point, and the point will be reset from 1 to 0. Otherwise, the given point will maintain 1 whether or not the current is conducted.	
Reset coil	When the current of reset coil is conducted, the given point is set to 0. The reset coil is retentive, only encountering the set coil of the same point, and the point will be reset from 0 to 1. Otherwise, the given point will maintain 0 whether or not the current is conducted.	

### 6.1.3 Function block

According to the different functions, the basic function blocks can be divided into mathematics, statistics, logic, comparison, conversion, data move, timer, counter, control, PLC and other groups. First select the group in the toolbar as shown in Fig.6.1, and then select the specific function block as shown in Fig.6.2. All the basic functions and parameters of function blocks are described in Chapter 5 "Basic Function Block".

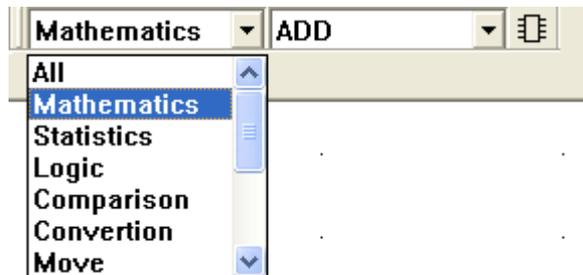


Fig.6.1 LD group selection

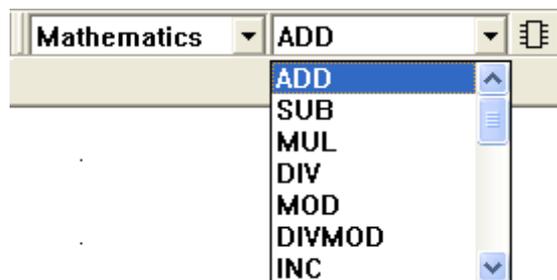


Fig.6.2 LD function block selection

## 6.2 Operation

By the left mouse button, double click the LD name in the project browser, so the LD program content is displayed in the right side editing area and can be edited.

In the LD editor, the background of window is logical grid, draw the left power rail on the left and the right power rail on the right. By using LD programming, only when connected to the left power rail, the LD objects (contacts, coils, function blocks) can be processed during LD programming. The outputs of internal coils and function blocks can be connected to the right power rail or not, but it is usually deemed to be connected and establish the current circuit.

Different function blocks occupy different amount of grids. For example, the smallest function blocks such as contact and coil only respectively occupy one grid. Page can be flipped by using **PgUp** or **PgDn** key, and also by using mouse, you can drag the scroll bar on the right side of the editing area to flip to any page desired quickly.

When the function block needs to be placed in editing area, just choose the function block in LD menu or in LD toolbar and click the editing area by the left mouse button, then the function block can be placed in the position clicked by the mouse; When the function block needs to be moved, just choose that function block and hold the mouse button to the specific position. Cut, copy, paste, and delete operations of function block can be achieved through the menu operations. The operation of multiple function blocks is the same to that of a single function block, but multiple function blocks should be selected firstly by block operation.

If double click a contact or coil, an enter box will appear above the contact or coil, in which the required parameter can be entered. For the parameters and pins of function blocks, they can be also entered into the pop up parameter enter box by double click. Just as shown in Fig.6.3.

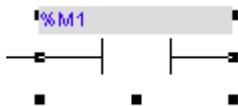


Fig.6.3 Enter parameter

When entering parameter, the form that “type + number” can be used, such as %I0001 (it can be abbreviated as %I1) and %MW0001, etc. The defined name in point table also can be used.

## 6.2.1 Link

### ◆ : Link

The **Link** operation between the input and output pins of the contact, coil and function block is equivalent to the menu operation **【LD】 / 【Link】** .

#### “Magnet” function

When two function blocks are near to a certain distance, a connection relationship will be automatically established and the link is added. This means that the connection relationship has been established between both function blocks. When any function block of both is moved, the connection will automatically calculate and the relationship between both will remain. If one is deleted, all links connected with this function block will be automatically deleted.

#### Manual Link

Select the icon  and move the mouse to the start position of link, so the mouse

pointer will become the icon  , then click the mouse to select the start of link.

Move the mouse to the end position of link, the mouse pointer will become the

icon  , then click the mouse to complete the link operation.

## 6.2.2 Negate

### ◆ : Negate

**Negate** operation is to cyclically switch the contacts and coils and is equivalent to the menu operation **【LD】 / 【Negate】** .

#### Switch among contacts

Select the contact need to be switched, and click the **Negate** icon  each time, the contact will cyclically switch among the following four types of contacts.

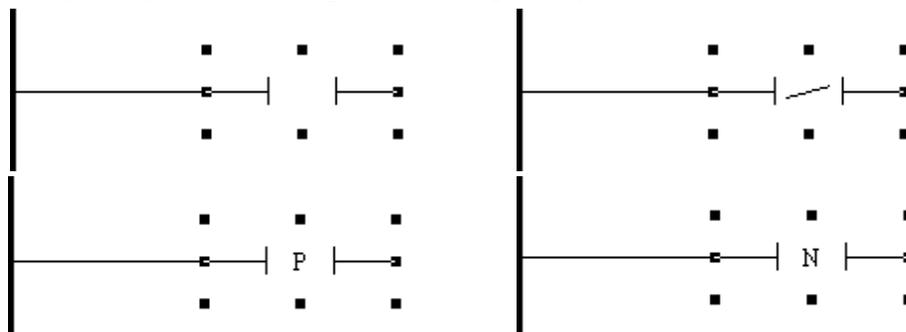


Fig.6.4 Contact negate

### Switch among coils

Select the coil need to be switched, and click the **Negate** icon  each time, the coil will cyclically switch among the following six types of coils.

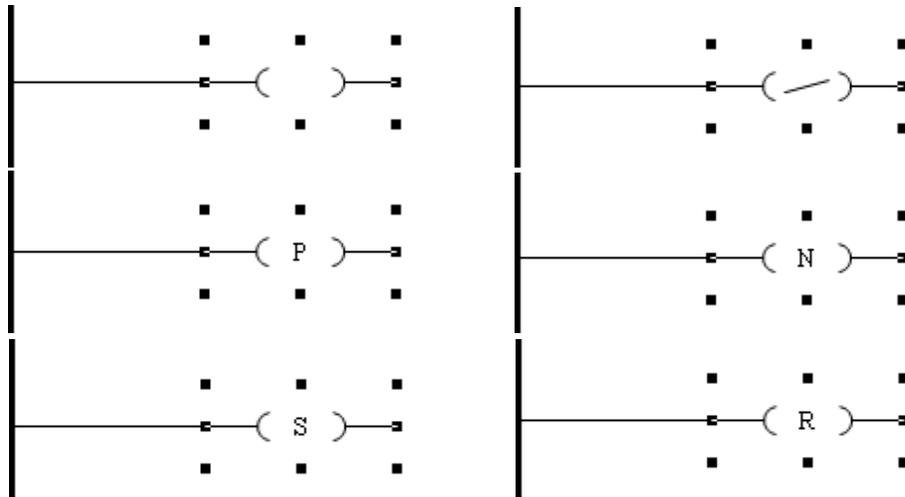


Fig.6.5 Coil negate

### 6.2.3 Label

#### ◆ :Label

**Label** is the sign for **Goto. Label** operation is equivalent to the menu operation **【LD】 / 【Label】**.

#### Placement of label

Select the **Label** icon  and click the mouse in LD edit window, so the label is placed. **The label should occupy one row in LD**, as shown in the following figure.

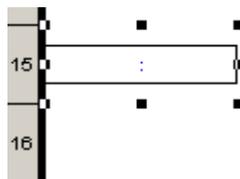


Fig.6.6 Label in LD

#### Add label name

Double click the label, as shown in the following figure. The label name can be input into the gray part, such as "LABLE".

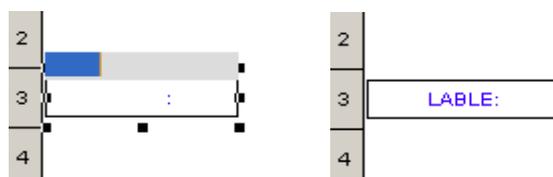


Fig.6.7 Label name input in LD

◆ :Goto

**Goto** operation is equivalent to the menu operation **【LD】 / 【Goto】** .

**Placement of Goto**

Select the **Goto** icon  and click the mouse in LD edit window, so the **Goto** icon is placed. The icon is shown in the following figure.

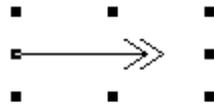


Fig.6.8 Goto in LD

**Add Goto name**

Double click the **Goto** icon, as shown in the following figure; the label name can be entered into the gray part, such as "LABEL". This means that when a bit change of %I0001 from 0 to 1 appears, the program will jump to the position of label "LABEL".



Fig.6.9 Goto name input in LD

## 6.2.4 Return

◆ :Return

**Return** operation is equivalent to the menu operation **【LD】 / 【Return】** .

When LD performs the **Return** operation, the program returns to the part which calls this subprogram and continues to the next. All the programs following this **Return** operation will no longer be scanned and performed. Icon is shown as below:



Fig.6.10 Return in LD

## 6.2.5 Comment

### ◆ :Comment

**Comment** operation is equivalent to the menu operation **【LD】 / 【Comment】** .

The text annotation can be inputted in anywhere in the LD editor.

## 6.2.6 Zoom

### ◆ :Zoom

**Zoom** operation is equivalent to the menu operation **【LD】 / 【Zoom】** .

This operation changes the size of LD editor.

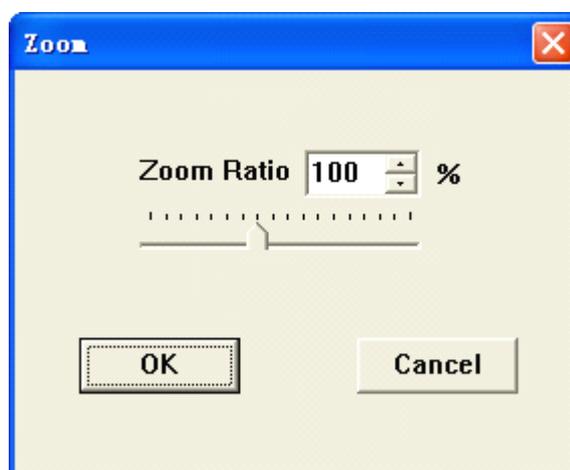


Fig.6.11 Zoom in LD

## 6.2.7 Insert

### ◆ : Insert

**Insert** operation is equivalent to the menu operation **【LD】 / 【Insert】** .

Row is the unit of LD editor. If want to insert some new LD function blocks between two rows, the **Insert** operation can be used to enlarge the editor, leaving space to add new program section.

※ **There shall be no other function block existed in the position to insert one row.**

## 6.2.8 Remove

### ◆ : Remove

**Remove** operation is equivalent to the menu operation **【LD】 / 【Remove】** .

If there are too many blank rows between two consecutive function blocks in LD editor, the **Remove** operation can be used to delete blank rows for beautiful looking.

※ **There shall be no other function block existed in the position to delete one row.**

# Chapter 7 FBD Programming

## 7.1 Using FBD programming language

### 7.1.1 Properties of FBD program

- ◆ There is a grid behind FBD program. Grid cell is the smallest possible space between two objects in FBD editor.
- ◆ FBD programming language is not grid cell oriented, but the object is still aligned with the grid cell.
- ◆ Execution order depends on the position of FBD in section (perform from left to right, from top to bottom). If FBD is connected to the network of graphic link, the execution order depends on the signal flow.

### 7.1.2 New FBD program

Select **【File】 / 【New program】** and the window will pop up as shown in Fig.7.1. After selecting program type **FBD**, the new program name can be inputted into “Program Name” and the comment can be inputted into “Program Description”.

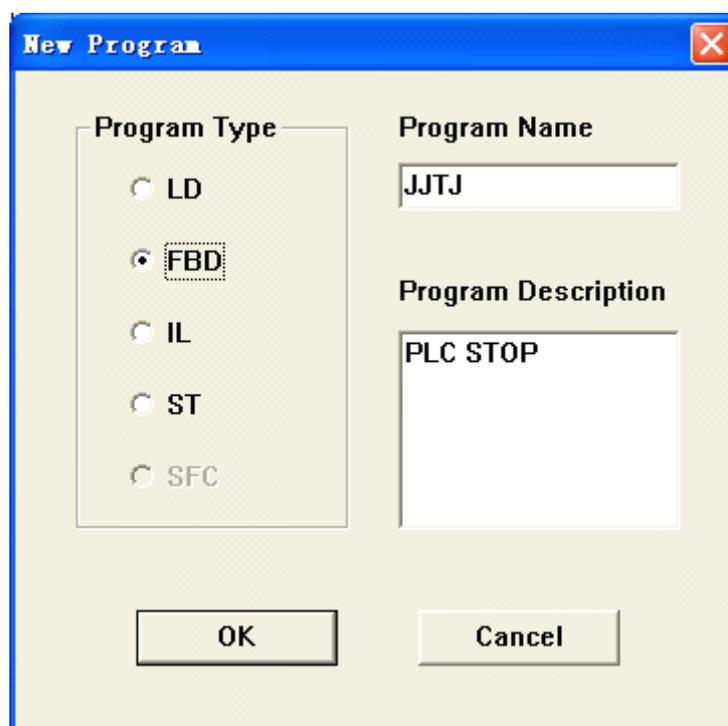


Fig.7.1 New FBD program

## 7.2 Edit FBD program

### 7.2.1 Function block

According to the different functions, the basic function blocks can be divided into mathematics, statistics, logic, comparison, conversion, data move, timer, counter, control, PLC and other groups. First select the group in the toolbar as shown in Fig.7.2, and then select the specific function block as shown in Fig.7.3. All the basic functions and parameters of function blocks are described in Chapter 5 "Basic function block".

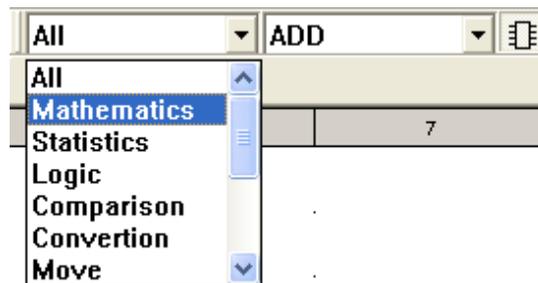


Fig.7.2 FBD group selection

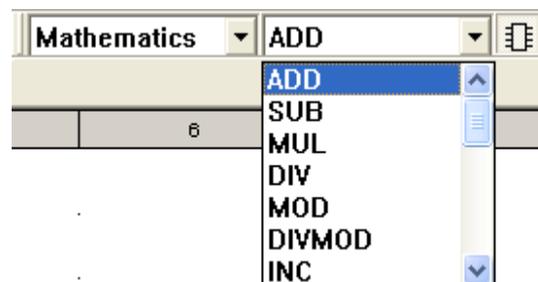


Fig.7.3 FBD function block selection

### 7.2.2 Property modification of function block

The properties of each function block can be modified, for example, display or hide **EN/ENO** in FBD, some function blocks can increase the number of input parameters. As shown in Fig.7.4, take the **AND** function block as example, first select the function block then click the right mouse button, so the property modification dialog box will pop up as shown in Fig.7.5, and the parameters can be modified here.

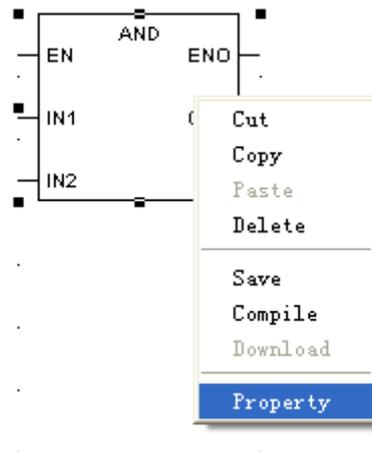


Fig.7.4 FBD property

**FB Property**
✕

**AND AND**

<p><b>Program Name</b> <input style="width: 100%;" type="text" value="TEST1"/></p> <p><b>Start Position</b> <input style="width: 100%;" type="text" value="{4,3}"/></p> <p><b>Input Number</b> <input style="width: 100%;" type="text" value="4"/></p>	<p><b>Execution Order</b> <input style="width: 100%;" type="text"/></p> <p><input checked="" type="checkbox"/> <b>Display EN/ENO</b></p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------

<p><b>Input 1</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 2</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 3</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 4</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 5</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 6</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 7</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 8</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 9</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 10</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 11</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 12</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 13</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 14</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 15</b> <input style="width: 100%;" type="text"/></p> <p><b>Input 16</b> <input style="width: 100%;" type="text"/></p>	<p><b>Output 1</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 2</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 3</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 4</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 5</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 6</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 7</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 8</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 9</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 10</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 11</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 12</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 13</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 14</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 15</b> <input style="width: 100%;" type="text"/></p> <p><b>Output 16</b> <input style="width: 100%;" type="text"/></p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fig.7.5 Properties of function block

## 7.3 Operation

### 7.3.1 Link

#### ◆ :Link

The **Link** operation connects the input and output pints of function block, and is equivalent to the menu operation **【FBD】 / 【Link】** .

#### “Magnet “ function

When two function blocks are near to a certain distance, a connection relationship will be automatically established and the link is added. This means that the connection relationship has been established between both function blocks. When any function block of both is moved, the connection will automatically calculate and the relationship between both will remain. If one is deleted, all links connected with this function block will be automatically deleted.

#### Manual Link

Select the icon  and move the mouse to the start position of link, so the mouse

pointer will become the icon  , then click the mouse to select the start of link.

Move the mouse to the end position of link, the mouse pointer will become the

icon  , then click the mouse to complete the link operation.

### 7.3.2 Negate

#### ◆ :Negate

**Negate** operation is to negate the input and output, and is equivalent to the menu operation **【FBD】 / 【Negate】** .

As shown in Fig.7.6, input pin IN1, IN2 of **AND** function block can be negated to 0 as a valid input, but 1 is still the valid input for the input pin IN3.

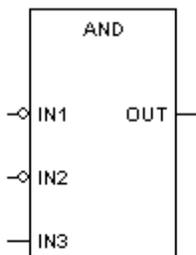


Fig.7.6 Negate operation

### 7.3.3 Label

◆ **L:**:Label

**Label** is the sign for **Goto. Label** operation is equivalent to the menu operation **【FBD】 / 【Label】** .

**Placement of Label**

Select the **Label** icon **L:** and click the mouse in FBD edit window, so the label is placed. **The label should occupy one row in FBD**, as shown in the following figure.

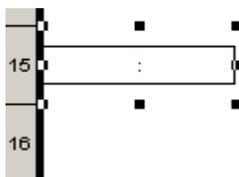


Fig.7.7 Label in FBD

**Add label name**

Double click the label and the label name can be entered into the gray part, as shown in the following figure, such as" LABLE".

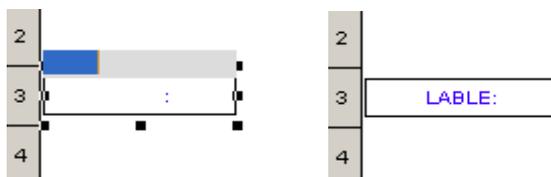


Fig.7.8 Label name input in FBD

◆ **⇒**:Goto

**Goto** operation is equivalent to the menu operation **【FBD】 / 【Goto】** .

**Placement of Icon**

Select the **Goto** icon **⇒** and click the mouse in FBD edit window, so the **Goto** icon is placed. The icon is shown in the following figure.

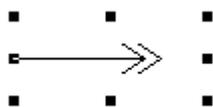


Fig.7.9 Goto in FBD

**Add Goto name**

Double click the **Goto** icon, and the label name can be entered into the gray part, as

shown in the following figure, such as “LABLE”.

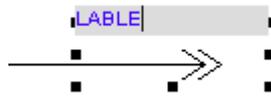


Fig.7.10 Goto name input in FBD

### 7.3.4 Return

#### ◆ :Return

**Return** operation is equivalent to the menu operation **【FBD】 / 【Return】**.

When FBD performs the **Return** operation, the program returns to the part which calls this subprogram and continues to the next. All the programs following this **Return** operation will no longer be scanned and performed. Icon is shown as below:



Fig.7.11 Return in FBD

### 7.3.5 Comment

#### ◆ : Comment

**Comment** operation is equivalent to the menu operation **【FBD】 / 【Comment】**.

The text annotation can be inputted in anywhere in the FBD editor.

### 7.3.6 Zoom

#### ◆ :Zoom

Zoom operation is equivalent to the menu operation **【FBD】 / 【Zoom】**.

This operation changes the size of FBD editor.

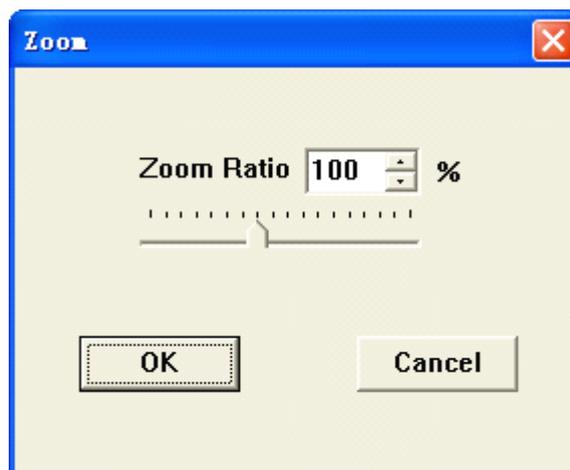


Fig.7.12 Zoom in FBD

### 7.3.7 Insert

#### ◆ :Insert

**Insert** operation is equivalent to the menu operation **【FBD】 / 【Insert】** .

Row is the unit of FBD editor. If want to insert some new FBD function blocks between two rows, the **Insert** operation can be used to enlarge the editor, leaving space to add new program section.

※ **There shall be no other function block existed in the position to insert one row.**

### 7.3.8 Remove

#### ◆ :Remove

**Remove** operation is equivalent to the menu operation **【FBD】 / 【Remove】** .

If there are too many blank rows between two consecutive function blocks in FBD editor, the **Remove** operation can be used to delete blank rows for beautiful looking.

※ **There shall be no other function block existed in the position to delete one row.**

## Chapter 8 IL Programming

With the help of IL programming language, the function blocks and functions can be conditionally or unconditionally called, assignment can be done, and the conditional or unconditional jump can also be performed within the section.

An IL program contains a series of instructions. Each instruction contains the following contents:

- An operator
- If necessary, a modifier
- If necessary, one or more operands

If more than one operand is used, commas should be used to separate these operands. A label can be used with a colon followed before an instruction. Comment can be added anywhere of IL editor.

### 8.1 The structure of programming language

IL is the so-called accumulator-oriented language. It means that each instruction will use or change the contents of current accumulator. Thus, an IL always begins with the "LD" operator (load accumulator instruction).

Examples for accumulation:

Instruction	Interpretation
LD 10	Load 10 to the accumulator.
ADD 25	Add 25 to the accumulator.
ST A	Store the result in "A". Now the accumulator and "A" value are 35. If the next instruction doesn't begin with "LD", 35 will be recognized as the accumulator value.

Similarly, accumulator is frequently used in comparison operation. The Boolean result after comparison is stored in the accumulator as the current value.

Examples for comparison:

Instruction	Interpretation
LD B	Load "B" to the accumulator.
GT 10	Compare the accumulator with 10.
ST A	Store the comparison result in "A".

	If "B" is less than or equal to 10, the "A" and the accumulator value is 0 (FALSE). If "B" is greater than 10, the "A" and the accumulator value is 1 (TRUE).
--	---------------------------------------------------------------------------------------------------------------------------------------------------------------

## 8.2 Execution order

The instructions are performed line by line and from top to bottom. This order can be changed by parentheses.

If the respective value of "A", "B", "C" and "D" are 1, 2, 3 and 4, and the instruction operates as follows:

```
LD      A
ADD     B
SUB     C
MUL     D
ST      E
```

Then the given result of "E" is 0.

If the instruction operates as follows:

```
LD      A
ADD     B
SUB (
LD      C
MUL     D
)
ST      E
```

Then the given result of "E" is -9.

## 8.3 Instruction interpretation

### 8.3.1 Operand

The operands can be constants, points, etc.

### 8.3.2 Modifier

Modifier "N" is used to negate the value of operand bit by bit.

Example for modifier "N":

In the following example, if the "A" value is 1, and the "B" value is 0, then the "C" value is 1.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
ANDN B	Perform the "AND" operation of the accumulator value and the negated "B" value.
ST C	Store the result in "C".

Modifier "C" is used to perform the relevant instructions when the value of accumulator is 1 (TURE).

Example for modifier "C":

In the following example, only when the "A" value is 1 and "B" value is 1 too, the jump to "START" is performed.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
AND B	Perform the "AND" operation of the accumulator value and the "B" value.
JMPC START	When the value of accumulator is 1, jump to the label "START" and continue to perform instructions.

Modifier "CN" is used to perform the relevant instructions when the value of accumulator is 0 (FALSE).

Example for modifier "CN":

In the following example, only when the "A" value is 0 or "B" value is 0, the jump to "START" is performed.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
AND B	Perform the "AND" operation of the accumulator value and the "B" value.
JMPCN START	When the value of accumulator is 0, jump to the label "START" and continue to perform instructions.

Modifier "(" and ")": The modifier of left parenthesis "(" is used to delay the operation till the right parenthesis ")" appears. The number of right parenthesis modifier must be equal to the number of left parenthesis modifier. The parenthesis can be nested.

Example for modifier "(" and ")":

In the following example, if the "C" or "D" is 1, and only when both "A" and "B" are 1, then the "E" is 1.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
AND B	Perform the "AND" operation of the accumulator value and the "B" value.
AND (	"AND" is delayed till the appearance of right parenthesis.
LD C	Load "C" value to the accumulator.
OR D	Perform the "OR" operation of the accumulator value and the "D" value.
)	The delayed "AND" is started. Perform the "AND" operation of the result of "A AND B" and accumulator value (the result of "C OR D").
ST E	Store the result in "E".

### 8.3.3 Operator

Operator	Operator meaning	Modifier type	Operand type
LD	Load the operand value to the accumulator.	N	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
ST	Store the accumulator value in operand.	N	Q, QW, M, MW, N, NW
S	If the accumulator value is 1, set the operand to 1.		Q, M, N
R	If the accumulator value is 1, set the operand to 0.		Q, M, N
AND	Logic AND	N, N(, (	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
OR	Logic OR	N, N(, (	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
XOR	Logic exclusive OR	N, N(, (	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
ADD	Addition	(	Constant, IW, QW, MW, NW, SW
SUB	Subtraction	(	Constant, IW, QW, MW, NW, SW
MUL	Multiplication	(	Constant, IW, QW, MW, NW, SW
DIV	Division	(	Constant, IW, QW, MW, NW,

			SW
MOD	Modulo	(	Constant, IW, QW, MW, NW, SW
GT	Greater than	(	Constant, IW, QW, MW, NW, SW
GE	Greater than or equal to	(	Constant, IW, QW, MW, NW, SW
EQ	Equal to	(	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
NE	Not equal to	(	Constant, I, Q, IW, QW, M, MW, N, NW, S, SW
LE	Less than or equal to	(	Constant, IW, QW, MW, NW, SW
LT	Less than	(	Constant, IW, QW, MW, NW, SW
JMP	Jump to label	C, CN	Label
CAL	Call	C, CN	Function block, program
RET	Return	C, CN	None

### 8.3.4 Label

Label is used to specify the jump target and must be the first element of a line. The label must be unique in the entire program and doesn't change with the conditions. The longest length of label can be 24 characters. The name of label must conform to IEC named specification. The label and the following instruction must be separated by a colon ":". The label can only appear at the beginning of the expression, otherwise an undefined value will appear in the accumulator.

### 8.3.5 Comment

In IL editor, comment always begins with character string "(" and end with character string "\*". Any character string can be inserted into these two character strings. The comment will display with different color.

The character string "//" is used to comment current line.

## 8.4 Operator

### 8.4.1 Load (LD and LDN)

“LD” is used to load the operand value to accumulator. The data bits of accumulator are adjusted automatically according to the data type of operand. This operation is also suitable for derived data type.

Example for “LD” is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
ADD B	Perform the “ADD” operation of the accumulator value and “B” value.
ST C	Store the result in “C”.

The loaded operand can be negated by modifier N.

Example for “LDN” is as follows:

Instruction	Interpretation
LDN A	Load the bitwise NOT "A" value to the accumulator.
ADD B	Perform the “ADD” operation of the accumulator value and “B” value.
ST C	Store the result in “C”.

### 8.4.2 Store (ST and STN)

“ST” is used to store the current value of accumulator in operand. So the data type of operand must be consistent with that of accumulator. Whether use original result in the following operation depends on whether there is a “LD” after the “ST” or not.

Example for “ST” is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
ADD B	Perform the “ADD” operation of the accumulator value and “B” value.
ST C	Store the result in “C”.
ADD D	Perform the “ADD” operation of the “C” value (that is, the current value of accumulator) and “D” value.
ST E	Store the result in “E”.

LD F	Now load "F" value to the accumulator.
SUB 2	Subtract 2 from the accumulator value.
ST G	Store the result in "G".

The stored operand can be negated by modifier N.

Example for "STN" is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
ADD B	Perform the "ADD" operation of the accumulator value and "B" value.
STN C	Store the bitwise NOT result in "C".

### 8.4.3 Set (S), Reset (R)

If the current value of accumulator is Boolean 1, then "S" will set the operand to 1.

Example for "S" is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
S B	If the accumulator value ("A" value) is 1, then set "B" to 1.

"R" operator and "S" operator are always used together as a pair (Trigger).

Example for "RS" trigger ("R" dominant) is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
S B	If the accumulator value ("A" value) is 1, then set "B" to 1.
LD C	Load "C" value to the accumulator.
R B	If the accumulator value ("C" value) is 1, then set "B" to 0.

If the current value of accumulator is Boolean 1, then "R" will set the operand to 0.

Example for "R" is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
R B	If the accumulator value ("A" value) is 1, then set "B" to 0.

“S” operator and “R” operator are always used together as a pair (Trigger).

Example for “SR” trigger (“S” dominant) is as follows:

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
R B	If the accumulator value (“A” value) is 1, then set “B” to 0.
LD C	Load “C” value to the accumulator.
S B	If the accumulator value (“C” value) is 1, then set “B” to 1.

## 8.4.4 Logic operation

### AND (AND, AND(), ANDN, ANDN())

By “AND” instruction, a logic “AND” operation happens between the accumulator value and operand. For the integer data type, the operation operates bit by bit.

In the following example, if “A”, “B” and “C” are all 1, then “D” is 1.

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
AND B	Perform the “AND” operation of the accumulator value and “B” value.
AND C	Perform the “AND” operation of the accumulator value (the result of “A AND B”) and “C” value.
ST D	Store the result in “D”.

“AND” can be together used with the modifier left parenthesis “(”.

In the following example, if “A” is 1, and “B” or “C” is 1, then “D” is 1.

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
AND (	“AND” is delayed till the appearance of right parenthesis.
LD B	Load “B” value to the accumulator.
OR C	Perform the “OR” operation of the “C” value and accumulator value.
)	The delayed “AND” is started. Perform the “AND” operation of the accumulator value (the result of “B OR C”) and “A” value.
ST D	Store the result in “D”.

“AND” can be together used with the modifier “N”.

In the following example, if “A” is 1, “B” and “C” is 0, then “D” is 1.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
ANDN B	Perform the "AND" operation of the negated "B" value and accumulator value.
ANDN C	Perform the "AND" operation of the negated "C" value and accumulator value (the result of "A ANDN B").
ST D	Store the result in "D".

"AND" can be together used with the modifiers "N" and left parenthesis "(".

In the following example, if "A" is 1, "B" is 0, and "C" is 1, then "D" is 1.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
ANDN (	"AND" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
ORN C	Perform the "OR" operation of the negated "C" value and accumulator value.
)	The delayed "AND" is started. Perform the "AND" operation of the "A" value and negated accumulator value (the result of "B ORN C").
ST D	Store the result in "D".

### **OR (OR, OR(), ORN, ORN())**

By "OR" instruction, a logic "OR" operation happens between the value of accumulator value and operand. For the integer data type, the operation operates bit by bit.

In the following example, if "A" or "B" is 1, and "C" is 1, then "D" is 1.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
OR B	Perform the "OR" operation of the accumulator value and "B" value.
AND C	Perform the "AND" operation of the accumulator value (the result of "A OR B") and "C" value.
ST D	Store the result in "D".

"OR" can be together used with the modifier left parenthesis "(".

In the following example, if "A" is 1, or "B" and "C" is 1, then "D" is 1.

Instruction	Interpretation
-------------	----------------

LD A	Load "A" value to the accumulator.
OR (	"OR" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
AND C	Perform the "AND" operation of the "C" value and accumulator value.
)	The delayed "OR" is started. Perform the "OR" operation of the accumulator value (the result of "B AND C") and "A" value.
ST D	Store the result in "D".

"OR" can be together used with the modifier "N".

In the following example, if "A" is 1, or "B" is 0 and "C" is 1, then "D" is 1.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
ORN B	Perform the "OR" operation of the negated "B" value and accumulator value.
AND C	Perform the "AND" operation of the "C" value and accumulator value (the result of "A ORN B").
ST D	Store the result in "D".

"OR" can be together used with the modifiers "N" and left parenthesis "(".

In the following example, if "A" is 1, or one of "B" and "C" is 0, , then "D" is 1.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
ORN (	"OR" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
AND C	Perform the "AND" operation of the "C" value and accumulator value.
)	The delayed "OR" is started. Perform the "OR" operation of the negated accumulator value (the result of "B AND C") and "A" value.
ST D	Store the result in "D".

### **Exclusive OR (XOR, XOR(), XORN, XORN())**

By "XOR" instruction, a logic "XOR" operation happens between the value of accumulator and operand. For the integer data type, the operation operates bit by bit.

In the following example, if one of "A" and "B" is 1, another is 0, then "D" is 1. If "A" and "B" are the same value (both are 0 or 1), then "D" is 0.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
XOR B	Perform the "XOR" operation of the accumulator and "B" value.
ST D	Store the result in "D".

"XOR" can be together used with the modifier left parenthesis "(".

In the following example, if one of "A" and "B AND C" is 1, another is 0, then "D" is 1. If "A" and "B AND C" are the same value (both are 0 or 1), then "D" is 0.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
XOR (	XOR is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
AND C	Perform the "AND" operation of the "C" value and accumulator value.
)	The delayed XOR is started. Perform the "XOR" operation of the accumulator value (the result of "B AND C") and "A" value.
ST D	Store the result in "D".

"XOR" can be together used with the modifier "N".

In the following example, if the value of "A" and "B" is the same (both are 0 or 1), then "C" is 1. If the value of "A" and "B" is not the same, then "C" is 0.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
XORN B	Perform the "XOR" operation of the accumulator value and negated "B" value.
ST C	Store the result in "C".

"XOR" can be together used with the modifiers "N" and left parenthesis "(".

In the following example, if the value of "A" and "B AND C" is the same (both are 0 or 1), then "D" is 1. If the value of "A" and "B AND C" is not the same, then "D" is 0.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
XORN (	"XOR" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.

AND C	Perform the “AND” operation of the “C” value and accumulator value.
)	The delayed “XOR” is started. Perform the “XOR” operation of the negated accumulator value (the result of “B AND C”) and “A” value.
ST D	Store the result in “D”.

## 8.4.5 Mathematics operation

### Addition (ADD & ADD())

By “ADD” instruction, the operand value is added to the accumulator value.

In the following example, a formula:  $D=A+B+C$  is followed.

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
ADD B	Add “B” value to the accumulator value.
ADD C	Add “C” value to the value of accumulator value (“A + B”).
ST D	Store the result in “D”.

ADD can be together used with the modifier left parenthesis “(”.

In the following example, a formula:  $D=A+(B-C)$  is followed.

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
ADD (	“ADD” is delayed till the appearance of right parenthesis.
LD B	Load “B” value to the accumulator.
SUB C	Subtract “C” value from the accumulator value.
)	The delayed “ADD” is started. Add the value of accumulator value (“B - C”) to “A” value.
ST D	Store the result in “D”.

### Subtraction (SUB 和 SUB())

By “SUB” instruction, the operand value is subtracted from the accumulator value.

In the following example, a formula:  $D=A-B-C$  is followed.

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
SUB B	Subtract “B” value from the accumulator value.

SUB C	Subtract "C" value from the accumulator value ("A - B").
ST D	Store the result in "D".

"SUB" can be together used with the modifier left parenthesis "(".

In the following example, a formula:  $D=A-(B-C)$  is followed.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
SUB (	"SUB" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
SUB C	Subtract "C" value from the accumulator value.
)	The delayed "SUB" is started. Subtract the accumulator value ("B - C") from "A" value.
ST D	Store the result in "D".

### **Multiplication (MUL and MUL())**

By "MUL" instruction, the accumulator value is multiplied by the operand value.

In the following example, a formula:  $D=A*B*C$  is followed.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
MUL B	Multiply the accumulator value by "B" value.
MUL C	Multiply the accumulator value ("A * B") by "C" value.
ST D	Store the result in "D".

"MUL" can be together used with the modifier left parenthesis "(".

In the following example, a formula:  $D=A*(B-C)$  is followed.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
MUL (	"MUL" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
SUB C	Subtract "C" value from the accumulator value.
)	The delayed "MUL" is started. Multiply the value of accumulator value ("B - C") by "A" value.
ST D	Store the result in "D".

**Division (DIV 和 DIV())**

By “DIV” instruction, the accumulator value is divided by the operand value.

In the following example, a formula:  $D=A/B/C$  is followed.

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
DIV B	Divide the accumulator value by “B” value.
DIV C	Divide the accumulator value (“A / B”) by “C” value.
ST D	Store the result in “D”.

“DIV” can be together used with the modifier left parenthesis “(”.

In the following example, a formula:  $D=A/(B-C)$  is followed.

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
DIV (	“DIV” is delayed till the appearance of right parenthesis.
LD B	Load “B” value to the accumulator.
SUB C	Subtract “C” value from the accumulator value.
)	The delayed “DIV” is started. Divide the “A” value by the accumulator value (“B - C”).
ST D	Store the result in “D”.

**Modulo (MOD 和 MOD())**

By “MOD” instruction, the result is the remainder of division that the accumulator value is divided by the operand.

In the following example, a formula:  $C=A-(A/B)*B$  is followed, in which  $(A/B)$  is the integer part of the division.

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
MOD B	The result is the remainder of division that the accumulator value is divided by “B” value.
ST C	Store the result in “C”.

“MOD” can be together used with the modifier left parenthesis “(”.

In the following example, a formula:  $D=A-(A/(B-C))*(B-C)$  is followed, in which  $(A/(B-C))$  is

the integer part of the division.

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
MOD (	"MOD" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
SUB C	Subtract "C" value from the accumulator value.
)	The delayed "MOD" is started. The result is the remainder of division that the "A" value is divided by the accumulator value ("B - C").
ST D	Store the result in "D".

## 8.4.6 Comparison operation

### Greater than (GT & GT())

By "GT" instruction, the accumulator value and operand value are compared. If the accumulator value is greater than the operand value, then the result is Boolean 1. If the accumulator value is less than or equal to the operand value, then the result is Boolean 0.

Example for "GT" is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
GT 10	Compare the accumulator value with 10.
ST B	If "A" value is greater than 10, then store 1 in "B". If "A" value is less than or equal to 10, then store 0 in "B".

"GT" can be together used with the modifier left parenthesis "(".

Example for "GT()" is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
GT (	"GT" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
SUB C	Subtract "C" value from the accumulator value.
)	The delayed "GT" is started. Compare the "A" value with the accumulator value ("B - C").
ST D	If "A" value is greater than ("B - C"), then store 1 in "D". If "A" value is less than or equal to ("B - C"), then store 0 in "D".

**Greater than or equal to (GE 和 GE())**

By “GE” instruction, the accumulator value and operand value are compared. If the accumulator value is greater than or equal to the operand value, then the result is Boolean 1. If the accumulator value is less than the operand value, then the result is Boolean 0.

Example for “GE” is as follows:

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
GE 10	Compare the accumulator value with 10.
ST B	If “A” value is greater than or equal to 10, then store 1 in “B”. If “A” value is less than 10, then store 0 in “B”.

“GE” can be together used with the modifier left parenthesis “(”.

Example for “GE( )” is as follows:

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
GE (	“GE” is delayed till the appearance of right parenthesis.
LD B	Load “B” value to the accumulator.
SUB C	Subtract “C” value from the accumulator value.
)	The delayed “GE” is started. Compare “A” value with the accumulator value (“B - C”).
ST D	If “A” value is less than (“B - C”), then store 0 in “D”. If “A” value is greater than or equal to (“B - C”), then store 1 in “D”.

**Equal to (EQ 和 EQ())**

By “EQ” instruction, the accumulator value and operand value are compared. If the accumulator value is equal to the operand value, then the result is Boolean 1. If the accumulator value is not equal to the operand value, then the result is Boolean 0.

Example for “EQ” is as follows:

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
EQ 10	Compare the accumulator value with 10.
ST B	If “A” value is equal to 10, then store 1 in “B”. If “A” value is not equal to 10, then store 0 in “B”.

“EQ” can be together used with the modifier left parenthesis “(”.

Example for “EQ( )” is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
EQ (	"EQ" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
SUB C	Subtract "C" value from the accumulator value.
)	The delayed "EQ" is started. Compare "A" value with the accumulator value ("B - C").
ST D	If "A" value is not equal to ("B - C"), then store 0 in "D". If "A" value is equal to ("B - C"), and then store 1 in "D".

### Not equal to (NE 和 NE())

By "NE" instruction, the accumulator value and operand value are compared. If the accumulator value is not equal to the operand value, then the result is Boolean 1. If the accumulator value is equal to the operand value, then the result is Boolean 0.

Example for "NE" is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
NE 10	Compare the accumulator value with 10.
ST B	If "A" value is equal to 10, store 0 in "B". If "A" value is not equal to 10, store 1 in "B".

"NE" can be together used with the modifier left parenthesis "(".

Example for "NE()" is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
NE (	"NE" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
SUB C	Subtract "C" value from the accumulator value.
)	The delayed "NE" is started. Compare "A" value with the accumulator value ("B - C").
ST D	If "A" value is not equal to ("B - C"), then store 1 in "D". If "A" value is equal to ("B - C"), and then store 0 in "D".

### Less than or equal to (LE 和 LE())

By "LE" instruction, the accumulator value and operand value are compared. If the accumulator value is less than or equal to the operand value, then the result is Boolean 1.

If the accumulator value is greater than the operand value, then the result is Boolean 0.

Example for "LE" is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
LE 10	Compare the accumulator value with 10.
ST B	If "A" value is greater than 10, then store 0 in "B". If "A" value is less than or equal to 10, then store 1 in "B".

"LE" can be together used with the modifier left parenthesis "(".

Example for "LE()" is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
LE (	"LE" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
SUB C	Subtract "C" value from the accumulator value.
)	The delayed "LE" is started. Compare "A" value with the accumulator value ("B - C").
ST D	If "A" value is greater than ("B"-"C"), then store 0 in "D". If "A" value is less than or equal to ("B - C"), then store 1 in "D".

### Less than (LT 和 LT())

By "LT" instruction, the accumulator value and operand value are compared. If the accumulator value is less than the operand value, then the result is Boolean 1. If the accumulator value is greater than or equal to the operand value, then the result is Boolean 0.

Example for "LT" is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
LT 10	Compare the accumulator value with 10.
ST B	If "A" value is less than 10, then store 1 in "B". If "A" value is greater than or equal to 10, then store 0 in "B".

"LT" can be together used with the modifier left parenthesis "(".

Example for "LT()" is as follows:

Instruction	Interpretation
-------------	----------------

LD A	Load "A" value to the accumulator.
LT (	"LT" is delayed till the appearance of right parenthesis.
LD B	Load "B" value to the accumulator.
SUB C	Subtract "C" value from the accumulator value.
)	The delayed "LT" is started. Compare "A" value with the accumulator value ("B - C").
ST D	If "A" value is less than ("B - C"), then store 1 in "D". If "A" value is greater than or equal to ("B - C"), then store 0 in "D".

### 8.4.7 Jump (JMP, JMPC and JMPCN)

#### JMP

By "JMP" instruction, an unconditional jump to the label can be achieved.

Example is as follows:

Instruction	Interpretation
start: LD A	Load "A" value to the accumulator.
AND B	Perform the "AND" operation of the "B" value and accumulator value.
OR C	Perform the "OR" operation of the "C" value and accumulator value.
ST D	Store the result in "D".
JMP start	Jump to the label "start" and have nothing to do with the accumulator value ("D" value).

#### JMPC

By "JMPC" instruction, a conditional jump to the label can be achieved. (The condition is 1.)

Example is as follows:

Instruction	Interpretation
start: LD A	Load "A" value to the accumulator.
AND B	Perform the "AND" operation of the "B" value and accumulator value.
OR C	Perform the "OR" operation of the "C" value and accumulator value.
JMPC start	Only when the accumulator value is 1, the jump to label "start" is achieved.

#### JMPCN

By "JMPCN" instruction, a conditional jump to the label can be achieved. (The condition is

0.)

Example is as follows:

Instruction	Interpretation
start: LD A	Load "A" value to the accumulator.
AND B	Perform the "AND" operation of the "B" value and accumulator value.
OR C	Perform the "OR" operation of the "C" value and accumulator value.
JMPCN start	Only when the accumulator value is 0, the jump to label "start" is achieved.

## 8.4.8 Call (CAL, CALC and CALCN)

### CAL

By "CAL" instruction, an unconditional call of the function block and subprogram can be achieved.

Example is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
AND B	Perform the "AND" operation of the "B" value and accumulator value.
OR C	Perform the "OR" operation of the "C" value and accumulator value.
ST D	Store the result in "D".
CAL IL1	Call the subprogram "IL1" and have nothing to do with the accumulator value ("D" value).

### CALC

By "CALC" instruction, a conditional call of the function block and subprogram can be achieved. (The condition is 1.)

Example is as follows:

Instruction	Interpretation
LD A	Load "A" value to the accumulator.
AND B	Perform the "AND" operation of the "B" value and accumulator value.
OR C	Perform the "OR" operation of the "C" value and accumulator value.
CALC IL1	Call subprogram "IL1" only when the accumulator value is 1.

### CALCN

By “CALCN” instruction, a conditional call of the function block and subprogram can be achieved. (The condition is 0.)

Example is as follows:

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
AND B	Perform the “AND” operation of the “B” value and accumulator value.
OR C	Perform the “OR” operation of the “C” value and accumulator value.
CALCN IL1	Call subprogram “IL1” only when the accumulator value is 0.

## 8.4.9 Return (RET, RETC and RETCN)

### RET

An unconditional return from subprogram can be achieved by “RET” instruction.

Example is as follows:

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
AND B	Perform the “AND” operation of the “B” value and accumulator value.
OR C	Perform the “OR” operation of the “C” value and accumulator value.
ST D	Store the result in “D”.
RET	Return to the main program from subprogram and have nothing to do with the accumulator value (“D” value).

### RETC

A conditional return from subprogram can be achieved by “RETC” instruction. (The condition is 1.)

Example is as follows:

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
AND B	Perform the “AND” operation of the “B” value and accumulator value.
OR C	Perform the “OR” operation of the “C” value and accumulator value.
RETC	Return to the main program from subprogram only when the accumulator value is 1.

### RETCN

A conditional return from subprogram can be achieved by “RETCN” instruction. (The condition is 0.)

Example is as follows:

Instruction	Interpretation
LD A	Load “A” value to the accumulator.
AND B	Perform the “AND” operation of the “B” value and accumulator value.
OR C	Perform the “OR” operation of the “C” value and accumulator value.
RETCN	Return to the main program from subprogram only when the accumulator value is 0.

## 8.5 Function block

According to the different functions, the basic functional blocks can be divided into mathematics, statistics, logic, comparison, conversion, move, timer, counter, control, PLC, and other groups. First select the group in the toolbar as shown in Fig.8.1, and select the specific function block as shown in Fig.8.2, then click the icon , the selected function block is inserted in IL editor. All the basic functions and parameters of function blocks are described in Chapter 5 "Basic function block".

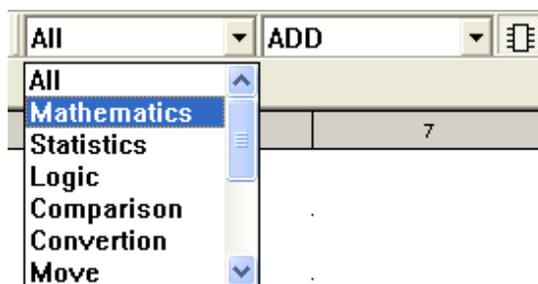


Fig.8.1 IL group selection

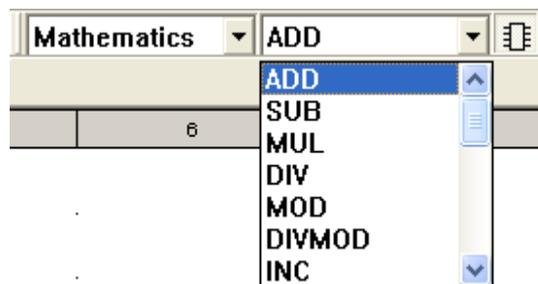


Fig.8.2 IL function block selection

## Chapter 9 ST Programming

The ST language is ostensibly similar to PASCAL language. However, it is a specialized programming language for industrial control applications, which has strong programming capability, and is used for variable assignment and functional blocks, creating expressions, editing conditional statements and iterative programs etc. ST is very suitable for applications with complex arithmetic.

The ST program format is free, and it can be inserted tabs, newline characters, and comments into any places between the keywords and identifiers. For the users familiar with the development of high-level computer language, the ST language is easy to learn and use. Moreover, the ST language is easy to read and understand, especially when using the meaningful identifiers and annotations to comment.

### 9.1 Expression

The expression is consisted of the operator, base function and operand.

#### 9.1.1 Operand

The operands can be constants, points, etc.

#### 9.1.2 Operator table

Operator	Operator meaning	Operand type	Level
()	Use parameters	Expression	1
NOT	Negation	Expression, constant, I, Q, IW, QW, M, MW, N, NW, S, SW	2
*	Multiplication	Expression, constant, IW, QW, MW, NW, SW	3
/	Division	Expression, constant, IW, QW, MW, NW, SW	3
MOD	Modulo	Expression, constant, IW, QW, MW, NW, SW	3
+	Addition	Expression, constant, IW, QW, MW, NW, SW	4

-	Subtraction	Expression, constant, IW, QW, MW, NW, SW	4
<	Less than	Expression, constant, IW, QW, MW, NW, SW	5
>	Greater than	Expression, constant, IW, QW, MW, NW, SW	5
<=	Less than or equal to	Expression, constant, IW, QW, MW, NW, SW	5
>=	Greater than or equal to	Expression, constant, IW, QW, MW, NW, SW	5
=	Equal to	Expression, constant, I, Q, IW, QW, M, MW, N, NW, S, SW	6
<>	Not equal to	Expression, constant, I, Q, IW, QW, M, MW, N, NW, S, SW	6
AND	And	Expression, constant, I, Q, IW, QW, M, MW, N, NW, S, SW	7
XOR	Exclusive Or	Expression, constant, I, Q, IW, QW, M, MW, N, NW, S, SW	8
OR	Or	Expression, constant, I, Q, IW, QW, M, MW, N, NW, S, SW	9
:=	Assignment	Q, QW, M, MW, N, NW	10

## 9.2 Operator

### 9.2.1 Parentheses ( ( ) )

The execution sequence of operators can be changed by using the parentheses.

In the following example, if "A" is 1, "B" is 2, "C" is 3, and "D" is -4, then:

Instruction	Result
E:=A+B-C*D	"E" is 15.
E:=(A+B-C)*D	"E" is 0.

### 9.2.2 Negation (NOT)

The operand will inverse by bit by using "NOT".

In the following example, if “A” is “11011011”, then:

Instruction	Result
B:=NOT A	“B” is 00100100.

### 9.2.3 Multiplication (\*)

The first operand value will multiply with the second operand value by using “\*”.

In the following example, if “A” is 2, and “B” is 3, then:

Instruction	Result
C:=A*B	“C” is 6.

### 9.2.4 Division (/)

The first operand value will be divided by the second operand value by using “/”.

In the following example, if “A” is 6, and “B” is 3, then:

Instruction	Result
C:=A/B	“C” is 2.

### 9.2.5 Modulo (MOD)

The first operand value will be divided by the second operand value and the remainder will be shown on the result column by using “MOD”.

In the following example, if “A” is 6, and “B” is 4, then:

Instruction	Result
C:=A MOD B	“C” is 2.

### 9.2.6 Addition (+)

The first operand value is added to the second operand value by using “+”.

In the following example, if “A” is 6, and “B” is 4, then:

Instruction	Result
C:=A+B	“C” is 10.

## 9.2.7 Subtraction (-)

The first operand value will subtract the second operand value by using “-”.

In the following example, if “A” is 6, and “B” is 4, then:

Instruction	Result
C:=A-B	“C” is 2.

## 9.2.8 Greater than (>)

By “>” operation, the first operand value will be compared with the second operand value. If the first operand value is greater than the second operand value, then the result is Boolean 1. If the first operand value is less than or equal to the second operand value, then the result is Boolean 0.

The example is as follows:

Instruction	Result
C:=A>B	If “A” is 6, and “B” is 4, then “C” is 1.
C:=A>B	If “A” is 2, and “B” is 4, then “C” is 0.
C:=A>B	If “A” is 2, and “B” is 2, then “C” is 0.

## 9.2.9 Greater than or equal to (>=)

By “>=” operation, the first operand value will be compared with the second operand value. If the first operand value is greater than or equal to the second operand value, then the result is Boolean 1. If the first operand value is less than the second operand value, then the result is Boolean 0.

The example is as follows:

Instruction	Result
C:=A>=B	If “A” is 6, and “B” is 4, then “C” is 1.
C:=A>=B	If “A” is 4, and “B” is 4, then “C” is 1.
C:=A>=B	If “A” is 2, and “B” is 4, then “C” is 0.

## 9.2.10 Equal to (=)

By “=” operation, the first operand value will be compared with the second operand value.

If the first operand value is equal to the second operand value, then the result is Boolean 1. If the first operand value is not equal to the second operand value, then the result is Boolean 0.

The example is as follows:

Instruction	Result
C:=A=B	If "A" is 4, and "B" is 4, then "C" is 1.
C:=A=B	If "A" is 4, and "B" is 6, then "C" is 0.
C:=A=B	If "A" is 6, and "B" is 4, then "C" is 0.

### 9.2.11 Not equal to (<>)

By "<>" operation, the first operand value will be compared with the second operand value. If the first operand value is not equal to the second operand value, then the result is Boolean 1. If the first operand value is equal to the second operand value, then the result is Boolean 0.

The example is as follows:

Instruction	Result
C:=A<>B	If "A" is 4, and "B" is 4, then "C" is 0.
C:=A<>B	If "A" is 4, and "B" is 6, then "C" is 1.
C:=A<>B	If "A" is 6, and "B" is 4, then "C" is 1.

### 9.2.12 Less than (<)

By "<" operation, the first operand value will be compared with the second operand value. If the first operand value is less than the second operand value, then the result is Boolean 1. If the first operand value is greater than or equal to the second operand value, then the result is Boolean 0.

The example is as follows:

Instruction	Result
C:=A<B	If "A" is 4, and "B" is 6, then "C" is 1.
C:=A<B	If "A" is 6, and "B" is 4, then "C" is 0.
C:=A<B	If "A" is 6, and "B" is 6, then "C" is 0.

### 9.2.13 Less than or equal to (<=)

By “<=” operation, the first operand value will be compared with the second operand value. If the first operand value is less than or equal to the second operand value, then the result is Boolean 1. If the first operand value is greater than the second operand value, then the result is Boolean 0.

The example is as follows:

Instruction	Result
C:=A<=B	If “A” is 4, and “B” is 6, then “C” is 1.
C:=A<=B	If “A” is 6, and “B” is 6, then “C” is 1.
C:=A<=B	If “A” is 6, and “B” is 4, then “C” is 0.

### 9.2.14 And (AND)

By “AND” operation, the “AND” link among each operation can be set. For the integer data type, the operation operates bit by bit.

The example is as follows:

Instruction	Result
D:=A AND B AND C	If any one of “A”, “B” and “C” is 0, then “D” is 0, otherwise, “D” is 1.

### 9.2.15 Or (OR)

By “OR” operation, the “OR” link among each operation can be set. For the integer data type, the operation operates bit by bit.

The example is as follows:

Instruction	Result
D:=A OR B OR C	If any one of “A”, “B” and “C” is 1, then “D” is 1, otherwise, “D” is 0.

### 9.2.16 Exclusive or (XOR)

By “XOR” operation, the “XOR” link among each operation can be set. For the integer data type, the operation operates bit by bit.

In the following example, if “A” is different to “B”, then “C” is 1. If “A” is same to B (for

example both of them are 0 or 1), then “C” is 0.

Instruction	Result
C:=A XOR B	If “A” is 1, and “B” is 0, then “C” is 1.
C:=A XOR B	If “A” is 0, and “B” is 1, then “C” is 1.
C:=A XOR B	If “A” is 1, and “B” is 1, then “C” is 0.
C:=A XOR B	If “A” is 0, and “B” is 0, then “C” is 0.

### 9.2.17 Assignment (:=)

By “:=” operation, the current value of variable will be replaced by the assignment result of expression. The assignment includes the left variable specification, the second following assignment operator “:=”, and the third following expression to assign.

In the following example, the assignment is used to assign one variable value to another variable.

Instruction	Interpretation
A:=B	Replace “A” value by the current value of “B”.

In the following example, the assignment is used to assign the immediate for variable.

Instruction	Interpretation
A:=10	Assign 10 to “A”.

In the following example, the assignment is used to assign the result returned from function block to variable.

Instruction	Interpretation
A:=ABS(B)	Assign the result of function block “ABS” to “A”.

In the following example, Assignment is used to assign the operation result to variable.

Instruction	Interpretation
A:=(B+C-D)*E	Assign the operation result of “(B+C-D)*E” to “A”.

## 9.3 Statement

### 9.3.1 Instruction

Instruction is the “command” of structured programming language, which must end with a semicolon. The multiple instructions can be input into one line (each instruction is separated by semicolon).

### 9.3.2 IF...THEN...ELSE...END\_IF

When the Boolean value of instruction behind “IF” is 1 (TRUE), the instruction or instruction set behind “THEN” will be executed. When the Boolean value of instruction behind “IF” is 0 (FALSE), the instruction or instruction set behind “ELSE” will be executed. “END\_IF” instruction is used to mark the end of instruction.

The example is as follows.

Instruction	Interpretation
IF %M1 = 1	Judge whether “%M1” meets the condition or not?
THEN %MW1 := 1;	If met, execute this instruction.
ELSE %MW1 := 2;	If not met, execute this instruction.
END_IF;	Mark the end of instruction.

### 9.3.3 CASE...OF... ELSE... END\_CASE

“CASE” instruction is consisted of an integer data type expression and a set of instructions. Each set has a mark, and this mark is consisted of one or multiple integer(s). If the mark contains the calculated selector value, then the instruction will be executed; if the mark does not contain that value, then the instruction will not be executed. “OF” instruction indicates the start of mark. “ELSE” instruction can be executed in the “CASE” instruction; however, it only can be executed when the mark does not contain any selector value. “END\_CASE” is used to mark the end of instruction.

The example is as follows.

Instruction	Interpretation
CASE A+B OF	Which conditions does the value of “A+B” in multi-branch statement meet?
1,5: C:=SIN(A) * COS(B);	If the value is 1 or 5, then this instruction is

	executed.
2: B:=C-A;	If the value is 2, then this instruction is executed.
3,4,6: C:=C*A;	If the value is 3, 4 or 6, then this instruction is executed.
ELSE B:=C*A; C:=A / B;	If any above condition is not met, then this instruction is executed.
END_CASE;	Mark the end of instruction.

### 9.3.4 FOR...TO...BY...DO...END\_FOR

“FOR” instruction is used on the condition that the number of current matching items can be determined. If the number of current matching items can’t be determined, only the “WHILE” or “REPEAT” instruction can be used. “FOR” instruction is used to repeat the sequence of instructions till “END\_FOR” instruction occurs. The number of matching items is dependent on the start value, end value and control variable which must be the same data type and can’t be modified by one of repeating instructions. “FOR” instruction is used to add the control variable value from start value to end value. The default value of incremental value is 1. If use other value, the explicit incremental value (constant) can be specified. You need to check the variable value before each update loop running. If the variable value is out of the range of the start value and end value, the loop is no use. Before the first loop running, the check must be executed so as to determine the incremental value of control variable, and to determine whether it begins from the start value, and moves to the end value. If no moving, the loop will be not executed. Using this principle, the loop can be executed normally. “DO” command is used to identify the ending of repeating definitions and the starting of commands. Also you can end the repeating earlier by “EXIT” instruction. “END\_FOR” is used to mark the end of instruction.

“FOR” example with the incremental value of 1: in the following example, “I” is the control variable, 1 is the start value, and 5 is the end value.

Instruction	Interpretation
FOR I:= 1 TO 5 DO	“I” is increased by the default value 1, the looping execution of the following instruction is done, when “I” is greater than 5, end the “FOR” looping instruction.
C:= C+4;	The instruction set of looping execution.
END_FOR;	Mark the end of instruction.

If the incremental value is not equal to 1, then you can define the incremental value using “BY”. The processing direction (forward or backward) is dependent on the symbol of “BY” constant. If the symbol is positive, the loop will run forward, if negative, the loop will run backward.

“FOR” example with the incremental value of not equal to 1: in the following example, “I” is the control variable, “1” is the start value, and 5 is the end value.

Instruction	Interpretation
FOR I:= 1 TO 5 BY 2 DO	“I” is increased by the incremental value 2, the looping execution of the following instruction is done, when “I” is greater than 5, end the “FOR” looping instruction.
C:= C+4;	The instruction set of looping execution.
END_FOR;	Mark the end of instruction.

### 9.3.5 WHILE...DO...END\_WHILE

The execution result of “WHILE” instruction is that: the instruction sequence will be repeatedly executed till its associated Boolean expression is 0 (FALSE). If the start of expression is 0, then the instruction set will be not executed. “DO” command is used to identify the ending of repeating definitions and the starting of commands. Also you can end the process earlier by “EXIT” instruction. “END\_WHILE” is used to mark the end of instruction.

The example is as follows.

Instruction	Interpretation
A := 1;	Assign the constant 1 to “A”.
WHILE A<= 100 DO A := A + 4;	If “A” is less than or equal to 100, then repeatedly execute the loop instruction “A := A + 4”; If “A” is greater than 100, then no longer repeatedly execute the loop instruction “A := A + 4”.
END_WHILE;	Mark the end of instruction.

### 9.3.6 REPEAT...UNTIL...END\_REPEAT

The execution result of “REPEAT” instruction is that the instruction sequence will be

repeatedly executed (once at least) till its associated Boolean expression is 1 (TRUE). “UNTIL” instruction is used to mark the end of condition. You can end the process earlier by “EXIT” instruction. “END\_REPEAT” is used to mark the end of instruction.

The example is as follows.

Instruction	Interpretation
A := 1;	Assign the constant 1 to "A".
REPEAT	
A := A + 2	Execute the loop instruction "A := A + 2".
UNTIL A >= 100	If "A" is less than 100, then execute the loop instruction "A := A + 2"; If "A" is greater than or equal to 100, then end the loop instruction.
END_REPEAT;	Mark the end of instruction.

### 9.3.7 EXIT

“EXIT” instruction is used to end the repeat instruction (such as “FOR”, “WHILE”, “REPEAT”) before encountering the end condition. If the “EXIT” instruction exists in embedded matching item, the most inner loop will be remained (that is the position of “EXIT”). The next operation is to execute the first instruction following behind the loop end (“END\_FOR”, “END\_WHILE” or “END\_REPEAT”).

### 9.3.8 RETURN

“RETURN” instruction is used to interrupt the scan of sub-program, and return to the main-program without conditions.

### 9.3.9 Comment

In ST editor, comment always begins with character string “(”, and end with character string “)”. Any comment can be inserted into these two character strings. Also any place of ST editor can be input comment. The comment will display with different color.

The character string “//” is used to comment current line.

### 9.3.10 GOTO

In ST editor, the label is end with “:”, such as “AA:”. “GOTO” statement is used to achieve the jump to the specified label, such as “GOTO AA;”.

## 9.4 Function block

According to the different functions, the basic functional modules can be divided into mathematics, statistics, logic, comparison, conversion, move, timer, counter, control, PLC and other groups. First select the group in the toolbar as shown in Fig.9.1, and select the specific function block as shown in Fig.9.2, then click the icon , the selected function block is inserted in ST editor. All the basic functions and parameters of function blocks are described in Chapter 5 "Basic function block".

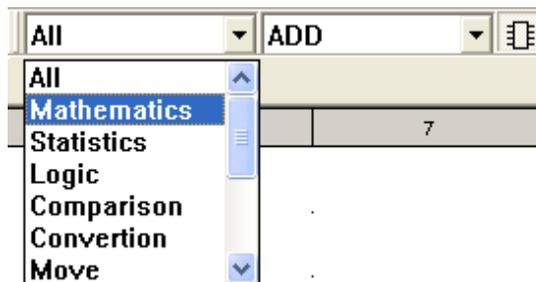


Fig.9.1 ST group selection

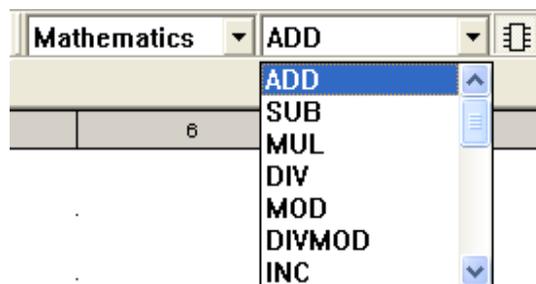


Fig.9.2 ST function block selection

## Chapter 10 Program Debugging

NA200Pro programming software supports a variety of programming languages such as LD, FBD, IL, and ST. According to the characteristics of each programming language, NA200Pro designs a variety of debugging methods to help technologists to debug and modify the program conveniently during the implementation of project, and also setups two debugging environments of PLC online debugging and simulator online debugging. For the PLC online debugging, the program debugging is carried out when NA200Pro directly links with PLC, and the modifications are directly downloaded into PLC. For the simulator online debugging, it will simulate an environment that the NA200Pro and PLC are virtually linked, and can carry out preliminary tests of program correctness by the means of setting the variable, forcing I/O points, etc. to ensure the success of online debugging and the security of equipment operation and debugging. The following is the illustrations of detailed debugging steps for different programming languages.

### 10.1 LD / FBD debugging

#### 10.1.1 Online modifying

The debugging of LD/ FBD is quite intuitive. After online, all the conducted currents are red (%M0008), the non-conducted currents are green (%M0007), as shown in Fig.10.1. LD program is generally used to execute the state judgment, so you can directly view the point state.

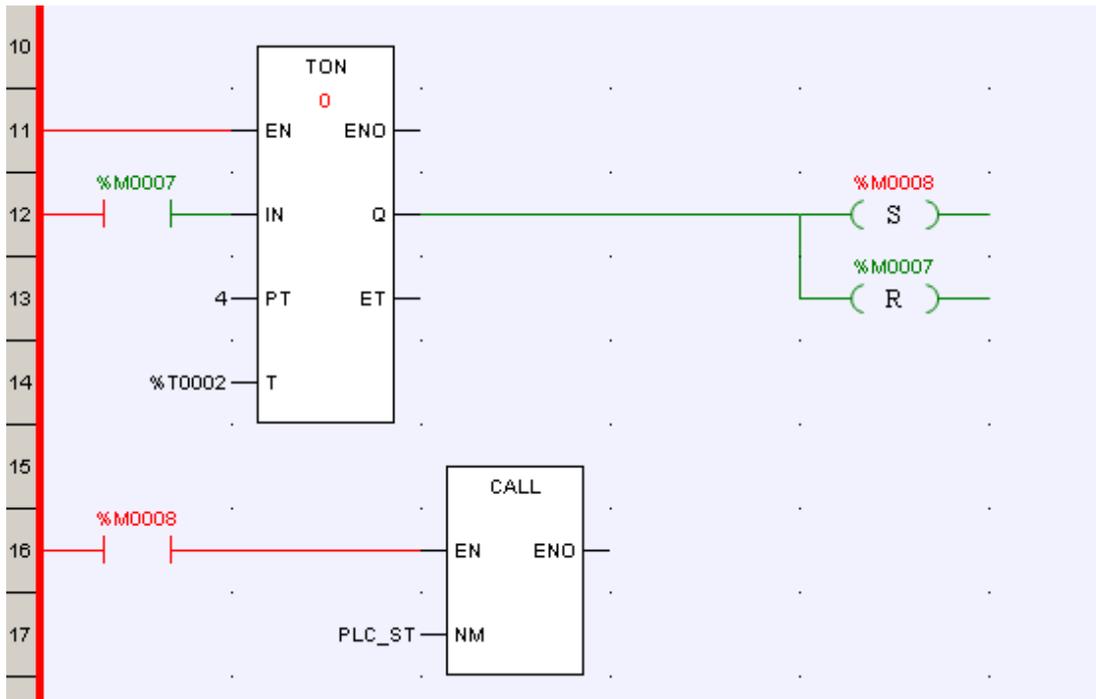


Fig.10.1 LD online

After online, the modification of LD/ FBD can be done yet, as shown in Fig.10.2.

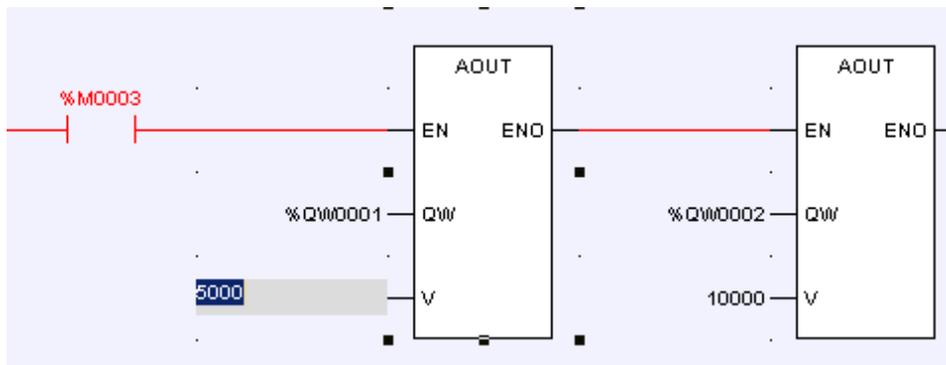


Fig.10.2 Modify parameter

After online and the program has been modified, the "\*" mark will be added on the program name in project browser, for example the "MAIN\*" as shown in Fig.10.3, it indicates that the current program has been modified but not downloaded yet.

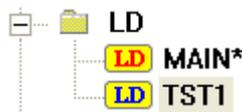


Fig.10.3 Program modified but not downloaded

After completing the modification, select the system toolbar  icon to save the modified file, then select the **【Online】 / 【Refresh Program】**, as shown in Fig.10.4, download the modified program to PLC.



Fig.10.4 Refresh program

## 10.1.2 Online debugging

The online debugging function is generally used in the single-step debugging of program, and a breakpoint can be set in any function block of LD/FBD during the debugging process. When the program executes to the breakpoint, it will stop and wait the instruction **Step**. If the program debugging is finished, clear all breakpoints, and click the **Continue** button, the program will continue normal scan.

### • Step

When debugging the program, each function block is one step, so it can be found that the program will stop in the original place after finishing each function block and wait for the instruction of **Step**. The way to send a **Step** instruction is as follows: move the mouse to point to the **Step** button in LD/FBD toolbar, and then press the left button to send out this instruction.

### • Continue

When the program stops execution, press the **Continue** button in LD/FBD toolbar, then the program will continue the execution. If there are breakpoints in the program, the program will stop when encountering the first breakpoint and wait for the **Step** instruction. If there is no breakpoint in the program, the program will be executed all the time.

### • Insert/remove breakpoint

Breakpoint can be set at any function block of the program, and the method of setting breakpoint is shown as follows: select the function block need to set breakpoint (by the method of moving the mouse, pointing to the function block need to set breakpoint, and

pressing the left mouse button to select it), then move the mouse to the **Insert/Remove Breakpoint** button in LD/FBD toolbar, press the left mouse button, so a green dot appears on the top right of the selected function block, it indicates that the breakpoint is successfully set. If press the **Insert/Remove Breakpoint** button in LD/FBD toolbar again, the breakpoint is cleared. If press the button third time, the breakpoint is set again. A program can set 10 breakpoints at most in the different positions at the same time. As shown in Fig.10.5:

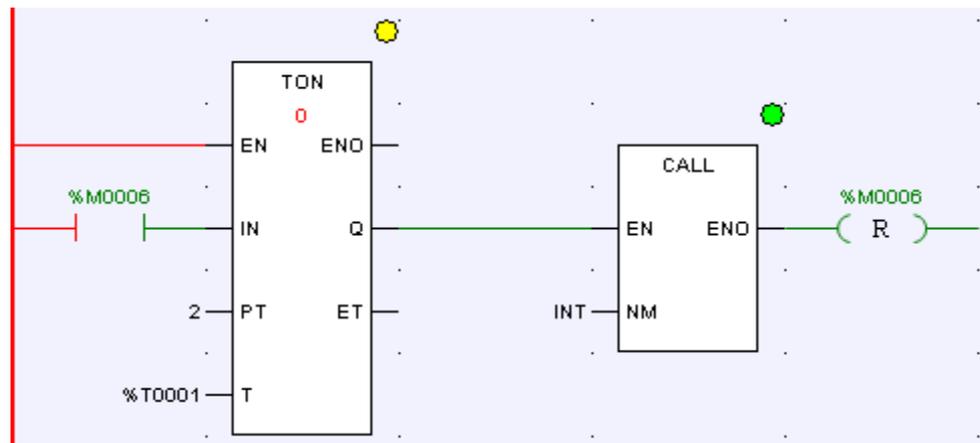


Fig.10.5 Set breakpoint in LD

- **Clear all breakpoints** 

All breakpoints in the program can be cleared once. The method is shown as follows: point to the **Clear All Breakpoints** button in LD/FBD toolbar and press the left button, then all breakpoints will be cleared.

※**After offline, all breakpoints will be automatically cleared.**

## 10.2 IL debugging

The online debugging function is mainly used in the single-step debugging of program. In the process of debugging, breakpoint can be set in any valid line of IL. The program execution will stop when encountering a breakpoint and wait for **Step** instruction. If debugging is completed, all breakpoints will be cleared. Click on the **Continue** button, and the program will continue normal scan.

- **Step** 

When debugging the program, each valid line is one step. It can be found that the program will stop in the original position and wait for a **Step** instruction after one step is finished. The method of sending **Step** instruction is shown as follows: move the mouse to the button of **Step** in IL toolbar and press the left mouse button, then a **Step** instruction is

sent.

- **Continue** 

When the program stops execution, press the **Continue** button in IL toolbar, then the program will continue the execution. If there are breakpoints in the program, the program will stop when encountering the first breakpoint and wait for the **Step** instruction. If there is no breakpoint in the program, the program will be executed all the time.

- **Insert/remove breakpoint** 

Breakpoint can be set in any valid line of the program, and the method of setting breakpoint is shown as follows: move the cursor to the line where need to set breakpoint, then move the mouse to the **Insert/Remove Breakpoint** button in IL toolbar, press the left mouse button, so the line will turn green, it is indicated that the breakpoint is successfully set. If press the **Insert/Remove Breakpoint** button again, the breakpoint is cleared. If press the button third time, the breakpoint is set again. A program can set 10 breakpoints at most in the different positions at the same time. As shown in Fig.10.6:

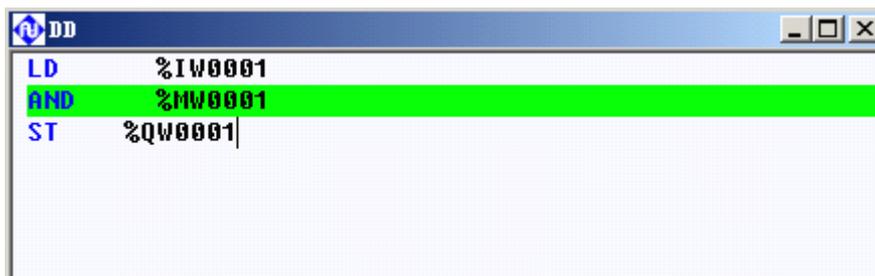


Fig.10.6 Set breakpoint in IL

- **Clear all breakpoints** 

All breakpoints in the program can be cleared once. The method is shown as follows: point to the **Clear All Breakpoints** button in IL toolbar and press the left button, then all breakpoints will be cleared.

※**After offline, all breakpoints will be automatically cleared.**

## 10.3 ST debugging

The online debugging function is mainly used in single-step debugging of program. In the process of debugging, breakpoint can be set in any valid line of ST. The program execution will stop when encountering a breakpoint and wait for **Step** instruction. If debugging is completed, all breakpoints will be cleared. Click on the **Continue** button, and the program will continue normal scan.

- **Step** 

When debugging the program, each valid line is one step. It can be found that the program will stop in the original position and wait for a **Step** instruction after one step is finished. The method of sending **Step** instruction is shown as follows: move the mouse to the button of **Step** in ST toolbar and press the left mouse button, then a **Step** instruction is sent.

- **Continue** 

When the program stops execution, press the **Continue** button in ST toolbar, then the program will continue the execution. If there are breakpoints in the program, the program will stop when encountering the first breakpoint and wait for the **Step** instruction. If there is no breakpoint in the program, the program will be executed all the time.

- **Insert/remove breakpoint** 

Breakpoint can be set in any valid line of the program, and the method of setting breakpoint is shown as follows: move the cursor to the line where need to set breakpoint, then move the mouse to the **Insert/Remove Breakpoint** button in ST toolbar, press the left mouse button, so the line will turn green, it is indicated that the breakpoint is successfully set. As shown in Fig.10.10. If press the **Insert/Remove Breakpoint** button again, the breakpoint is cleared. If press the button third time, the breakpoint is set again. A program can set 10 breakpoints at most in the different positions at the same time. As shown in Fig.10.7:

```

R0010 :=0;
FOR i := 1 TO I1_PCNT BY 1 DO
  IF I1_BUF[i]=1 THEN
    R0010 := R0010+1;

```

Fig.10.7 Set breakpoint in ST

- **Clear all breakpoints** 

All breakpoints in the program can be cleared once. The method is shown as follows: point to the **Clear All Breakpoints** button in ST toolbar and press the left button, then all breakpoints will be cleared.

※**After offline, all breakpoints will be automatically cleared.**

## 10.4 Simulator

NA200Pro software provides two debugging modes of simulator online and PLC online to test the finished programs.

Simulator online function means that the system will be virtual online when the NA200Pro

software doesn't link with PLC and carry out the virtual online debugging of the finished PLC program or carry out the simulated test without a real PLC, and simulate the production process of actual trial site and test the finished programs so as to achieve the aim of viewing the execution effect or the correctness of current programs.

PLC online function means that the NA200Pro software has already linked with PLC. Using the communication and powerful online monitoring capabilities of NA200Pro, according to the actual signals or analog signals or control processes, the system can carry out the online monitoring and debugging for the edited program to achieve the aim of viewing the execution effect or the correctness of current programs.

The differences between simulator online and PLC online are as follows:

In simulator online mode, NA200Pro software doesn't link with PLC, the force and modification of all signals only occur in the computer where NA200Pro is. The operation results or data are produced from the computer, however, it will never produce the real output control.

In PLC online mode, NA200Pro links with PLC, the force and modification of all signals occur in real PLC. The operation results or data are produced from PLC, and can generate the real output control.

The simulator online mode can be started by **【Online】/【Connect】/【Simulator】** menu, and the PLC online mode can be started by **【Online】/【Connect】/【PLC】** menu. Two modes can exit by **【Online】/【Disconnect】** menu.

Both modes can achieve the aim of viewing the execution effect or the correctness of current programs. In general, the simulator online mode is suitable for the project pre-design phase while PLC online mode is suitable for the plant debugging phase and the field debugging phase after sale.

The main functions of the simulator online mode are shown as follows:

1. It can force all input and output signals, and modify the value.
2. It can fully simulate the actual executive effects of logical links, application instructions and programs.

# Chapter 11 Communication Protocol

NA200 PLC is generally equipped with serial ports and supports MODBUS protocol.

## 11.1 General instructions

### 11.1.1 Exchange properties

MODBUS is a master/slave protocol, which allows the operations to read/write one or more words (16 bits), but doesn't allow the operations to read/write byte at any condition. The message exchange is carried out by master taking the initiative, that is, the exchange is started up by master. Except the broadcast message, all the complete exchange is consisted of two messages of downlink and uplink:

- ♦ Downlink message: send out a request from master
- ♦ Uplink message: send back a response from slave

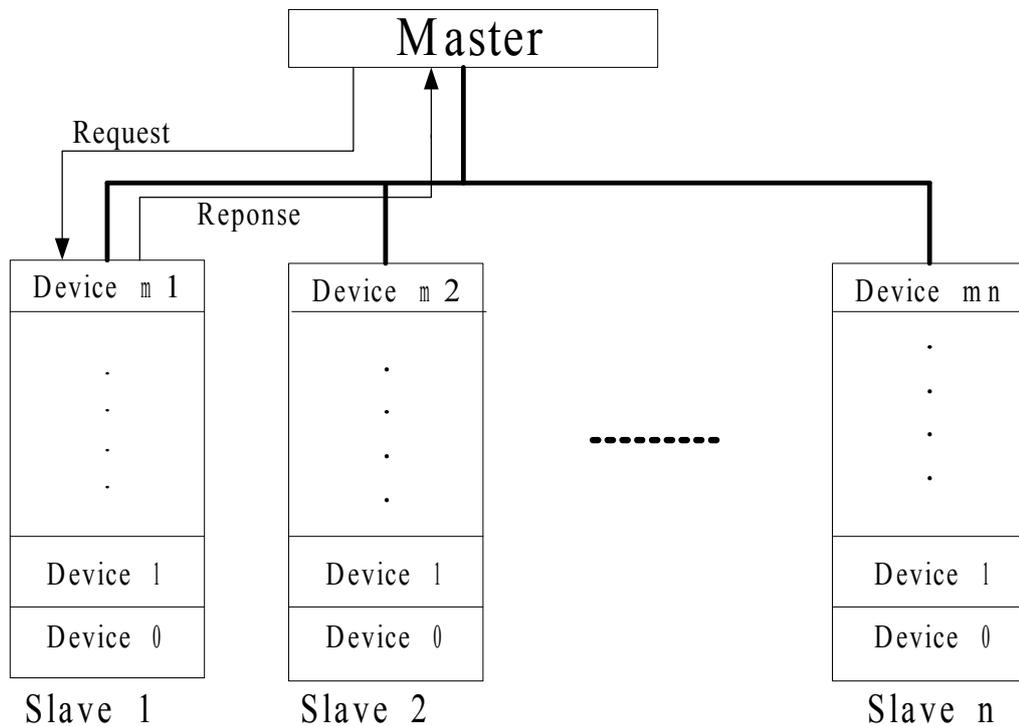


Fig.11.1 Exchange of general message

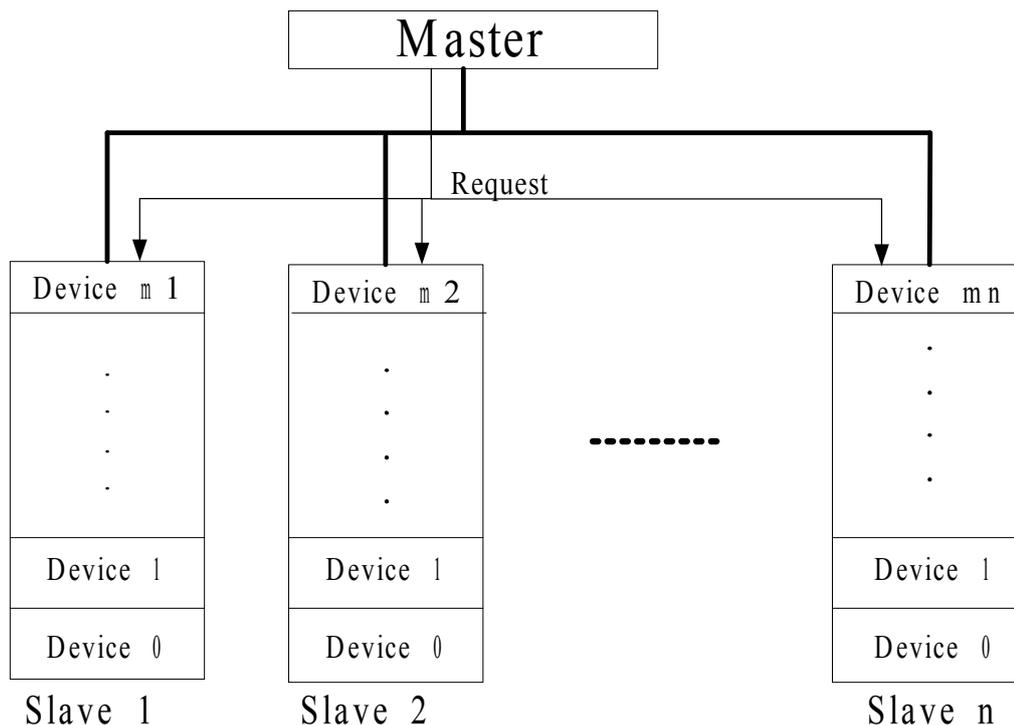


Fig.11.2 Exchange of broadcast message

The request sent from master is generally only sent out to a certain specified slave (it is identified by slave address which is specified by the first byte of request frame), as shown in Fig.11.1. In the broadcast message (the slave address is 0), this request will send out to all of the slaves, however, the protocol specifies that the broadcast message shall be the write command, and the slaves will never send back the responses, as shown in Fig.11.2.

### 11.1.2 Message format

All of the exchanged RTU type messages (frames), including uplink/downlink, have the same structure shown as follows.

Slave address	Function code	Data area	CRC16 check
1 Byte	1 Byte	n Bytes	2 Bytes

Each message includes 4 types of information:

#### Slave address

The slave address is 1 byte, and the value range is 0-FFH. Exceptionally, if the value is 0, it can be acted as the broadcast message identifier. Therefore, the slave address actually used shall be only 01H-FFH (That is 1-255).

#### Function code

The function code is 1 byte, and it can be used to choose a command (read, write or

answer whether the check is right or not, etc.). The range of effective function code is 1-255, and the function codes supported by this manual will be described in the following.

### Data area

The data area is n bytes; it includes a serial of hex data related to the function code.

### CRC16 check

CRC16 check is 2 bytes; it is a check from the first byte (slave address) of message to the end byte of data area, which is calculated by the algorithm of CRC16.

**【Example】** Use C language to implement the CRC16 algorithm, as follows:

```
unsigned short CRC16(unsigned char* buf, unsigned short len)
{
    unsigned short crc=0xffff;
    unsigned short i,j,k;
    for(i=0;i<len;i++)
    {
        crc =crc ^ buf[i];
        for(j=0;j<8;j++)
        {
            k=crc & 01;
            crc=crc >> 1;
            if (k==0) continue;
            crc =crc ^ 0xA001;
        }
    }
    return crc;
}
```

## 11.2 Addressing mode

As shown above in Fig.11.1 and Fig.11.2, the master can contact with n slave(s),  $1 \leq n \leq 255$ . Exceptionally,  $n = 0$  (that is the slave address=0) can be used as the broadcast message of master.

It is observed from Fig.11.1 and Fig.11.2 that, the access of master to slave actually comes down to access to a number of addresses in a certain device of a certain slave, therefore, the address accessed by master shall be determined by three factors: slave address, device No., and address in device.

In the message, both of the device No. and address in device will be together described as "word address".

### **11.2.1 Addressing mode 1**

The device No. acts as the high byte of “word address” (the device No. is generally called as offset), and the address in device acts as the low byte of “word address”. Obviously, for a specific device, the master can only access to the word address of 0-255, moreover, the length of word address continuously accessed by the master is far less than 255, for the details, please see the descriptions in chapter 11.2.3.

In this addressing mode, the slave address can be 255 at most, and each slave can be consisted of 256 devices at most, therefore, in this MODBUS network, it can be connected with  $255 \times 256 = 65280$  devices at most, the Fig.11.1 and Fig.11.2 are given out by this addressing mode.

### **11.2.2 Addressing mode 2**

In the description of mode 1 above, it is indicated that, for a specific device, the master can only access to the word address of 0-255. But in many application environments, the address range in device can be more than this range, in this way, the addresses in device can be divided into a number of pages (also these pages are generally called as offset), the address range of each page is also 0-255. The access format of mode 2 is the same to that of mode 1 actually.

### **11.2.3 Difference of two addressing modes**

Mode 1: Each slave can be connected with 255 devices at most, but the address range of each device is 0-255.

Mode 2: For each slave, the amount of connected device is less than that in mode 1, but the address range in device can be larger than 255. When the number of devices in slave is equal to 1, theoretically, the largest number of word address that can be accessed is equal to  $256 \times 256 = 65536$ , that is, the range is 0-65535, and however, the length limit for word address that can be continuously accessed each time by master is also the same to that of mode 1.

### **11.2.4 Special instructions for addressing mode 2**

In many applications, the slave is connected with only 1 device, at this time, there is no concept difference between the “slave” and “device” connected on the slave, and both of the “page” and “word address” is combined into the word address in a broad sense.

## 11.3 Error response

Except the broadcast message, for the requests sent from master, the slaves make the following normal or abnormal responses:

- If the slave communication is normal, and the message contents are correct, then the slave will send back the corresponding response message;
- If the slave can not normally receive the request message for the communication failure, in this condition, the master will make a deal with the corresponding time-out error;
- If the slave can receive the request message from master, but there is a CRC16 check error, in this condition, the slave does not send out the response message, and the master will make a deal with the corresponding time-out error;
- If the slave can normally receive the request message from master, but can not normally deal with the request message (For example, the master requests to read the inexistent measuring points), in this condition, the slave will send back the error response message, and prompt the master. In the error response message:

<b><i>Slave address</i></b>	<b><i>Function code   80H</i></b>	<b><i>Error type code</i></b>	<b><i>CRC16 check</i></b>
1 byte	1 byte	1 byte	2 bytes

**Function code:** 1 byte, function code of master request | 80H;

**Data area:** 1 byte, error type code, as follows:

<b><i>Error type code</i></b>	<b><i>Definition</i></b>	<b><i>Meaning</i></b>
01	ILLEGAL_FUNCTION	The function code is illegal.
02	ILLEGAL_DATA_ADDRESS	The data address requesting to access is illegal.
03	ILLEGAL_DATA_VALUE	The data of request message is illegal.
04	SLAVE_DEVICE_FAILURE	If the slave makes a response to the request, the unrecoverable failure occurs.
05	ACKNOWLEDGE	The slave needs to take a long time to deal with the request from master.

<b>Error type code</b>	<b>Definition</b>	<b>Meaning</b>
06	SLAVE_DEVICE_BUSY	The slave is busy, it can not make a response to the request temporarily, please resend it later.

**【Example】** Slave error response message

The master requests to read the digital output status with the address of 1234 (04A1H) in slave (slave address is 10), but in the slave, this digital output measuring point does not exist, then the slave sends back an error message.

The request message from master:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	0A
2	Function code	01
3	High 8-bit of start address	04
4	Low 8-bit of start address	A1
5	High 8-bit of data number	00
6	Low 8-bit of data number	01
CRC16	-	-

The error response message from slave:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	0A
2	Function code	81
3	Error type code	02
CRC16	-	-

## 11.4 MODBUS protocol

### 11.4.1 Function code descriptions

In this manual, MODBUS uses the following function codes:

<b>Function code (Decimal)</b>	<b>Meaning</b>
--------------------------------	----------------

<b><i>Function code (Decimal)</i></b>	<b><i>Meaning</i></b>
01	Read Coil Status
02	Read Input Status
03	Read Holding Registers
04	Read Input Registers
05	Force Single Coil
06	Write Single Register
15	Force Multiple Coils
16	Write Multiple Registers

### 11.4.2 Function code and data classification

In this manual, the relationships in the function codes and the corresponding data are as follows (Take CPU401-0301 for example, the address upper limits of the other type CPU are different):

<b><i>Reference</i></b>	<b><i>Type of measuring point</i></b>	<b><i>Read operation function code</i></b>	<b><i>Write operation function code</i></b>	<b><i>Range of measuring point address</i></b>	<b><i>Range of protocol address</i></b>
0X	Q	01	05 / 15	0001-0128	00000-00127
	M	01	05 / 15	0001-1024	02000-03023
	N	01	05 / 15	0001-0256	07000-07255
1X	I	02		0001-0128	00000-00127
	S	02		0001-0256	02000-02255
3X	IW	04		0001-0032	00000-00031
	SW	04		0001-0256	02000-02255
4X	MW	03	06 / 16	0001-1024	00000-01023
	NW	03	06 / 16	0001-0256	05000-05225
	QW	03	06 / 16	0001-0032	08000-08031
	Clock	03	16		09994-09998

In the MODBUS protocol, the address start number is 0. For example, the protocol address of %Q0005 is 4.

System clock format:

	Year	Month	Day	Hour	Minute	Second	Millisecond
Address	09994	09995	09995	09996	09997	09997	09998
Byte	2	1	1	2	1	1	2

### 11.4.3 Details for function code

#### I. 01 Read coil status

##### 1. Description

Reads the ON/OFF status of measuring points Q, M and N (0X references) in the slave.

##### 2. Master request

The request message defines the start address and quantity of measuring points to be read. Coils are addressed starting at zero: the address of %Q1-%Q128 is 0-127; the address of %M1-%M1024 is 2000-3023; the address of %N1-%N256 is 7000-7255.

【Example】 Request to read %Q20-%Q56 from slave 17 is as follows:

<i>Byte</i>	<i>Meaning</i>	<i>Example (HEX)</i>
1	Slave address	11
2	Function code	01
3	High 8-bit of start address	00
4	Low 8-bit of start address	13
5	High 8-bit of data number	00
6	Low 8-bit of data number	25
CRC16	-	-

##### 3. Slave response

- ♦ The status data sent back from slave complies that: the status of start address is indicated by the LSB of the first byte of data, and also complies that it is continuously stored “from low bit to high bit, and from low byte to high byte” by bit.
- ♦ If the number of return status is not the integral multiple of 8, then fill the extra bits of highest byte with “0”.
- ♦ Status representation: 1 = ON; 0 = OFF.

【Example】 Response to the request above is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	01
3	Byte count	05
4	Data 1 (%Q27-%Q20)	CD
5	Data 2 (%Q35-%Q28)	6B
6	Data 3 (%Q43-%Q36)	B2
7	Data 4 (%Q51-%Q44)	0E
8	Data 5 (%Q56-%Q52)	1B
CRC16	-	-

Wherein,

- The status of %Q27-%Q20 is CDH (1100 1101 B), that is, the most significant bit (MSB) of CDH is the status of %Q27, and the least significant bit (LSB) is the status of %Q20. The actual status of %Q27-%Q20 is ON-ON-OFF-OFF-ON-ON-OFF-ON.
- The last byte 1BH represents the status of %Q56-%Q52: ON-ON-OFF-ON-ON. Fill the highest 3 bits of this byte with "0".

## II. 02 Read input status

### 1. Description

Reads the ON/OFF status of measuring points I and S (1X references) in the slave.

### 2. Master request

The request message defines the start address and quantity of measuring points to be read. Inputs are addressed starting at zero: the address of %I1-%I128 is 0-127; and the address of %S1-%S256 is 2000-2255.

**【Example】** Request to read %I20-%I56 from slave 17 is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	02
3	High 8-bit of start address	00
4	Low 8-bit of start address	13

5	High 8-bit of data number	00
6	Low 8-bit of data number	25
CRC16	-	-

### 3. Slave response

- ♦ The status data sent back from slave complies that: the status of start address is indicated by the LSB of the first byte of data, and it also complies that it is continuously stored “from low bit to high bit, and from low byte to high byte” by bit.
- ♦ If the number of return status is not the integral multiple of 8, then fill the extra bits of highest byte with “0”.
- ♦ Status representation: 1 = ON; 0 = OFF.

【Example】 Response to the request above is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	02
3	Byte count	05
4	Data 1 (%I27-%I20)	CD
5	Data 2 (%I35-%I28)	6B
6	Data 3 (%I43-%I36)	B2
7	Data 4 (%I51-%I44)	0E
8	Data 5 (%I56-%I52)	1B
CRC16	-	-

Wherein,

- ♦ The status of %I27-%I20 is CDH (1100 1101 B), that is, the most significant bit (MSB) of CDH is the status of %I27, and the least significant bit (LSB) is the status of %I20. The actual status of %I27-%I20 is ON-ON-OFF-OFF-ON-ON-OFF-ON.
- ♦ The last byte 1BH represents the status of %I56-%I52: ON-ON-OFF-ON-ON. Fill the highest 3 bits of this byte with “0”.

## III. 03 Read holding registers

### 1. Description

Reads the binary contents of measuring points MW, NW, QW and system time (4X references) in the slave.

## 2. Master request

The request message defines the start address and quantity of measuring points to be read. Registers are addressed starting at zero: the address of %MW1-%MW1024 is 0-1023; the address of %NW1-%NW256 is 5000-5255; the address of %QW1-%QW32 is 8000-8031; and the address of system time is 9994-9998.

【Example】 Request to read %MW108-%MW110 from slave 17 is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	03
3	High 8-bit of start address	00
4	Low 8-bit of start address	6B
5	High 8-bit of data number	00
6	Low 8-bit of data number	03
CRC16	-	-

## 3. Slave response

- ♦ The data returned from slave complies that: it is stored continuously by “high byte in front, low byte at behind”

【Example】 Response to the request above is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	03
3	Byte count	06
4	High 8-bit of data 1 (%MW108)	02
5	Low 8-bit of data 1 (%MW108)	2B
6	High 8-bit of data 2 (%MW109)	00
7	Low 8-bit of data 2 (%MW109)	00
8	High 8-bit of data 3 (%MW110)	00
9	Low 8-bit of data 3 (%MW110)	64
CRC16	-	-

In the example, the measuring values of %MW108-%MW110 are 022BH, 0000H and 0064H.

#### 4. Read system time

The address of system time is 9994-9998.

【Example】 Request to read the system time from slave 17 is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	03
3	High 8-bit of start address	27
4	Low 8-bit of start address	0A
5	High 8-bit of data number	00
6	Low 8-bit of data number	05
CRC16	-	-

【Example】 Response to the request above is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	03
3	Byte count	0A
4	High 8-bit of year	07
5	Low 8-bit of year	D8
6	Day	0A
7	Month	09
8	High 8-bit of hour	00
9	Low 8-bit of hour	0C
10	Second	0E
11	Minute	0D
12	High 8-bit of millisecond	01
13	Low 8-bit of millisecond	02

CRC16	-	-
-------	---	---

In the example,

- The return time is: year 2008(07D8H), month 9(09H), day 10(0AH), hour 12(0CH), minute 13(0DH), second 14(0EH), and millisecond 258(0102H).

## IV.04 Read Input registers

### 1. Description

Reads the binary contents of measuring points IW and SW (3X references) in the slave.

### 2. Master request

The request message defines the start address and quantity of measuring points to be read. Registers are addressed starting at zero: the address of %IW1-%IW32 is 0-31; the address of %SW1-%SW256 is 2000-2255.

**【Example】** Request to read %IW18-%IW20 from slave 17 is as follows:

<b><i>Byte</i></b>	<b><i>Meaning</i></b>	<b><i>Example (HEX)</i></b>
1	Slave address	11
2	Function code	04
3	High 8-bit of start address	00
4	Low 8-bit of start address	11
5	High 8-bit of data number	00
6	Low 8-bit of data number	03
CRC16	-	-

### 3. Slave response

- The data returned from slave complies that: it is stored continuously by “high byte in front, low byte at behind”

**【Example】** Response to the request above is as follows:

<b><i>Byte</i></b>	<b><i>Meaning</i></b>	<b><i>Example (HEX)</i></b>
1	Slave address	11
2	Function code	04
3	Byte count	06
4	High 8-bit of data 1 (%IW108)	02
5	Low 8-bit of data 1 (%IW108)	2B

6	High 8-bit of data 2 (%IW109)	00
7	Low 8-bit of data 2 (%IW109)	00
8	High 8-bit of data 3 (%IW110)	00
9	Low 8-bit of data 3 (%IW110)	64
CRC16	-	-

In the example, the measuring values of %IW108-%IW110 are 022BH, 0000H and 0064H.

## V. 05 Force single coil

### 1. Description

Forces a single measuring point Q, N or M (0X reference) in the slave to either ON or OFF.

### 2. Master request

- ♦ The request message defines the address and status of measuring point to be forced.
- ♦ Coils are addressed starting at zero: the address of %Q1-%Q128 is 0-127; the address of %M1-%M1024 is 2000-3023; the address of %N1-%N256 is 7000-7255.
- ♦ Force data: A value of FF00H requests the coil to be ON. A value of 0000H requests it to be OFF.

【Example】 Request to force %Q73 ON in slave 17 is as follows:

<i>Byte</i>	<i>Meaning</i>	<i>Example (HEX)</i>
1	Slave address	11
2	Function code	05
3	High 8-bit of start address	00
4	Low 8-bit of start address	48
5	High 8-bit of force data	FF
6	Low 8-bit of force data	00
CRC16	-	-

### 3. Slave response

If right, the slave will return the original request message.

【Example】 Response to the request above is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	05
3	High 8-bit of start address	00
4	Low 8-bit of start address	48
5	High 8-bit of force data	FF
6	Low 8-bit of force data	00
CRC16	-	-

## VI.06 Write single register

### 1. Description

Writes a value into a single measuring point MW, NW or QW (4X reference) in the slave.

### 2. Master request

The request message defines the address and value of measuring point to be written. Registers are addressed starting at zero: the address of %MW1-%MW1024 is 0-1023; the address of %NW1-%NW256 is 5000-5255; and the address of %QW1-%QW32 is 8000-8031.

【Example】 Request to write %MW2 into 3 in slave 17 is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	06
3	High 8-bit of start address	00
4	Low 8-bit of start address	01
5	High 8-bit of write value	00
6	Low 8-bit of write value	03
CRC16	-	-

### 3. Slave response

If right, the slave will return the original request message.

【Example】 Response to the request above is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
-------------	----------------	----------------------

1	Slave address	11
2	Function code	06
3	High 8-bit of start address	00
4	Low 8-bit of start address	01
5	High 8-bit of write value	00
6	Low 8-bit of write value	03
CRC16	-	-

## VII. 15 Force multiple coils

### 1. Description

Forces multiple measuring points Q, N or M (OX reference) in the slave to either ON or OFF.

### 2. Master request

- The request message defines the start address, quantity, and force states of measuring points to be forced.
- Coils are addressed starting at zero: the address of %Q1-%Q128 is 0-127; the address of %M1-%M1024 is 2000-3023; the address of %N1-%N256 is 7000-7255.
- The force status is indicated by bit, and it complies that: the status of start address is indicated by the LSB of the first byte of data, and also complies that it is continuously stored “from low bit to high bit, and from low byte to high byte” by bit.
- If the number of status is not the integral multiple of 8, then fill the extra bits of highest byte with “0”.
- Force status representation: 1 = ON; 0 = OFF.

【 Example 】 In slave 17, forces %Q29-%Q20 to be OFF-ON-ON-ON-OFF-OFF-ON-ON-OFF-ON, the data is CDH and 01H.

Bit:	1	1	0	0	1	1
	0	1		0	0	0
	0	0	0	0	1	
Q: 27	26	25	24	23	22	21
	20		-	-	-	-
	-	-	29	28		

<b><i>Byte</i></b>	<b><i>Meaning</i></b>	<b><i>Example (HEX)</i></b>
1	Slave address	11
2	Function code	0F
3	High 8-bit of start address	00
4	Low 8-bit of start address	13
5	High 8-bit of force number	00
6	Low 8-bit of force number	0A
7	Byte count of force data	02
8	Force data byte 1	CD
9	Force data byte 2	01
CRC16	-	-

### 3. Slave response

If right, the slave will return the start address and quantity of measuring points.

【Example】 Response to the request above is as follows:

<b><i>Byte</i></b>	<b><i>Meaning</i></b>	<b><i>Example (HEX)</i></b>
1	Slave address	11
2	Function code	0F
3	High 8-bit of start address	00
4	Low 8-bit of start address	13
5	High 8-bit of force number	00
6	Low 8-bit of force number	0A
CRC16	-	-

## VIII. 16 Write multiple registers

### 1. Description

Writes values into multiple measuring points MW, NW, QW or system time (4X reference) in the slave.

### 2. Master request

The request message defines the start address, quantity and values of measuring points to be written. Registers are addressed starting at zero: The address of %MW1-%MW1024

is 0-1023; the address of %NW1-%NW256 is 5000-5255; the address of %QW1-%QW32 is 8000-8031; and the address of system time is 9994-9998.

【 Example 】 Request to write %MW2-%MW3 into 000AH and 0102H in slave 17 is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	10
3	High 8-bit of start address	00
4	Low 8-bit of start address	01
5	High 8-bit of register number	00
6	Low 8-bit of register number	02
7	Byte count of write data	04
8	High 8-bit of register 1 data	00
9	Low 8-bit of register 1 data	0A
10	High 8-bit of register 2 data	01
11	Low 8-bit of register 2 data	02
CRC16	-	-

### 3. Slave response

If right, the slave will return the start address and quantity of measuring points.

【 Example 】 Response to the request above is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	10
3	High 8-bit of start address	00
4	Low 8-bit of start address	01
5	High 8-bit of register number	00
6	Low 8-bit of register number	02
CRC16	-	-

### 4. Set the system time

The address of system time is 9994-9998.

【Example】 Request to set the system time of slave 17 to be: Year 2008 (07D8H), month 9 (09H), day 10 (0AH), hour 12 (0CH), minute 13 (0DH), second 14 (0EH), millisecond 0 (0000H).

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	10
3	High 8-bit of start address	27
4	Low 8-bit of start address	0A
5	High 8-bit of register number	00
6	Low 8-bit of register number	05
7	Byte count	0A
8	High 8-bit of year	07
9	Low 8-bit of year	D8
10	Day	0A
11	Month	09
12	High 8-bit of hour	00
13	Low 8-bit of hour	0C
14	Second	0E
15	Minute	0D
16	High 8-bit of millisecond	00
17	Low 8-bit of millisecond	00
CRC16	-	-

【Example】 Response to the request above is as follows:

<b>Byte</b>	<b>Meaning</b>	<b>Example (HEX)</b>
1	Slave address	11
2	Function code	10
3	High 8-bit of start address	27
4	Low 8-bit of start address	0A
5	High 8-bit of register number	00

6	Low 8-bit of register number	05
CRC16	-	-

# **Chapter 12      Connection with Intelligent Touch Screen**

## **12.1    Touch screen application**

With the development of automation technology, more and more industrial processes are remotely controlled by operator, and the human-machine interfaces in local place are less and less. In one hand, the control is convenient, and the cost is reduced, however, the device maintenance is not convenient because the maintainers can't obtain the effective experimental data when inspecting and repairing the device in site and have to turn to the remote operator for help. So the local control device always offers the human-machine interface in site as well as the remote control. Taking into account the harsh environments of industrial site, the human-machine interface always applies the industrial intelligent touch screen to reduce the cost and achieve the local human-machine interaction. Considering that, NA200 PLC offers the touch-screen interface to assure that the link is very easy and convenient.

## **12.2    Relative setup**

There is touch screen interface on the CPU module of NA200 PLC, for the connection method, please refer to the hardware manual of relative product.

**CPU Module Parameter**

**COM1**

Address: 1

Baudrate: 38400

Data Bit: 8

Stop Bit: 1

Parity: None

Protocol: MODBUS RTU

**COM2**

Address: 1

Baudrate: 9600

Data Bit: 8

Stop Bit: 1

Parity: None

Protocol: MODBUS RTU

Start Number (%I): 1

Start Number (%Q): 1

Start Number (%IW): 1

Start Number (%QW): 1

Save Source File to PLC

Create Debug Info

OK Cancel Advanced >>

Fig.12.1 Serial port setup of NA200 PLC

The interface to touch screen applies the COM1 of CPU module. For the serial port communication, it needs to configure its parameters. As described above, in PLC setup, select the CPU module and set the “Baudrate”, “Data Bit”, “Stop Bit”, “Parity” and “Protocol” of COM1.

### 12.3 Data access of touch screen

Take the Pro-face GP series touch screen from Digital company as example, and the following will introduce the data access of the touch screen to NA200 PLC.

### 12.3.1 Serial port setup

For the serial port on the side of touch screen, it also needs to setup the relative parameters and keep the same with the parameters on the side of PLC. If the data is big, the recommended baud rate is 38400. The setting interface of touch screen serial port is as follows, and for the setup method please refers to its operation manual.

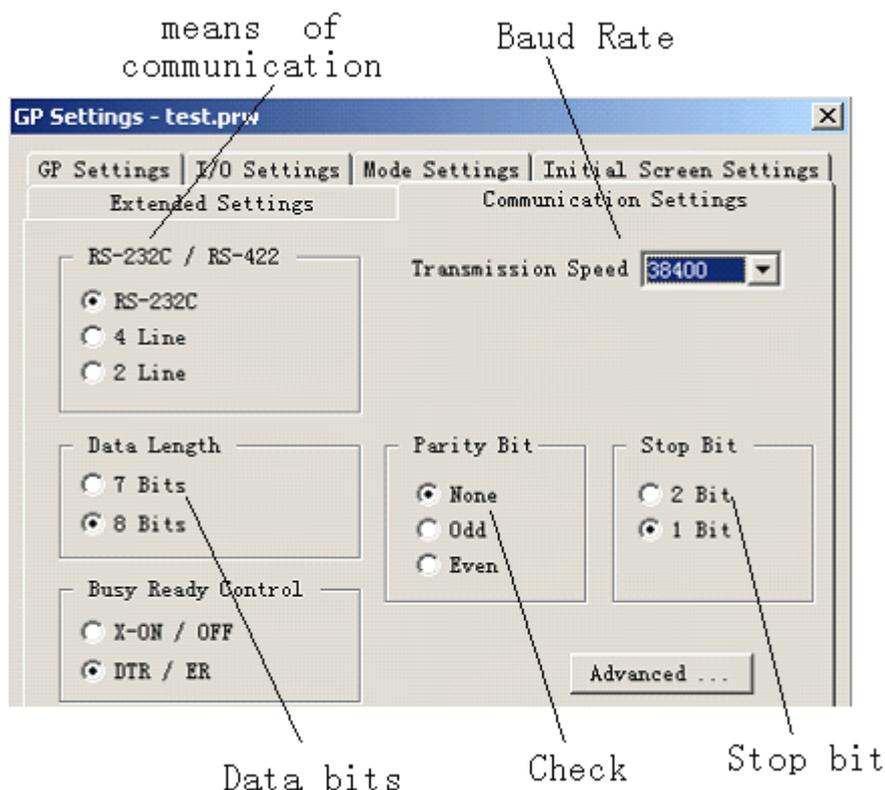


Fig.12.2 Serial port setup of touch screen

### 12.3.2 Communication protocol

The GP series touch screen offers many types of PLC or communication protocols which can be directly connected without editing the relative serial port driver. NA200 PLC supports the standard Modbus-RTU communication protocol which can be directly connected with this series of touch screen.

In the GP series touch screen software, the PLC type or communication protocol can be selected by "Change PLC Type". For NA200 PLC, select "Modicon Modbus (MASTER)".

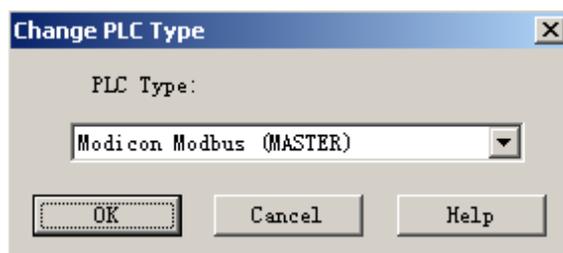


Fig.12.3 Change PLC type

For other touch screens supporting Modbus protocol, it also can be directly connected with the NA200 PLC.

### 12.3.3 Register access

By the method of register accessing, the PLC point can be directly accessed.

Point type	Point address range	Classification prefix	Protocol address range
Q	0001~0128	0x	00001~00128
M	0001~1024		02001~03024
N	0001~0256		07001~07256
I	0001~0128	1x	00001~00128
S	0001~0256		02001~02256
IW	0001~0032	3x	00001~00032
SW	0001~0256		02001~02256
MW	0001~1024	4x	00001~01024
NW	0001~0256		05001~05256
QW	0001~0032		08001~08032
Clock			09995~09998

The above table is the comparison table of the NA200 PLC CPU points with the GP touch screen addresses. That is, the PLC points correspond to the touch screen addresses, by accessing to the touch screen addresses, and the PLC points can be accessed. For example, if want to read the point value of “%I0001”, the address in touch screen is “100001”.

### 12.3.4 LS access

Applying the register to access is simple and intuitive, however, it will affect the screen

refresh rate, especially when a single screen accesses the much more points, and the efficiency of reading data is lower. For this case, GP touch screen offers a kind of LS access mode. In this mode, the touch screen can read more data in the designated storage area of PLC each time, if firstly store the desired data in this area, the screen refresh rate will be much faster when reading.

The configuration of LS access area can be set in the GP software. Select the “GP Setup->Mode Settings”, the setting interface is as shown in the following:

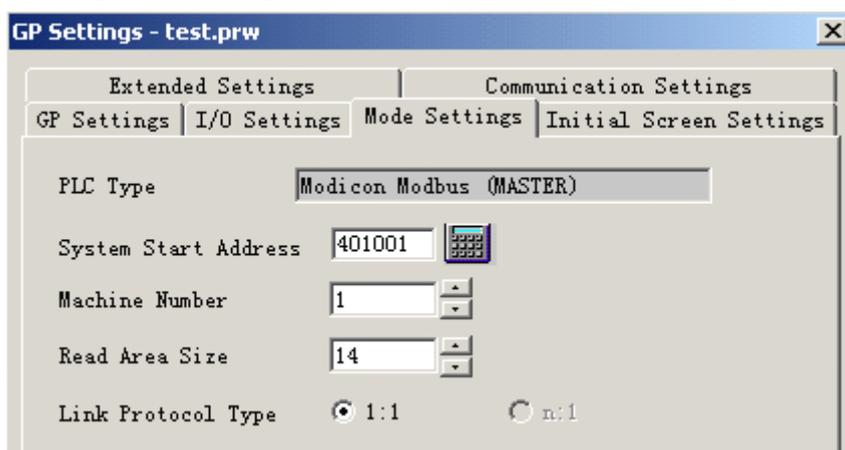
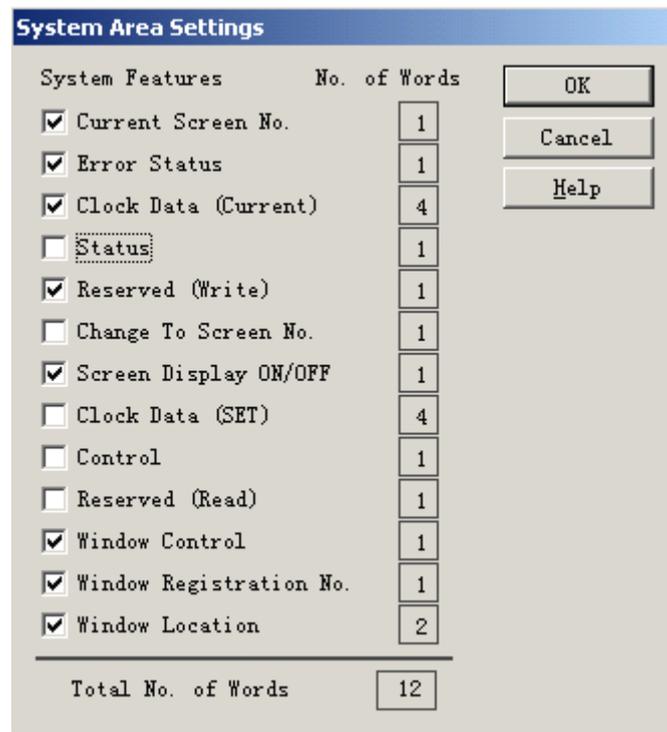


Fig.12.4 LS access

Wherein, the “PLC Type” shows that the current type is Modicon Modbus (MASTER); the “System Start Address” is the start address of access area, and this address is touch screen address, searching in the comparison table above, 401001 is correspond to the point “%MW1001”; the “Machine Number” is COMx address; the “Read Area Size” is the size of reading data area, the maximum is 256. The configurations in above Fig., it is indicated that GP will read data of “%MW1001” to “%MW1014” from PLC to touch screen. In touch screen, these points are assigned to new address LS, for example, when reading point “%MW1001”, the input address is “LS0”; the LS address supports the bit access, each LS address is a word, including 16 bits of 00~15, when accessing, only need to add two bits of bit address behind the corresponding word address. For example, when accessing to the sixteenth bit of “%MW1002”, only need to input address “LS15”, wherein “LS1” is correspond to “%MW1002”, “15” means the sixteenth bit. When using LS address access, the screen refresh rate will be faster.

By using LS address, the corresponding point value only can be read from PLC, but writing value into PLC is not possible. The GP has opened 20 system data areas of “LS0~LS19”, these parts of addresses have been assigned the special usages by GP, such as system time, display screen NO., etc. These parts of addresses can be read and written. For the unnecessary system data, when using, they can be not selected, in this way, the system data area will be decreased, but the user LS data area will be increased, and the total LS data keep the same. In GP “Extended Settings” tab, there is a “System

Area Settings” button to select the required system data.



The dialog box titled "System Area Settings" contains a table with two columns: "System Features" and "No. of Words". Each row in the table has a checkbox in the "System Features" column and a text input field in the "No. of Words" column. The features listed are: Current Screen No., Error Status, Clock Data (Current), Status, Reserved (Write), Change To Screen No., Screen Display ON/OFF, Clock Data (SET), Control, Reserved (Read), Window Control, Window Registration No., and Window Location. The "Status" checkbox is unchecked, while all others are checked. The "No. of Words" values are: 1, 1, 4, 1, 1, 1, 1, 4, 1, 1, 1, 1, and 2. At the bottom of the table, there is a row for "Total No. of Words" with a value of 12. To the right of the table are three buttons: "OK", "Cancel", and "Help".

System Features	No. of Words
<input checked="" type="checkbox"/> Current Screen No.	1
<input checked="" type="checkbox"/> Error Status	1
<input checked="" type="checkbox"/> Clock Data (Current)	4
<input type="checkbox"/> Status	1
<input checked="" type="checkbox"/> Reserved (Write)	1
<input type="checkbox"/> Change To Screen No.	1
<input checked="" type="checkbox"/> Screen Display ON/OFF	1
<input type="checkbox"/> Clock Data (SET)	4
<input type="checkbox"/> Control	1
<input type="checkbox"/> Reserved (Read)	1
<input checked="" type="checkbox"/> Window Control	1
<input checked="" type="checkbox"/> Window Registration No.	1
<input checked="" type="checkbox"/> Window Location	2
<hr/>	
Total No. of Words	12

Fig.12.5 System area settings

As shown above , the left column shows the system data contents, each item is optional; the right column shows the number of words occupied by system data. At the bottom, it shows the total word number of system data selected. All the system data are selected, and then the word number is 20.