

K5 series PLC



SOFTWARE OPERATION **MANUAL**

Contents

CONTENTS	1
CHAPTER I WELCOME TO USE KINCOBUILDER.....	10
1.1 OVERVIEW	10
1.2 GENERAL DESIGNATION IN THE MANUAL	11
CHAPTER II HOW TO USE KINCOBUILDER ... A QUICK GUIDE.....	14
2.1 COMPUTER REQUIREMENTS	14
2.1.1 <i>Minimum hardware requirements to run KincoBuilder:</i>	14
2.1.2 <i>Minimum Software requirements to run KincoBuilder:</i>	14
2.2 USER INTERFACE OF KINCOBUILDER	16
2.3 USING KINCOBUILDER TO CREATE PROGRAMS FOR YOUR APPLICATIONS	18
2.3.1 <i>Project Components</i>	18
2.3.2 <i>Where to store the Project Files</i>	19
2.3.3 <i>Importing and Exporting a Project</i>	19
2.4 HOW THE CPU EXECUTES ITS TASKS IN A SCAN CYCLE?	22
2.5 HOW TO CONNECT THE COMPUTER WITH THE KINCO-K5.....	24
2.6 HOW TO MODIFY THE CPU'S COMMUNICATION PARAMETERS.....	27
2.7 EXAMPLE: COMMON STEPS TO CREATE A PROJECT	28
CHAPTER III CONCEPTS FOR PROGRAMMING	36
3.1 POU (PROGRAMME ORGNIZATION UNIT).....	36
3.2 DATA TYPES	38
3.3 IDENTIFIERS	40
3.3.1 <i>How to define an identifier</i>	40

3.3.2 Use of Identifiers.....	40
3.4 CONSTANT.....	41
3.5 VARIABLES	43
3.5.1 Declaration.....	43
3.5.2 Declaring Variables in KincoBuilder.....	44
3.5.3 Checking Variables	44
3.6 HOW TO ACCESS PLC MEMORY.....	45
3.6.1 Memory Types and Characteristics.....	45
3.6.2 Direct Addressing.....	47
3.6.3 Indirect Addressing	54
3.6.4 Memory Address Ranges.....	56
3.6.5 Function Block and Function Block Instance	58
3.6.6 Using FB Instances.....	60
3.6.7 FB Instances Memory Ranges.....	62
CHAPTER IV HOW TO USE KINCOBUILDER ... BASIC FUNCTIONS.....	63
4.1 CONFIGURING GENERAL SOFTWARE OPTIONS	63
4.2 ABOUT DOCKING WINDOWS	65
4.3 CONFIGURING HARDWARE.....	66
4.3.1 How to open the Hardware window.....	67
4.3.2 Copy and paste the hardware configuration in different projects.....	67
4.3.3 Add/Remove Modules.....	67
4.3.4 Configuring Module Parameters	68
4.4 THE INITIAL DATA TABLE.....	77
4.4.1 Opening the Initial Data Table.....	77
4.4.2 Editing a Cell.....	77
4.4.3 Making Initial Data Assignments.....	78

4.4.4 <i>Editing the Initial Data Table</i>	78
4.5 THE GLOBAL VARIABLE TABLE.....	79
4.5.1 <i>Opening the Global Variable Table</i>	80
4.5.2 <i>Declaring the Global Variables</i>	80
4.6 THE CROSS REFERENCE TABLE.....	82
4.6.1 <i>Opening the Cross Reference Table</i>	82
4.6.2 <i>The Pop-up Menu</i>	83
4.7 THE STATUS CHART.....	84
4.7.1 <i>Opening the Status Chart</i>	86
4.7.2 <i>Monitoring the Variable Value</i>	86
4.7.3 <i>The Force Function</i>	86
4.7.4 <i>Right-click Menu</i>	87
4.7.5 <i>Force and Cancel Force</i>	87
4.8 PASSWORD PROTECTION.....	89
4.8.1 <i>Protection Privileges</i>	89
4.8.2 <i>How to change the password and the protection level</i>	89
4.8.3 <i>How to recover from a lost password</i>	91
CHAPTER V HOW TO USE KINCOBUILDER ... PROGRAMMING.....	92
5.1 PROGRAMMING IN IL.....	92
5.1.1 <i>Overview</i>	92
5.1.2 <i>Rules</i>	93
5.1.3 <i>The IL Editor in KincoBuilder</i>	95
5.1.4 <i>Converting IL Program to LD Program</i>	98
5.1.5 <i>Debug and Monitor the Program</i>	98
5.2 PROGRAMMING IN LD.....	101
5.2.1 <i>Overview</i>	101

5.2.2 Network.....	101
5.2.3 Standardized graphic symbols.....	101
5.2.4 The LD Editor in KincoBuilder.....	104
5.2.5 Monitoring and Debugging the Program.....	111
CHAPTER VI KINCO-K5 INSTRUCTION SET.....	114
6.1 SUMMARY.....	114
6.2 BIT LOGIC INSTRUCTIONS.....	115
6.2.1 Standard Contact.....	115
6.2.2 Immediate Contact.....	119
6.2.3 Coil.....	121
6.2.4 Immediate Coil.....	123
6.2.5 Set And Reset Coil.....	124
6.2.6 Set and Reset Block Coil.....	126
6.2.7 Set And Reset Immediate Coil.....	128
6.2.8 Edge detection.....	129
6.2.9 NCR (NOT).....	131
6.2.10 Bistable elements.....	132
6.2.11 ALT (Alternate).....	135
6.2.12 NOP (No Operation).....	137
6.2.13 Bracket Modifier.....	138
6.3 MOVE INSTRUCTIONS.....	140
6.3.1 MOVE.....	140
6.3.2 BLKMOVE (Block Move).....	141
6.3.3 FILL (Memory Fill).....	144
6.3.4 SWAP.....	145
6.4 COMPARE INSTRUCTIONS.....	148

6.4.1	<i>GT (Greater Than)</i>	148
6.4.2	<i>GE (Greater than or Equal to)</i>	150
6.4.3	<i>EQ (Equal to)</i>	152
6.4.4	<i>NE (Not Equal to)</i>	154
6.4.5	<i>LT (Less than)</i>	156
6.4.6	<i>LE (Less than or Equal to)</i>	158
6.5	LOGICAL OPERATIONS	160
6.5.1	<i>NOT</i>	160
6.5.2	<i>AND</i>	162
6.5.3	<i>ANDN</i>	164
6.5.4	<i>OR</i>	166
6.5.5	<i>ORN</i>	168
6.5.6	<i>XOR (Exclusive OR)</i>	170
6.6	SHIFT/ROTATE INSTRUCTIONS	172
6.6.1	<i>SHL (Shift left)</i>	172
6.6.2	<i>ROL (Rotate left)</i>	174
6.6.3	<i>SHR (Shift right)</i>	176
6.6.4	<i>ROR (Rotate right)</i>	178
6.6.5	<i>SHL_BLK (Bit String Shift Left)</i>	180
6.6.6	<i>SHR_BLK (Bit String Shift Right)</i>	182
6.7	CONVERT INSTRUCTIONS	185
6.7.1	<i>DI_TO_R (DINT To REAL)</i>	185
6.7.2	<i>R_TO_DI (REAL To DINT)</i>	186
6.7.3	<i>B_TO_I (BYTE To INT)</i>	188
6.7.4	<i>I_TO_B (INT To BYTE)</i>	189
6.7.5	<i>DI_TO_I (DINT To INT)</i>	192
6.7.6	<i>I_TO_DI (INT To DINT)</i>	194

6.7.7	<i>BCD_TO_I (BCD To INT)</i>	195
6.7.8	<i>I_TO_BCD (INT To BCD)</i>	196
6.7.9	<i>I_TO_A (INT To ASCII)</i>	199
6.7.10	<i>DI_TO_A (DINT To ASCII)</i>	201
6.7.11	<i>R_TO_A (REAL To ASCII)</i>	203
6.7.12	<i>H_TO_A (Hexadecimal To ASCII)</i>	206
6.7.13	<i>A_TO_H (ASCII to Hexadecimal)</i>	208
6.7.14	<i>ENCO (Encoding)</i>	210
6.7.15	<i>DECO (Decoding)</i>	212
6.7.16	<i>SEG (7-segment Display)</i>	214
6.7.17	<i>TRUNC (Truncate)</i>	215
6.8	NUMERIC INSTRUCTIONS	217
6.8.1	<i>ADD and SUB</i>	217
6.8.2	<i>MUL and DIV</i>	219
6.8.3	<i>MOD (Modulo-Division)</i>	221
6.8.4	<i>INC and DEC</i>	223
6.8.5	<i>ABS (Absolute Value)</i>	225
6.8.6	<i>SQRT (Square Root)</i>	226
6.8.7	<i>LN (Natural Logarithm), LOG (Common Logarithm)</i>	227
6.8.8	<i>EXP (Exponent with the base e)</i>	228
6.8.9	<i>SIN (sine), COS (cosine), TAN (tangent)</i>	229
6.8.10	<i>ASIN (arc-sine), ACOS (arc-cosine), ATAN (arc-tangent)</i>	231
6.9	PROGRAM CONTROL.....	233
6.9.1	<i>LBL and JMP Instructions</i>	233
6.9.2	<i>Return Instructions</i>	235
6.9.3	<i>CAL (Call a subroutine)</i>	237
6.9.4	<i>FOR/NEXT (FOR/NEXT Loop)</i>	240

6.9.5	<i>END (Terminate the scan cycle)</i>	243
6.9.6	<i>STOP (Stop the CPU)</i>	244
6.9.7	<i>WDR (Watchdog Reset)</i>	245
6.10	INTERRUPT INSTRUCTIONS	246
6.10.1	<i>How the Kinco-K5 handles Interrupt Routines</i>	246
6.10.2	<i>Interrupt Priority and Queue</i>	246
6.10.3	<i>Types of Interrupt Events Supported by the Kinco-K5</i>	246
6.10.4	<i>Interrupt Events List</i>	247
6.10.5	<i>ENI (Enable Interrupt), DISI (Disable Interrupt)</i>	248
6.10.6	<i>ATCH and DTCH Instructions</i>	249
6.11	CLOCK INSTRUCTIONS	253
6.11.1	<i>Adjusting the RTC online</i>	253
6.11.2	<i>READ_RTC and SET_RTC</i>	254
6.11.3	<i>RTC_R</i>	256
6.11.4	<i>RTC_W</i>	258
6.12	COMMUNICATION INSTRUCTIONS	261
6.12.1	<i>Free-protocol Communication</i>	261
6.12.2	<i>XMT and RCV</i>	261
6.12.3	<i>Modbus RTU Master Instructions</i>	271
6.12.4	<i>SDO</i>	277
6.12.5	<i>CAN Communication Command</i>	283
6.13	COUNTERS.....	292
6.13.1	<i>CTU (Up Counter) and CTD (Down Counter)</i>	292
6.13.2	<i>CTUD (Up-Down Counter)</i>	295
6.13.3	<i>High-speed Counter Instructions</i>	297
6.13.4	<i>High-speed Pulse Output Instructions</i>	313
6.14	TIMERS	329

6.14.1	The resolution of the timer	329
6.14.2	TON (On-delay Timer)	329
6.14.3	TOF (Off-delay Timer)	331
6.14.4	TP (Pulse Timer)	333
6.15	PID.....	336
6.16	POSITION CONTROL.....	341
6.16.1	Model	341
6.16.2	The correlative variables.....	342
6.16.3	PHOME (Homing).....	344
6.16.4	PABS (Moving Absolutely)	347
6.16.5	PREL (Moving Relatively)	349
6.16.6	PJOG (Jog).....	353
6.16.7	PSTOP (Stop).....	355
6.16.8	PFLO_F.....	356
6.16.9	Examples.....	358
6.17	ADDITIONAL INSTRUCTIONS	369
6.17.1	LINCO (Linear Calculation).....	369
6.17.2	CRC16 (16-Bit CRC)	371
6.17.3	SPD (Speed detection)	373
APPENDIX A	COMMUNICATE USING MODBUS RTU PROTOCOL	375
1.	PLC MEMORY AREA	375
1.1	Accessible Memory Areas	375
1.2	Modbus Register Number.....	375
2.	BASIC REPORT FORMAT OF MODBUS RTU	378
2.1	Modbus RTU.....	378
2.2	CRC Verification Algorithm in Modbus Protocol.....	381

APPENDIX B	DYNAMIC MODIFICATION PARAMETER OF RS485 COMMUNICATION PORT	385
1.	GENERAL DESCRIPTION	385
2.	REGISTER INSTRUCTION	386
3.	INSTRUCTIONS.....	389
4.	EXAMPLE	391
APPENDIX C	DATA BACKUP	393
APPENDIX D	ERROR DIAGNOSE	394
1.	ERROR LEVEL	394
2.	ERROR CODES.....	396
3.	HOW TO READ ERRORS OCCUR BEFORE.....	399
4.	ERROR REGISTER	401
APPENDIX E	DEFINITION OF SM AREA	404
1.	SMB0: SYSTEM STATUS BYTE	404
2.	SMB2: SYSTEM CONTROL BYTE	405
3.	COMMUNICATION PORT RESET.....	406
4.	OTHER FUNCTIONAL VARIABLES.....	408
5.	SMD12 AND SMD16: TIMER INTERRUPTION CYCLE.....	409

Chapter I Welcome to Use KincoBuilder

1.1 Overview

IEC61131-3 is the only global standard for industrial control programming. Its technical implications are high, leaving enough room for growth and differentiation. It harmonizes the way people design and operate industrial controls by standardizing the programming interface. IEC 61131-3 has a great impact on the industrial control industry, and it is accepted as a guideline by most PLC manufacturers. With its far-reaching support, it is independent of any single company.

KincoBuilder is the programming software for Kinco-K5 series Micro PLC, and it's a user-friendly and high-efficient development system with powerful functions.

KincoBuilder is developed independently and accords with the IEC61131-3 standard. It becomes easy to learn and use because many users have acquired most of the programming skills through different channels.

KincoBuilder is provided with the following features:

- Accords with the IEC61131-3 standard
- Supports two standard programming languages, i.e. IL (Instruction List) and LD (Ladder Diagram)
- Powerful instruction set, build-in standard functions, function blocks and other special instructions
- Supports structured programming
- Supports interrupt service routines
- Supports subroutines
- Supports direct represented variables and symbolic variables, easy to develop and manage the user project.

- User-friendly and high-efficient environment
- Flexible hardware configuration, you can define all types of the hardware parameters

1.2 General Designation in the Manual

▪ Micro PLC (Programmable Logic Controller)

According to the general classification rules, micro PLC generally refers to the type of PLC with the control points below 128. This type of PLC usually adopts compact structure, that is, a certain number of I/O channels, output power supply, high-speed output/input and other accessories are integrated on the CPU module.

▪ CPU body

Namely, the CPU module, it's the core of the control system. The user program is stored in the internal storage of the CPU module after being downloaded through the programming software, and will be executed by the CPU. Meanwhile, it also executes the CPU self-test diagnostics: checks for proper operation of the CPU, for memory areas, and for the status of any expansion modules.

▪ Expansion module & expansion bus

The expansion module is used to extend the functions of the CPU body and it is divided into expansion I/O module (to increase the input/output channels of the system) and expansion functional module (to expand the functions of CPU).

The expansion bus connects the CPU and expansion modules, and the 16-core flat cable is adopted as the physical media. The data bus, address bus and the expansion module's working power supply are integrated into the expansion bus.

▪ KincoBuilder

The programming software for Kinco-K5 series PLC, accords with IEC61131-3 standard KincoBuilder,

presently provides LD and IL languages for convenience and efficiency in developing the control programs for your applications. KincoBuilder provides a user-friendly environment to develop and debug the programs needed to control your applications.

- CPU firmware

It is the “operating system” of the CPU module, and is stored in the Flash memory. At power on, it starts operation to manage and schedule all the tasks of the CPU module.

- User program

It’s also called user project or application program, the program written by the user to execute some specific control functions. After the user program is downloaded to the CPU module, it is stored in the FRAM. At power on, the CPU module shall read it from FRAM into RAM to execute it.

- Main program and Scan Cycle

The CPU module executes a series of tasks continuously and cyclically, and we call this cyclical execution of tasks as *scan*.

The main program is the execution entry of the user program. In the CPU, the main program is executed once per scan cycle. Only one main program is allowed in the user program.

- Free-protocol communication

The CPU body provides serial communication ports that support the special programming protocol, Modbus RTU protocol (as a slave) and free protocols. Free-protocol communication mode allows your program to fully control the communication ports of the CPU. You can use free-protocol communication mode to implement user-defined communication protocols to communicate with all kinds of intelligent devices. ASCII and binary protocols are both supported.

- I/O Image Area

Including input image area and output image area. At the beginning of a scan cycle, signal status are transferred from input channels to the input image area; at the end of a scan cycle, the values stored in the output image area are transferred to output channels;

In order to ensure the consistency of data and to accelerate the program execution, the CPU module only access the image area during each scan cycle.

- Retentive Ranges

Through “Hardware” configuration in KincoBuilder, you can define four retentive ranges to select the areas of the RAM you want to retain on power loss. In the event that the CPU loses power, the instantaneous data in the RAM will be maintained by the super capacitor, and on the retentive ranges will be left unchanged at next power on. The retaining duration is 72 hours at normal temperature.

- Data backup

Data backup is the activity that you write some data into E²PROM or FRAM through relevant instruction for permanent storage. *Notice: Every type of permanent memory has its own expected life, for example, E²PROM allows 100 thousand of times of writing operations and FRAM allows 10 billions of times.*

Chapter II How to Use KincoBuilder ... A Quick Guide

In this chapter, you will learn how to install KincoBuilder on your computer and how to program, connect and run your Kinco-K5 PLC. The purpose of this chapter is to give you a quick guide, and further details will be presented in the following chapter.

2.1 Computer Requirements

2.1.1 Minimum hardware requirements to run KincoBuilder:

- CPU: 1 GHz or higher
- Hard disk space: at least 20M bytes of free space
- RAM: 512M or more
- Keyboard, mouse, a serial communication port
- 256-color VGA or higher, 1024*768 or higher

2.1.2 Minimum Software requirements to run KincoBuilder:

- Operating system: Windows XP(32bit), Windows Vista(32bit), Windows7(32/64bit), Windows8 (32/64bit), Windows 8.1(32/64bit)
- Users may find errors when running KincoBuilder on OS of Windows 7 or above. Possible solutions are as follows:
- [COM] port in the Communication setting is null

KincoBuilder detects available COM port on a computer by reading the hardware information in the REGEDIT. In previous versions, KincoBuilder requires authorities from the Administrators to run; otherwise it shows null port lists.

In the latest version, KincoBuilder will automatically detect branches. If KincoBuilder is not given authority to read port list, it will list ports from COM1 to COM9 for user to choose manually.

- KincoBuilder unable to run on computers

Users may turn to Compatibility Mode to run KincoBuilder and set as follows:

- Right click the shortcut of “KincoBuilder V1.5.x.x” and click [Properties];
- Click [Compatibility] in [Properties] dialogues, as shown in figure 2-1

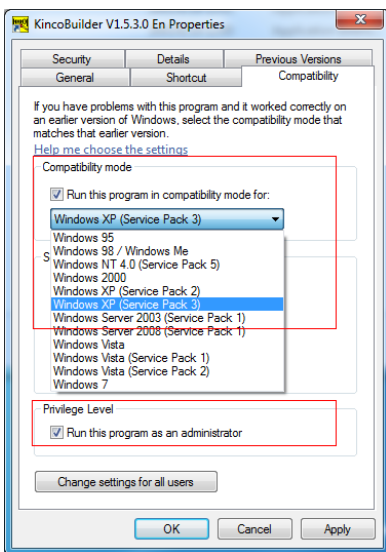


Figure 2-1 “Compatibility Mode” setting

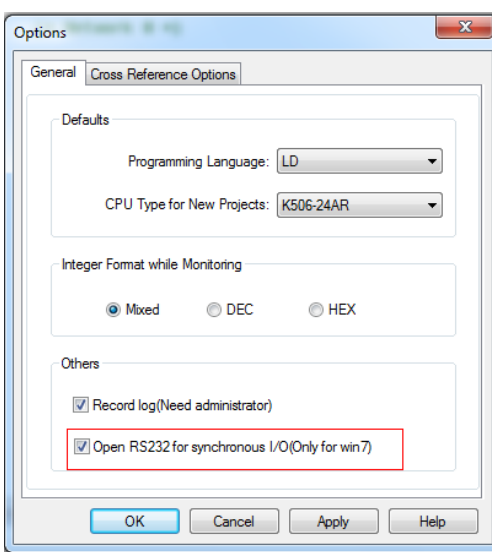


Figure 2-2 Open with Sync

- Fail when using USB to RS232 convertor to communicate with PLC

The failure is caused by the driver programme of the convertor not compatible with the computer. Most of cases are caused with 64-bit Windows 7.

Open KincoBuilder and find [Tool] → [Software Setting] → [Open Port Parallel], click “Open Port with Sync” and click “OK”. See figure 2-2.

Afterwards KincoBuilder will open ports with sync and in most cases will successfully work.

2.2 User Interface of KincoBuilder

The user interface uses standard Windows interface functionality along with a few additional features to make your development environment easy to use.

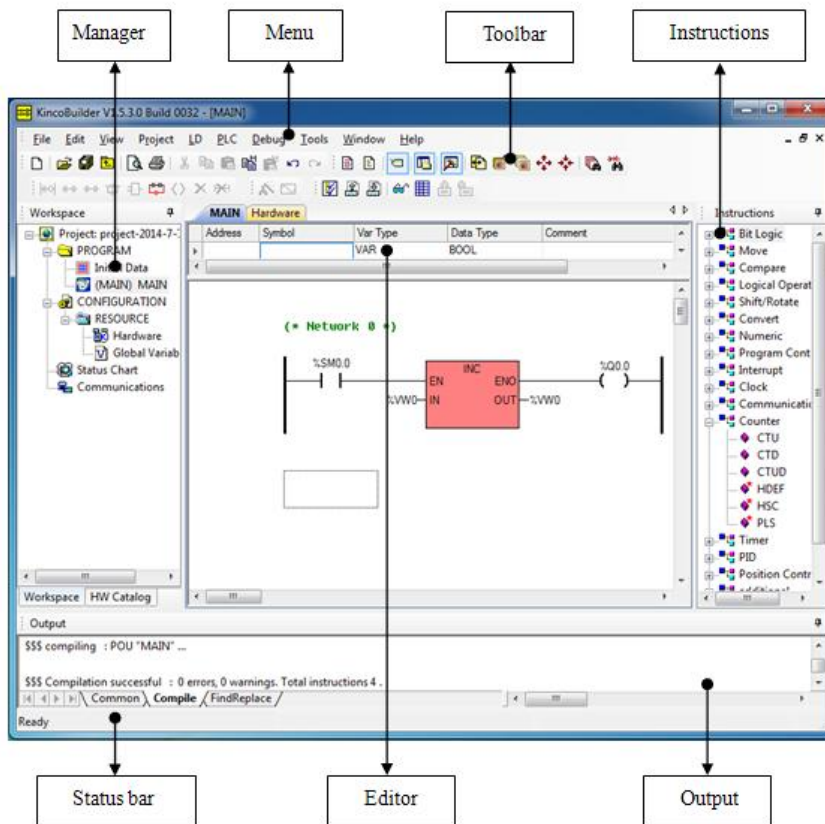


Figure 2-1 User Interface of KincoBuilder

- **Menu:** It contains all the operation commands in KincoBuilder.
- **Toolbar:** It provides easy mouse access to the most frequently used operation commands.
- **Statusbar:** It provides status information and prompts for the operations.
- **Manager:** The Manager window provides a tree view of all project objects, including *PROGRAM*, *Hardware*,

Global Variable, etc, and this can assist you in understanding the structure of the project. The project manager is a convenient tool for program organization and project management. A context menu will pop up when you right click on any tree node.

- **Editor:** It includes the Variable Table and the Program Editor (IL or LD). You can programming in the Program Editor and declare the local variables and input/output parameters of the POU in the Variable Table.
- **Instructions:** LD instruction set and IL instruction set. Here a tree view of all the available instructions is provided.
- **Output:** The Output Window displays several types of information. Select the tab at the base of the window to view the respective information: the “**Compile**” window displays the latest compiling information and the “**Common**” window displays some information concerning the latest operations.

2.3 Using KincoBuilder to Create Programs for Your Applications

2.3.1 Project Components

In this manual, a *user program* and a *user project* have the same meaning.

While programming for a specific application, you need to configure the controllers used in your control system, define symbolic variables and write all kinds of POU's, etc. In KincoBuilder, all of these data (including POU's, hardware configuration, global variables, etc.) are organized to structure a user project. You can manage the project information consistently and easily.

The components of a project are described in the following table. The items marked with "Optional" are not essential components in the project, so you can ignore them.

PROGRAM	Initial Data (Optional)	<p>You can assign initial numerical values to BYTE, WORD, DWORD, INT, DINT and REAL variables in V area.</p> <p>The CPU module processes the Initial Data once at power on and then starts the scan cycle.</p>
	Main Program	<p>It is the execution entry of the user program.</p> <p>The CPU module executes it once per scan cycle.</p> <p>Only 1 Main Program exists in a project.</p>
	Interrupt routines (Optional)	<p>They are interrupt service routines used to process the specific interrupt events. They are not invoked by the main program. You attach an interrupt routine to a predefined interrupt event, and the CPU module executes this routine only on each occurrence of the interrupt event.</p> <p>At most 16 interrupt routines are allowed in a project.</p>

	Subroutines (Optional)	<p>The subroutines can only be executed when they are invoked by the main program or interrupt routines.</p> <p>Subroutines are helpful to better structure the user program. They are reusable, and you can write the control logic once in a subroutine and invoke it as many times as needed. Formal input/output parameters can be used in the subroutines.</p> <p>At most 16 subroutines are allowable in a project.</p>
CONFIGURATION	Hardware	<p>Here you can configure the KINCO-K5 modules used in your control system, including their addresses, function parameters, etc.</p> <p>The CPU module shall process the hardware configuration once at power on and then execute other tasks.</p>
	Global variables (Optional)	<p>Here you can declare the global variables required in the project.</p>

Table 2-1 Project Components

2.3.2 Where to store the Project Files

When creating a project, KincoBuilder firstly ask you to specify a full path for the project file, and then an empty project file (with the ".kpr" extension) shall be created and saved in this path. In addition, a folder with the same name as the project shall be also created in this path; this folder is used to store all the program files, variable files and other temporary files of the project.

For example, if you create a project named "example" in "c:\temp" directory, the project file path is "c:\temp\example.kpr", and other files are stored in the "c:\temp\example" folder.

2.3.3 Importing and Exporting a Project

KincoBuilder provides **[File]>[Import...]** and **[File]>[Export...]** menu commands for you to backup and manage a project.

➤ **[Export...]**

Compress all the files related to the current project into one backup file (with the “.zip” extension).

- Select the **[File]> [Export...]** menu command.

The dialog box “Export Project...” appears, as shown in Figure 2-2.

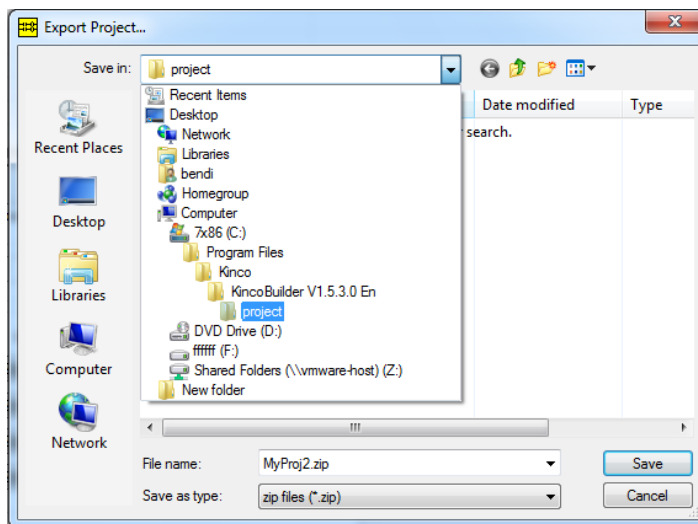


Figure 2-2 Export the Project

- Select the path and enter the filename, then click **[Save]**.

The backup file for the current project shall be created.

- **[Import...]**

Import a project from an existing backup file (with the extension .zip) and open it.

- Select the **[File]> [Import...]** menu command.

The dialog box “Import Project...” appears, as shown in Figure 2-3.

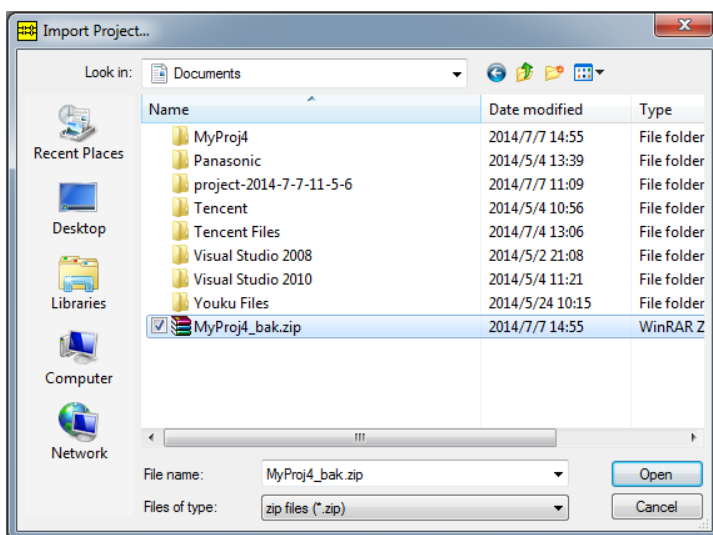


Figure 2-3 Import a Project: Select a backup file

- Select a backup file, and then click [**Open**].

The following dialog box appears for you to select the directory to save the restored project files.

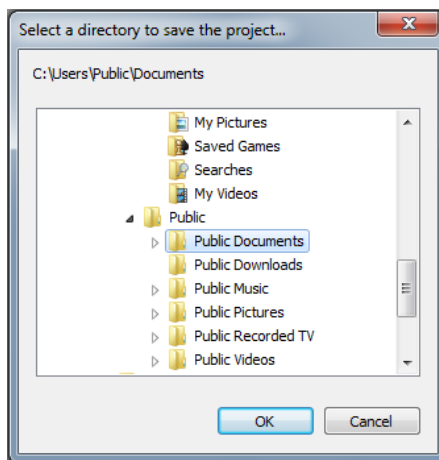


Figure 2-4 Import a Project: Select the destination directory

- Select a directory, then click [**OK**], and the project files shall be restored into the selected directory, with that the restored project shall be opened.

2.4 How The CPU Executes Its Tasks in a Scan Cycle?

The CPU module executes a series of tasks continuously and cyclically, and we call this cyclical execution of tasks as *scan*. Only can the main program and interrupt routines be executed directly in the CPU module. The main program is executed once per scan cycle; an interrupt routine is executed once only on each occurrence of the interrupt event associated with it.

The CPU module executes the following tasks in a scan cycle, as shown in Figure 2-11:

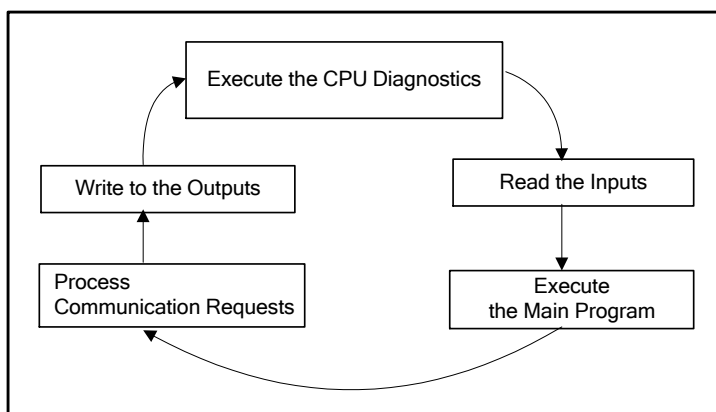


Figure 2-5 Scan Cycle

- Executing the CPU diagnostics: The CPU module executes the self-test diagnostics to check for proper operation of the CPU, for memory areas, and for the status of the expansion modules.
- Read the inputs: The Kinco-K5 samples all the physical input channels and writes these values to the input image areas.
- Executing the user program: The CPU module executes the instructions in the main program continuously and updates the memory areas.
- Processing communication requests
- Writing to the outputs: The Kinco-K5 writes the values stored in the output areas to the physical output channels.

Interrupt events may occur at any moment during a scan cycle. If you use interrupts, the CPU module will

interrupt the scan cycle temporarily when the interrupt events occur and immediately execute the corresponding interrupt routines. Once the interrupt routine has been completed, control is returned to the scan cycle at the breakpoint.

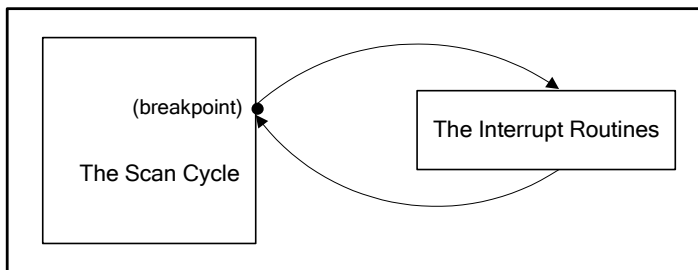


Figure 2-6 Execution of Interrupt Routines

2.5 How to connect the computer with the Kinco-K5

The CPU module provides an integrated RS232 or RS485 serial communication port to communicate with other equipments. Here we discuss how to connect a CPU module (with RS232 port) with the computer to start programming the Kinco-K5 PLC using KincoBuilder.

- Launch KincoBuilder, open an existing project or create a new project;

Connect the serial port of the computer with that of the CPU module with a proper programming cable.

Notice: RS232 connections are not hot-swappable, so you must switch off the power supply for at least one side (the CPU module or the computer) before you connect/disconnect the cable. Otherwise, the port may be damaged.

- ② Configure the parameters of the computer's serial communication port. *Notice: Communications can't be established unless the serial communication parameters of the computer's port are identical with those of the CPU's port.*

- Select [Tools]>[Communications...] menu command, or double-click the [Communications] node in the **Manager** window, or right-click the [Communications] node and select the [Open] command on the pop-up menu, then the "Communications" dialog box appears.

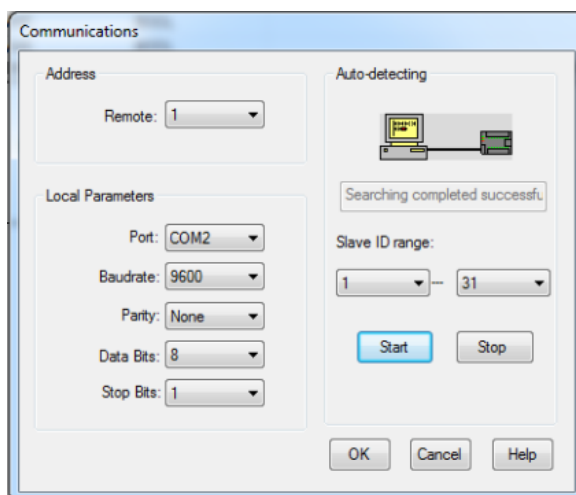


Figure 2-7 The “Communications ” Dialog Box

- Select the station number of the target PLC in the [**Remote**] list box; Select a COM port used on the computer in the [**Port**] list box; Configure the parameters of the selected COM port (including [**Baudrate**], [**Parity**], [**Data Bits**] and [**Stop Bits**]) according to those of the CPU’s port, and then click [**OK**] button to save and apply them.

If you don’t know the communication parameters of the CPU’s port, how to acquire them?

There are two ways:

- Select a [**Port**] used on the computer, then click [**Search**] button to make KincoBuilder search for the parameters of the online CPU module automatically. It shall take several seconds to several minutes to complete. If the search completes successfully, KincoBuilder will automatically configure the appropriate parameters for the computer.
- Turn off the power supply for the CPU module; Place its operation switch at STOP position; Then turn the power supply on, and now the CPU’s port will use the default serial communication parameters: Station number, 1; Baudrate, 19200; None parity check; 8 data bits; 1 stop bit. You can configure the computer’s

serial COM port according to these parameters. *Notice: Do not change the switch's position until you have modified the CPU's communication parameters.*

- After you have configured the communication parameters of the computer's COM port, you are ready to program the Kinco-K5 PLC.

2.6 How to modify the CPU's communication parameters

After you have connected a CPU module with the computer, you can modify its communication parameters at will using KincoBuilder.

- First, open the “Hardware” window by using one of the following ways:
 - Double-click the [**Hardware**] node in the **Manager** window;
 - Right-click the [**Hardware**] node, and then select the [**Open...**] command on the pop-up menu.

The upper part of the hardware window shows a detailed list of the PLC modules in table form, and we call it Configuration Table. The Configuration Table represents the real configuration; you arrange your modules in the Configuration Table just as you do in a real control system.

The lower part of the hardware window shows all the parameters of the selected module in the Configuration Table, and we call it Parameters Window.

- Select the CPU module in the Configuration Table, and then select the [**Communication Ports**] tab in the Parameters Window. Now, you can modify the communication parameters here, as shown in the following figure.

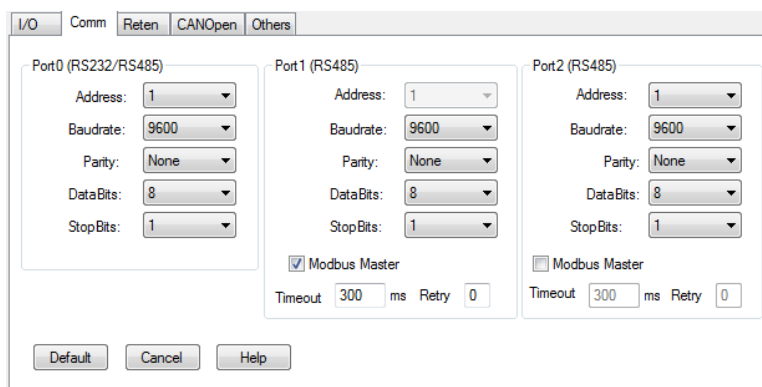


Figure 2-8 Communication Parameters

- After you have modified the parameters, you must download them into the CPU module. *Notice: The configuration parameters won't take effect unless they are downloaded.*

2.7 Example: Common Steps to Create a Project


In order to help the beginners to understand the Kinco-K5 quickly, in the following we'll use a simple example to introduce some common steps for creating and debugging a project step by step. Please refer to the related sections to know a specific function in detail in the following chapters.

Assume that we shall create the following project:

- Project: named "Example";
- Hardware: a Kinco-K506-24AT CPU module;
- Control logic: Toggle Q0.0---Q0.7 in turn and cyclically. For better structure, we use two POU: a subroutine named "Demo" to realize the control logic; the Main Program named "Main" in which "Demo" is invoked.
- Firstly, launch KincoBuilder.
- If necessary, modify the defaults used in KincoBuilder by using the following way:
- Select the [**T**ools]>[**O**ptions...] menu command

The "Options" dialog box appears, in which you can configure some defaults, e.g. the default "Programming language", etc. These defaults will be saved automatically; and so you just need configure them once before the next modification.

Default programming language is [LD Ladder Diagram].

- Create a new project by using one of the following ways:
- Select the [**F**ile]>[**N**ew project...] menu command
- Click the icon  in the toolbar

The "New Project..." dialog box appears. You just need to enter the project name and assign its directory, and then click [**S**ave], the new project shall be created.

For this example, let's select "D:\temp" as the project directory, and name the project as "Example".

➤ Modify the hardware configuration. You can configure the hardware at any time. However, because the hardware configuration is necessary for a project, you are recommended to complete it at first.

When a new project has been created, KincoBuilder will automatically add a default CPU assigned in the “Options” dialog box.

You can open the “Hardware” window by using one of the following ways:

- Double-click the [**Hardware**] node in the **Manager** window;
- Right-click the [**Hardware**] node, and then select the [**Open...**] command on the pop-up menu.

Please refer to [2.6 How to modify the CPU’s communication parameters](#) for detailed steps.

For this example, a Kinco-K506-24AT module with the default parameters is used.

➤ Initializing data

You may initialize the data at any time. You may assign initial values to BYTE, WORD, DWORD, INT, DINT and REAL variables in V area. Before CPU is turned power on and enters into the main loop, the initial data will be processed and the initial values assigned by the user will be valued corresponding addresses.

NOTE: Any memory areas that permanently saved by orders as “initialize data” or “data maintain” will be recovered or valued after CPU enters into the main loop. They will follow a sequence: recover the memory as per defined in “data maintain”, initialize value of areas as per defined in “initialize data”, recover permanently saved data as per defined by users.

➤ Create the example programs.


KincoBuilder provides IL and LD programming languages. You can select the [**Project**]>[**IL**] or [**Project**]>[**LD**] menu command to change the current POU’s language at will.

For this example, a main program named “Main” and a subroutine named “Demo” shall be written in LD language.

◆ Main Program

When creating a new project, KincoBuilder will automatically create an empty main program named “MAIN” at the same time.

◆ Create a new subroutine by using one of the following ways:

- Select the [**Project**]>[**New Subroutine**] menu command
- Click the icon  on the toolbar
- Right-click the [**PROGRAM**] node in the **Manager** window, and then select the [**New Subroutine**] command on the pop-up menu.

Then a new subroutine is created, and its default name is “SBR_0”. Now you can enter the following instructions, as shown in Figure 2-9.

After you have finished entering the instructions, you can rename this subroutine by using the following way: Close this subroutine window; Right-click the “(SBR00) SBR_0” node in the **Manager** window, then select [**Rename**] command on the pop-up menu to modify the name to “Demo”, or select [**Properties...**] command and make modification in the “Property” dialog box.

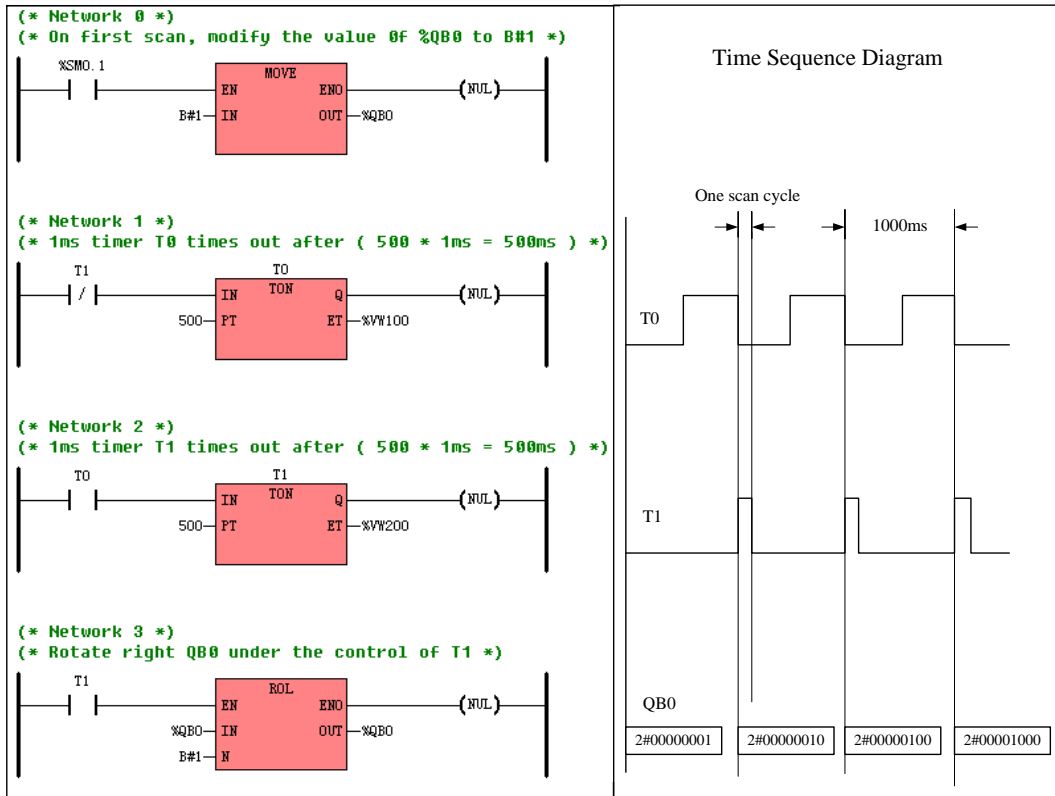


Figure 2-9 the Subroutine “Demo”

- ◆ Modify the main program.

Now we have finished the subroutine “Demo”, and we need to return to the main program to add the following instructions, as shown in the following figure. .

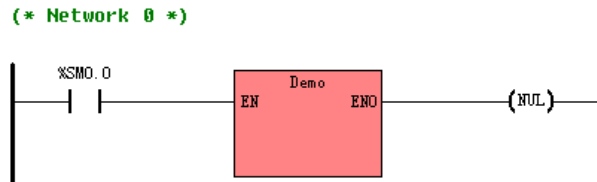



Figure 2-10 the Main Program

➤ Compile the project. After you have finished the whole project, you need to compile it. When compiling a project, KincoBuilder shall save it automatically at first to ensure it is the latest. You can start the compilation by using one of the following ways:


- Select the [PLC]>[**Compile All**] menu command
- Click the icon  on the toolbar
- Use the shortcut key **F7**

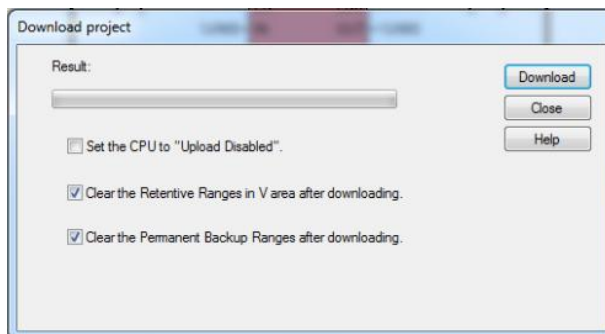
The “Compile” tab in the Output Window keeps a list of the latest compiling messages. To find the source code corresponding to an error, you can double-click on an error message in the “Compile” Window. You have to make modifications according to the error messages until the project is compiled successfully.

- Now, it is time to download the project. Notice: if necessary, you can modify the communication parameters of the computer’s serial port in the [**Communications**] dialog box.

NOTE: You may refer to [2.5 How to connect the computer with the Kinco-K5](#) to find the communication parameters.

You can download the project by using one of the following ways:

- Select [PLC]>[**Download...**] menu command
- Click the icon  on the toolbar
- Use the shortcut key **F8**
- “Download User Project” dialogue



- ◆ [Upload is prohibited]

If this item is clicked, CPU will prohibit any one to upload this project after this download.

- ◆ [Clear the data maintaining area in V area after download]

If this item is clicked, data in V area and C area will be cleared;


If not clicked, data in V area and C area will remain unchanged.

When finish setting, you may click [Start] button to download the project in the PLC and turn CPU to “RUN” to check the project.

If the CPU module is in RUN mode, a dialog box prompts you to place it in STOP mode. Click **Yes** to place it in STOP mode.

After the project has been downloaded, the CPU module goes to RUN mode, and the status LEDs for Q0.0---Q0.7 shall turn on and off in turn and cyclically.

Now, you have completed your first Kinco-K5 project.

➤ You can monitor the programs online by selecting the [**Debug**] > [**Monitor**] menu command or click the icon  on the toolbar, and then KincoBuilder shows the values of all the variables used in the program.

To stop the CPU module, place it in STOP mode by placing the operation switch at STOP position or by selecting the [**Debug**]>[**Stop**] menu command.

- Debug

You may use the online monitoring and Force functions to debug

➤ Online monitoring


Online monitoring contains two modes:

- Monitor in the Variable Status Table. You can input any variable to monitor;
- Monitor in the programme. You can observe how the programme is running. The programme is not allowed to be edited.

Online monitoring can only be effective after opening the Variable Status Table, LD or IL. Please be noted that online monitoring is a check command. Any time if you would like to quit online status you can repeat the

online monitoring command.

You may use any command to enter into online monitoring status as follows:

- [Debug] → [Online monitoring];
- Single click the icon  in the toolbar;
- Shortcut key F6.

➤ Force

You may use the Force to edit the variables value in I area, Q area, M area, V area, AI area or AQ area, among which variables in I area, Q area, M area and V area can be edited by bit, byte, word or double byte while variables in AI area and AQ area can be edited by word.

K5 allows users to use the Force to edit maximum 32 variables. Immediate command does not allow to execute the Force.

You may execute the Force via means as follows:

- Via Variable Status Table. You may detect the variables and input the value for Force via Variable Status Table and proceed with the menu commands (or you can right-click and find menu command [Force], [All Force] and etc.)
- Enter into the status of online monitoring and execute the Force.
On/off Variable: Right-click the Contact and the Coil, execute command [force to 0], [force to 1] or [force ...];
Non On/off Variable: Right-click the variable, execute command [force ...].

At mean time, a variable may have the possible values: values assigned by the user due to external input signal (I, AI) or user programme command (Q, AQ, M, V). Therefore followings rules will be effective:

- As for variables in M area and V area, the Force value will have the same priority with the command: the Force will be executed but will only be effective once in one scanning circle and the command will be effective afterwards;
- As for variables in I area and AI area, the Force value will override that of external input signal. If a Force value is assigned, it will be effective in prior;
- As for variables in Q area and AQ area, the value of external input signal will override in the processing;

but that of Force will override in the output tasks in the scanning circle.

You may use the menu command of [Cancel Force] to cancel the Force of any variable, or use [All Cancel] to cancel the Force of all variables.

When CPU is rebooted, the Force status of all variables will be canceled.

Chapter III Concepts for Programming

This chapter will detailedly introduce the fundamentals for programming Kinco-K5 PLC using KincoBuilder, and also some basic concepts of IEC61131-3 standard that are helpful for you to use any type of IEC61131-3 software. The purpose of this chapter is to help you to start primary programming and practice to achieve a level of “know what and know why”.

At the first reading, you are not recommended to pay it an in-depth understanding of every section but to practise while reading and this will be helpful to easy understanding of this manual.

3.1 POU (Programme Organization Unit)

The blocks from which programs and projects are built are called Program Organisation Units (POUs) in IEC61131-3. As the name implies, POUs are the smallest, independent software units containing the program code. The following three POU types are defined in IEC61131-3:

➤ **Programme**

Keyword: **PROGRAMME**

This type of POU represents the “main program”, and can be executed on controllers. *Programs* form run-time programs by associating with a *TASK* within the configuration.

Programme can have both input and output parameters.

➤ **Function**

Keyword: **FUNCTION**

Functions have both input parameters and a function value as return value. The *Function* always yields the same result as its function value if it is called with the same input parameters.

➤ **Function Block**

Keyword: **FUNCTION_BLOCK**

Function Block is called FB for short in the following sections of this manual.

FB can be assigned input/output parameters and has static variables, and the static variables can memorize the previous status. An FB will yield results that also depend on the status of the static variables if it is called with the same input parameters.

A user project consists of POU's that are either provided by the manufacturer or created by the user. POU's can call each other with or without parameters, and this facilitates the reuse of software units as well as the modularization of the user project. But recursive calls are forbidden, IEC 61131-3 clearly prescribes that POU's cannot call themselves either directly or indirectly

3.2 Data Types

Data types define the number of bits per data element, range of values and its default initial value. All the variables in the user program must be identified by a data type.

A group of elementary data types is defined in IEC61131-3, and as a result, the implications and usage of these data types are open and uniform for PLC programming.

The elementary data types that Kinco-K5 supports at present are shown in the following table.

Keyword	Description	Size in Bits	Range of Values	Default Initial Value
BOOL	Boolean	1	true, false	false
BYTE	Bit string of length 8	8	0 ~ 255	0
WORD	Bit string of length 16	16	0 ~ 65,535	0
DWORD	Bit string of length 32	32	0 ~ 4,294,967,295	0
INT	Signed integer	16	$-2^{15} \sim (2^{15}-1)$	0
DINT	Signed Double integer	32	$-2^{31} \sim (2^{31}-1)$	0
REAL	Floating-point number, ANSI/IEEE 754--1985 standard format	32	$1.18*10^{-38} \sim 3.40*10^{38}$, $-3.40*10^{38} \sim -1.18*10^{-38}$	0.0

Table 3-1 Elementary Data Types that the Kinco-K5 supports

Types of real numbers in K5 will follow the ANSI/IEEE 754-1985, which is described as FLOAT in C Language.

➤ Round-up difference/error of the REAL data

The binary system of real number is not precise. A REAL data will occupy a space of 4 byte and will present valid numbers with digits of maximum 7 digits. Numbers that are longer than this length will be rounded-up.

Valid numbers are data counted from the first number that is not 0 till the last number.

➤ Facts about “0.0”

Due to the round-up difference/error, “0.0” cannot be precisely shown in K5. Any real number that is in the

range of $[-0.000001, 0.000001]$ will be regarded as “0.0”.

➤ **Comparison of real numbers**

When using the comparison commands (GT, GE, EQ, NE, LT, LE), please be noted that two real numbers may not be precisely compared with. As long as the two real number be in the range of $[-0.000001, 0.000001]$ will K5 regards these two number are in equality and vice versa.

3.3 Identifiers

An *identifier* is a string of letters, digits, and underline characters that shall begin with a letter or underline character. (IEC61131-3)

3.3.1 How to define an identifier

You must comply with the following principles while defining an identifier:

- It should begin with a letter or underline character and be followed with some digits, letters or underline characters.
- Identifiers are not case-sensitive. For example, the identifiers abc, ABC and aBC shall be the same.
- The maximum length of the identifier is only restricted by each programming system.
In KincoBuilder, the maximum length of the identifier is 16-character.
- *Keywords* cannot be used as user-defined identifiers. *Keywords* are standard identifiers, and reserved for programming languages of IEC 61131-3.

3.3.2 Use of Identifiers

The language elements that can use identifiers in KincoBuilder are as follows:

- Programme name, function name and the FB instance name
- Variable name
- Label, etc.

3.4 Constant

A *constant* is a lexical unit that directly represents a value in a program. Use constants to represent numeric, character string or time values that cannot be modified. Constants are characterized by having a value and a data type. The features and examples of the constants that Kinco-K5 supports at present are shown in the following table.

Data Type	Format ⁽¹⁾	Range of value	Example
BOOL	true, false	true, false	false
BYTE	B#digits	B#0 ~ B#255	B#129
	B#2#binary digits		B#2#10010110
	B#8#octal digits		B#8#173
	B#16#hex digits		B#16#3E
WORD	W#digits	W#0 ~ W#65535	W#39675
	2#binary digits		2#100110011
	W#2#binary digits		W#2#110011
	8#octal digits		8#7432
	W#8#octal digits		8#174732
	16#hex digits		16#6A7D
	W#16#hex digits		W#16#9BFE
DWORD	DW#digits	DW#0 ~ DW#4294967295	DW#547321
	DW#2#binary digits		DW#2#10111
	DW#8#octal digits		DW#8#76543
	DW#16#hex digits		DW#16#FF7D
INT	Digits	-32768 ~ 32767	12345
	I#digits		I#-2345
	I#2#binary digits ⁽²⁾		I#2#1111110
	I#8#octal digits ⁽²⁾		I#8#16732
	I#16#hex digits ⁽²⁾		I#16#7FFF
DINT	DI#digits	DI#-2147483647 ~ DI#2147483647	DI#8976540
	DI#2#binary digits ⁽²⁾		DI#2#101111

	DI#8#octal digits ⁽²⁾		DI#8#126732
	DI#16#hex digits ⁽²⁾		DI#16#2A7FF
REAL	Digits with decimal point	$1.18*10^{-38} \sim 3.40*10^{38}$,	1.0, -243.456
	xEy	$-3.40*10^{38} \sim -1.18*10^{-38}$	-2.3E-23

Table 3-2 Constants



Notice:

- *The descriptor is not case-sensitive, e.g. the constants W#234 and w#234 shall be the same.*
- *The binary, octal and hex representations of INT and DINT constants all adopt standard Two's Complement Representation, and the MSB is the sign bit: a negative number if MSB is 1, a positive number if MSB is 0. For example, I#16#FFFF = -1, I#7FFF = 32767, I#8000 = -32768, etc.*

3.5 Variables

In contrast to *constants*, *variables* provide a means of identifying data objects whose contents may change **错误! 未找到引用源。**, e.g., data associated with the inputs, outputs, or memory of the PLC. (IEC61131-3)

Variables are used to initialize, memorize and process data objects. A variable must be declared to be a fixed data type. The storage location of a variable, i.e. the data object associated with a variable, can be defined manually by the user, or be allocated automatically by the programming system.

3.5.1 Declaration

A variable must be declared before it is used. Variables can be declared out of a POU and used globally; also, they can be declared as interface parameters or local variables of a POU. Variables are divided into different *variable types* for declaration purposes.

The standard variable types supported by Kinco-K5 are described in the following table. In the table, “Internal” indicates whether the variable can be read or written to within the POU in which it is declared, and “External” indicates whether the variable can be visible and can be read or written to within the calling POU.

Variable Type	External	Internal	Description
VAR	---	Read/Write	Local variables. They can only be accessed within their own POU.
VAR_INPUT	Write	Read	Input variables of the calling interface, i.e. formal input parameters. They can be written to within the calling POU, but can only be read within their own POU.
VAR_OUTPUT	Read	Read/Write	Output variables, which act as the return values of their own POU. They are read-only within the calling POU, but can be read and written to within their own POU.
VAR_IN_OUT	Read/Write	Read/Write	Input/output variables of the calling interface, i.e. formal input/output parameters. They have the combined features of VAR_INPUT and

			VAR_OUTPUT.
VAR_GLOBAL	Read/Write	Read/Write	Global variables. They can be read and written to within all POU's.

Table 2-3 Variable Types

3.5.2 Declaring Variables in KincoBuilder

Each type of variables shall be declared within the respective table, and thus it is convenient for you to enter the data. Moreover, KincoBuilder can strictly check your inputs.

Global variables are declared within the Global Variable Table, and other variables are declared within the Variable Table of the respective POU. Each POU has its own separate Variable Table.

If you use the same name for a variable at the local and global level, the local use takes precedence within its POU.

3.5.3 Checking Variables

While programming, KincoBuilder shall check the usage of each variable to verify whether it is accessed using the proper data type and variable type. For example, when a REAL value is assigned to a WORD variable or a VAR_INPUT variable is modified in its POU, KincoBuilder will warn you and prompt for modification.

Because the characteristic of a variable depends on its variable type and data type, the strict checking can assist you in avoiding those errors resulting from improper use of variables.

3.6 How to Access PLC Memory

The Kinco-K5 stores information in different memory units. To be convenient for the users, the Kinco-K5 provides two addressing methods to access the memory units:

- Direct Addressing
- Indirect addressing, i.e. pointer.

3.6.1 Memory Types and Characteristics

The memory of the Kinco-K5 PLC is divided into several different areas for different usage purposes, and each memory area has its own characteristics. The details are shown in the following table.

I	
Description	DI (Digital Input) Image Area. The Kinco-K5 reads all the physical DI channels at the beginning of each scan cycle and writes these values to I area.
Access Mode	Can be accessed by bit, by byte, by word and by double word
Access Right	Read only
Others	Can be forced, and cannot be retentive
Q	
Description	DO (Digital Output) Image Area. At the end of each scan cycle, the Kinco-K5 writes the values stored in Q area to the physical DO channels.
Access Mode	Can be accessed by bit, by byte, by word and by double word
Access Right	Read/write
Others	Can be forced, and cannot be retentive
AI	
Description	AI (Analog Input) Image Area. The Kinco-K5 samples all the AI channels at the beginning of each scan cycle, and converts the analog input values (such as current or voltage) into 16-bit digital values and writes these values to AI area.

Access Mode	Can be accessed by word (the data type is INT)
Access Right	Read only
Others	Can be forced, and cannot be retentive
AQ	
Description	AO (Analog Output) Image Area. At the end of each scan cycle, The Kinco-K5 converts the 16-bit digital values stored in AQ area into field signal values and writes to AO channels.
Access Mode	Can be accessed by word (the data type is INT)
Access Right	Read/write
Others	Can be forced, and cannot be retentive
HC	
Description	High-speed Counter Area. Used to store the current counting value of the high-speed counters.
Access Mode	Can be accessed by double word (the data type is DINT)
Access Right	Read only
Others	Cannot be forced, and cannot be retentive
V	
Description	Variable Area. It's relatively large and can be used to store a large quantity of data.
Access Mode	Can be accessed by bit, by byte, by word and by double word
Access Right	Read/write
Others	Can be forced, and can be retentive
M	
Description	Internal Memory Area. It can be used to store the internal status or other data. Compared with V area, M area can be accessed faster and more propitious to bit operation.
Access Mode	Can be accessed by bit, by byte, by word and by double word
Access Right	Read/write
Others	Can be forced, and can be retentive
SM	

Description	System Memory Area. System data are stored here. You can read some SM addresses to evaluate the current system status, and you can modify some addresses to control some system functions.
Access Mode	Can be accessed by bit, by byte, by word and by double word
Access Right	Read/write
Others	Cannot be forced and cannot be retentive
L	
Description	Local Variable Area. KincoBuilder shall assign memory locations in L area for all the local variables and input/output variables automatically. You are not recommended to access L area directly.
Access Mode	Can be accessed by bit, by byte, by word and by double word
Access Right	Read/write
Others	Cannot be forced and cannot be retentive

Table 3-4 Memory Types and Characteristics

3.6.2 Direct Addressing

Direct addressing means that variables can be assigned to the memory units to directly access them.

➤ Directly represented variable

According to IEC61131-3, direct representation of a single-element variable is provided by a special symbol formed by the concatenation of the percent sign “%”, a memory area identifier and a data size designation, and one or more unsigned integers, separated by periods (.)错误!未找到引用源。. For example, %QB7 refers to output byte location 7.

‘Directly represented variable’ corresponds to ‘Direct address’ in traditional PLC systems.

➤ Symbolic variable

You can assign a symbolic name to a ‘Directly represented variable’ to identify it conveniently. Identifier shall be used for symbolic representation of variables.

In KincoBuilder, you can declare symbolic variables within the Global Variable Table and the Variable Table

of the respective POU. Please refer to the corresponding sections for more information.

3.6.2.1 Directly represented variable

Direct address representation for each memory area is shown in the following table, wherein either x or y represents a decimal number.

➤ I Area

Bit Addressing	Format	%Ix.y
	Description	x : byte address of the variable y : bit number, i.e. bit of byte. Its range is 0 ~ 7.
	Data type	BOOL
	Example	%I0.0 %I0.7 %I5.6
Byte Addressing	Format	%IBx
	Description	x : byte address of the variable
	Data type	BYTE
	Example	%IB0 %IB1 %IB5
Word Addressing	Format	%IWx
	Description	x : starting byte address of the variable. Since the size of WORD is 2 bytes, x must be an even number.
	Data type	WORD, INT
	Example	%IW0 %IW2 %IW4
Double word Addressing	Format	%IDx
	Description	x : starting byte address of the variable. Since the size of DWORD is 4 bytes, x must be an even number.
	Data type	DWORD, DINT
	Example	%ID0 %ID4

➤ Q Area

Bit	Format	%Qx.y
------------	--------	--------------

Addressing	Description	x : byte address of the variable y : bit number, i.e. bit of byte. Its range is 0 ~ 7.
	Data type	BOOL
	Example	%Q0.0 %Q0.7 %Q5.6
Byte Addressing	Format	%QBx
	Description	x : byte address of the variable
	Data type	BYTE
	Example	%QB0 %QB1 %QB4
Word Addressing	Format	%QWx
	Description	x : starting byte address of the variable. Since the size of WORD is 2 bytes, x must be an even number.
	Data type	WORD, INT
	Example	%QW0 %QW2 %QW4
Double word Addressing	Format	%QDx
	Description	x : starting byte address of the variable. Since the size of DWORD is 4 bytes, x must be an even number.
	Data type	DWORD, DINT
	Example	%QD0 %QD4 %QD12

➤ **AI Area**

Word Addressing	Format	%AIWx
	Description	x : starting byte address of the variable. Since the size of INT is 2 bytes, x must be an even number.
	Data type	INT
	Example	%AIW0 %AIW2 %AIW12

➤ **AQ Area**

Word Addressing	Format	%AQWx
	Description	x : starting byte address of the variable. Since the size of INT is 2 bytes, x must be an even number.
	Data type	INT
	Example	%AQW0 %AQW2 %AQW12

➤ **M Area**

Bit Addressing	Format	%M_{x.y}
	Description	<i>x</i> : byte address of the variable <i>y</i> : bit number, i.e. bit of byte. Its range is 0 ~ 7.
	Data type	BOOL
	Example	%M0.0 %M0.7 %M5.6
Byte Addressing	Format	%MB_x
	Description	<i>x</i> : byte address of the variable
	Data type	BYTE
	Example	%MB0 %MB1 %MB10
Word Addressing	Format	%MW_x
	Description	<i>x</i> : starting byte address of the variable. Since the size of WORD is 2 bytes, <i>x</i> must be an even number.
	Data type	WORD, INT
	Example	%MW0 %MW2 %MW12
Double word Addressing	Format	%MD_x
	Description	<i>x</i> : starting byte address of the variable. Since the size of DWORD is 4 bytes, <i>x</i> must be an even number.
	Data type	DWORD, DINT
	Example	%MD0 %MD4 %MD12

➤ **V Area**

Bit Addressing	Format	%V_{x.y}
	Description	<i>x</i> : byte address of the variable <i>y</i> : bit number, i.e. bit of byte. Its range is 0 ~ 7.
	Data type	BOOL
	Example	%V0.0 %V0.7 %V5.6
Byte Addressing	Format	%VB_x
	Description	<i>x</i> : byte address of the variable
	Data type	BYTE
	Example	%VB0 %VB1 %VB10

Word Addressing	Format	%VW_x
	Description	<i>x</i> : starting byte address of the variable. Since the size of WORD is 2 bytes, <i>x</i> must be an even number.
	Data type	WORD, INT
	Example	%VW0 %VW2 %VW12
Double word Addressing	Format	%VD_x
	Description	<i>x</i> : starting byte address of the variable. Since the size of DWORD is 4 bytes, <i>x</i> must be an even number.
	Data type	DWORD, DINT
	Example	%VD0 %VD4 %VD12
REAL Addressing	Format	%VR_x
	Description	<i>x</i> : starting byte address of the variable. Since the size of REAL is 4 bytes, <i>x</i> must be an even number.
	Data type	REAL
	Example	%VR0 %VR4 %VR1200

➤ **SM Area**

Bit Addressing	Format	%SM_{x.y}
	Description	<i>x</i> : byte address of the variable <i>y</i> : bit number, i.e. bit of byte. Its range is 0 ~ 7.
	Data type	BOOL
	Example	%SM0.0 %SM0.7 %SM5.6
Byte Addressing	Format	%SMB_x
	Description	<i>x</i> : byte address of the variable
	Data type	BYTE
	Example	%SMB0 %SMB1 %SMB10
Word Addressing	Format	%SMW_x
	Description	<i>x</i> : starting byte address of the variable. Since the size of WORD is 2 bytes, <i>x</i> must be an even number.
	Data type	WORD, INT
	Example	%SMW0 %SMW2 %SMW12
Double word	Format	%SMD_x

Addressing	Description	x : starting byte address of the variable. Since the size of DWORD is 4 bytes, x must be an even number.
	Data type	DWORD, DINT
	Example	%SMD0 %SMD4 %SMD12

➤ **L Area (Notice: You are not recommended to access L area directly.)**

Bit Addressing	Format	%Lx,y
	Description	x : byte address of the variable y : bit number, i.e. bit of byte. Its range is 0 ~ 7.
	Data type	BOOL
	Example	%L0.0 %L0.7 %L5.6
Byte Addressing	Format	%LBx
	Description	x : byte address of the variable
	Data type	BYTE
	Example	%LB0 %LB1 %LB10
Word Addressing	Format	%LWx
	Description	x : starting byte address of the variable. Since the size of WORD is 2 bytes, x must be an even number.
	Data type	WORD, INT
	Example	%LW0 %LW2 %LW12
Double word Addressing	Format	%LDx
	Description	x : starting byte address of the variable. Since the size of DWORD is 4 bytes, x must be an even number.
	Data type	DWORD, DINT, REAL
	Example	%LD0 %LD4 %LD12

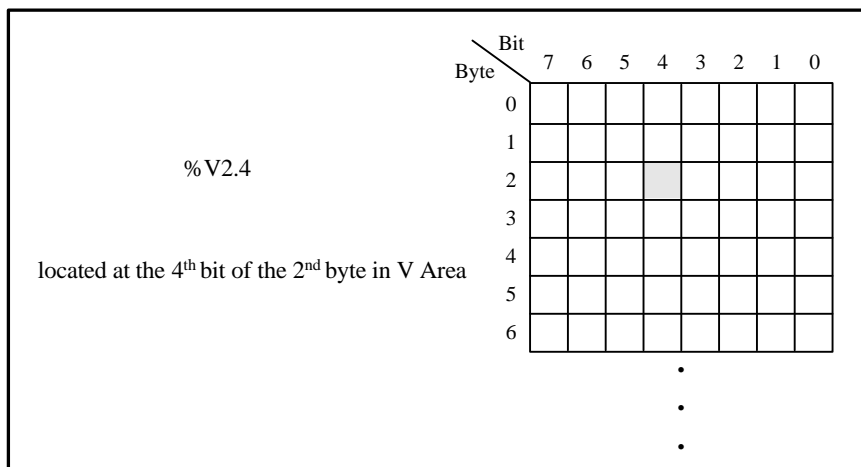
➤ **HC Area**

Double word Addressing	Format	%HCx
	Description	x : the high-speed counter number
	Data type	DINT
	Example	%HC0 %HC1

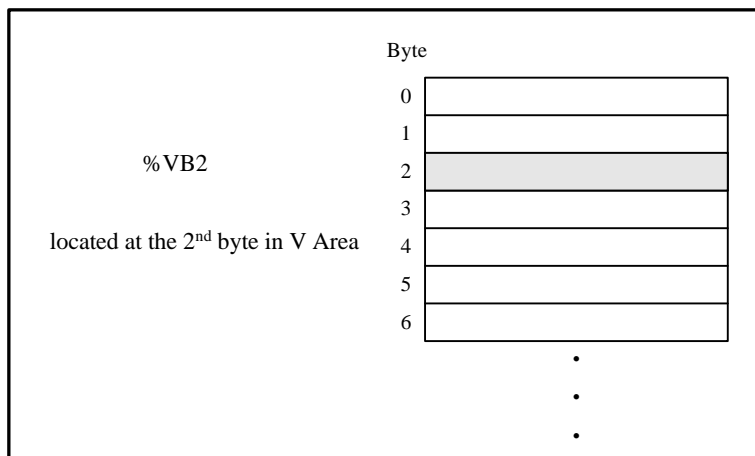
3.6.2.2 Mapping between Direct Address and PLC Memory Location

Each valid direct address corresponds to a PLC memory location, and the mapping relation between them is shown in the following diagram taking V area as an example.

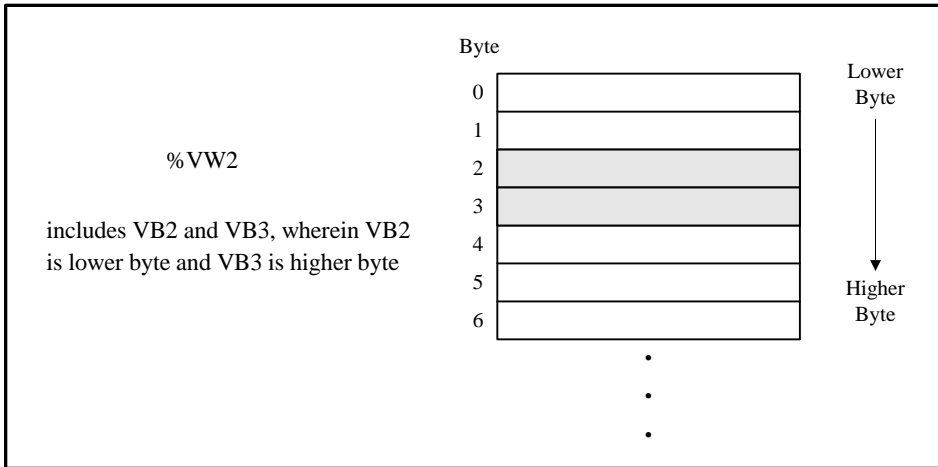
➤ Bit Addressing



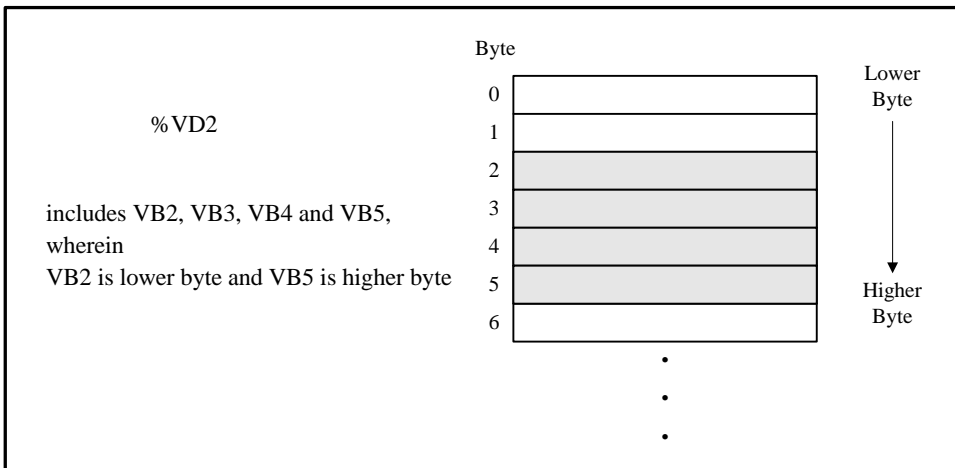
➤ Byte Addressing



➤ **Word Addressing**



➤ **Double word Addressing**



3.6.3 Indirect Addressing

A pointer is a double word variable which stores the physical address of a memory unit. Indirect addressing uses a pointer to access the data in the corresponding memory.

The Kinco-K5 allows pointers to access the V area (except an individual bit) only. In addition, only the

'Directly represented variable' in the V area can be used as pointer.

3.6.3.1 Creating a pointer

To indirectly access the data in a memory unit, you have to create a pointer to that unit firstly. The address operator '&' can be used, e.g., &VB100 stands for the physical address of VB100.

You can create a pointer using the following way: entering the address operator (&) in front of a directly represented variable to get its physical address, and then write the physical address into another directly represented variable as a pointer using the MOVE instruction.

For example:

(* Create a pointer (VD204) which points to VW2. i.e., the physical address of VW2 is stored in VD204. *)

```
MOVE    &VW2, %VD204
```

3.6.3.2 Access data using a pointer

'*' is the pointer operator. Entering a '*' in front of a pointer represents the direct address variable to which this pointer points. While using a pointer as an operand of an instruction, please pay attention to the data types of the instruction's operands.

For example:

```
LD      %SM0.0
```

```
MOVE    &VB0, %VD200    (*Create a pointer (VD200) which points to VB0. *)
```

```
MOVE    *VD200, %VB10    (* Assign the value of VB0 to VB10. The pointer VD200 points to VB0, *)  
                          (* so *VD200 represents VB0. *)
```

3.6.3.3 Modifying the value of a pointer

A pointer is a 32-bit variable, and so its value can be modified with such instructions as ADD and SUB, etc. Whenever a pointer's value is increased / reduced by 1, the direct address to which it points will be increased /

reduced by 1 byte correspondingly. So when you modify a pointer's value, you must pay attention to the data type of the variable pointed to.

- If a pointer points to a BYTE variable, you can modify the pointer's value by any double integer number.
- If a pointer points to a WORD or INT variable, you can modify the pointer's value by a multiple of 2.
- If a pointer points to a DWORD, DINT or REAL variable, you can modify the pointer's value by a multiple of 4.

3.6.3.4 Notice for using the pointers

- The validity of a pointer is guarantee by the user program. The pointer is very flexible, so you need to be very careful when using it. If a pointer points to an illegal address, it may lead to unexpected results.
- The Kinco-K5 only supports single-level pointer and address, multiple-level pointers and addresses are illegal. For example, the following instruction is illegal:

```
MOVE    &VB4, *VD44
```

3.6.3.5 Example

(* Network 0 *)

```
LD      %SM0.0
MOVE    &VW0, %VD200    (*Create a pointer (VD200) which points to VW0. *)
MOVE    *VD200, %VW50   (* Assign the value of VW0 to VW50. The pointer VD200 points to VW0, *)
                          (* so *VD200 represents VW0. *)
ADD     DI#2, %VD200    (* The pointer's value increases by 2, so it points to VW2 now. *)
MOVE    *VD200, %VW52   (* Assign the value of VW2 to VW52 *)
```

3.6.4 Memory Address Ranges

The Kinco-K5 provides several types of CPU module. The memory address ranges of different types of CPU may be different from each other, and the addresses beyond the respective range are illegal. In your program, you

must ensure that all the memory addresses that you enter are valid for the CPU. The detailed descriptions are given in the following table.

		CPU504	CPU504EX	CPU506, CPU506EA, CPU508
I	Size	1	5	32
	Bit address	%I0.0 --- %I0.7	%I0.0 --- %I4.7	%I0.0 --- %I31.7
	Byte address	%IB0, IB1	%IB0 --- %IB4	%IB0 --- %IB31
	Word address	%IW0	%IW0 --- %IW2	%IW0 --- %IW30
	Double-word address	-----	%ID0	%ID0 --- %ID28
Q	Size	1	5	32
	Bit address	%Q0.0 --- %Q0.7	%Q0.0 --- %Q4.7	%Q31.0 --- %Q31.7
	Byte address	%QB0	%QB0 --- %QB4	%QB0 --- %QB31
	Word address	-----	%QW0 --- %QW2	%QW0 --- %QW30
	Double-word address	-----	%QD0	%QD0 --- %QD28
AI	Size	0	16	64
	Word address	-----	%AIW0 --- %AIW14	%AIW0 --- %AIW62
AQ	Size	0	16	64
	Word address	-----	%AQW0 -- %AQW14	%AQW0 -- %AQW62
HC	Size	8		
	Word address	%HC0, %HC1		
V	Size	4096		
	Bit address	%V0.0 --- %V4095.7		
	Byte address	%VB0 --- %VB4095		
	Word address	%VW0 --- %VW4094		

	Double-word address	%VD0 --- %VD4092 %VR0 --- %VR4092
M	REAL address	1024
	Size	%M0.0 --- %M1023.7
	Bit address	%MB0 --- %MB1023
	Byte address	%MW0 --- %MW1022
	Word address	%MD0 --- %MD1020
SM	Double-word address	300
	Size	%SM0.0 --- %SM299.7
	Bit address	%SMB0 --- %SMB299
	Byte address	%SMW0 --- %SMW298
	Word address	%SMD0 --- %SMD296
L	Double-word address	272
	Size	%L0.0 --- %L271.7
	Bit address	%LB0 --- %LB271
	Byte address	%LW0 --- %LW270
	Word address	%LD0 --- %LD268

Table 3-5 CPU Memory Ranges

3.6.5 Function Block and Function Block Instance

3.6.5.1 Standard Function Blocks in IEC61131-3

- Timers: TP --- Pulse timer; TON --- On-delay timer; TOF --- Off-delay timer
- Counters: CTU --- Up-counter; CTD --- Down-counter; CTUD --- Up-Down counter
- Bistable elements: SR --- Set dominant; RS --- Ret dominant

- Edge detection: R_TRIG --- Rising edge detector; F_TRIG --- Falling edge detector

3.6.5.2 Instances of Function Blocks

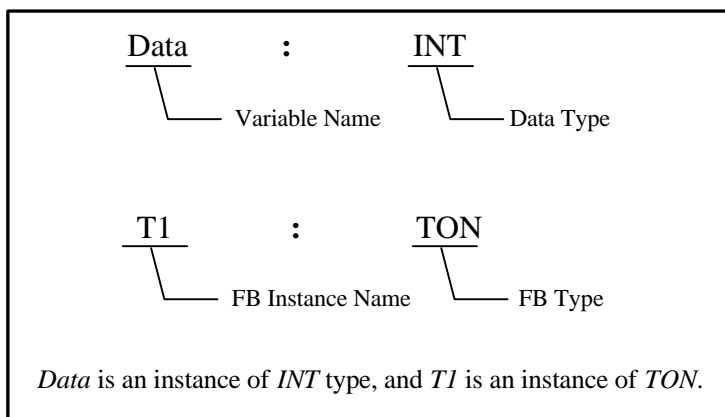
“Instantiation of FBs” is a very important concept in IEC61131-3.

Instantiation means that the programmer declares and creates a variable by specifying its name and data type.

After instantiation, the variable can be accessed in the programme.

FB also needs to be instantiated as a variable does. After instantiation, a FB (as an instance) can be used in the POU in which it is declared.

As shown in the following graph, only T1 can be called and accessed.



3.6.5.3 FB Instance Memory Areas

A fixed memory area is allocated for each type of FB to store its instances in the Kinco-K5 PLC, and the details are shown in the following table.

T	
Description	Timer Memory Area, where instances of TON, TOF and TP can be allocated. It's used to store the status bits and current values of all the timer instances.
Access mode	Directly access the status bit and current value of a timer instance

Access right	Read only
Others	Can not be retentive, and can not be forced
C	
Description	Counter Memory Area, where the instances of CTU, CTD and CTUD can be allocated. It's used to store the status bits and current values of all the counter instances.
Access mode	Directly access the status bit and current value of a counter instance
Access right	Read-only
Others	Can be retentive, and can not be forced
RS	
Description	RS Bistable Area, where instances of RS can be allocated. It's used for storing the status bits for all the RS instances.
Access Mode	Directly access the status of the RS instances
Access Rights	Read-only
Others	Can not be retentive, and can not be forced
SR	
Description	SR Bistable Area, where instances of SR can be allocated. It's used for storing the status for all the SR instances.
Access Mode	Directly access the status bit of the SR instances
Access Rights	Read-only
Others	Can not be retentive, and can not be forced

Table 3-6 FB Instance Memory Areas

3.6.6 Using FB Instances

A FB instance must be declared before it is used.

For the convenience of users, KincoBuilder complies with the following rules: the representation of FB instances accords with the traditional PLC, e.g. T0, C3; you just need to call the valid FB instances of the desired types in your programme, and KincoBuilder will generate the declarations automatically in the Global Variable Table.

➤ T

Format	T_x
Description	<i>x</i> : a decimal digit, indicating the timer number.
Data type	<p>BOOL --- status bit of the timer</p> <p>INT --- current value of the timer</p> <p>T_x is used to access both of the two variables. KincoBuilder will identify access to either the status bit or the current value according to the instruction used: instructions with BOOL operands access the status bit, but instructions with INT operands access the current value.</p>
Example	T0 T5 T20

➤ **C**

Format	C_x
Description	<i>x</i> : a decimal digit, indicating the counter number.
Data type	<p>BOOL --- status bit of the counter</p> <p>INT --- current counting value of the counter</p> <p>C_x is used to access both of the two variables. KincoBuilder will identify access to either the status bit or the current value according to the instruction used: instructions with BOOL operands access the status bit, but instructions with INT operands access the current value.</p>
Example	C0 C5 C20

➤ **RS**

Format	RS_x
Description	<i>x</i> : a decimal digit, indicating the RS Bistable number.
Data Types	BOOL --- the status of the RS Bistable
Example	RS0, RS5, RS10

➤ **SR**

Format	SR_x
Description	<i>x</i> : a decimal digit, indicating the SR Bistable number.
Data Types	BOOL --- the status of the SR Bistable

Example	SR0, SR5, SR10
----------------	----------------

3.6.7 FB Instances Memory Ranges

The size of the memory area that the PLC can allocate to a type of FB instances is limited by the resource of the hardware; therefore, each type of Kinco-K5 CPU allocates a different memory range for the FB instances. The detailed descriptions are given in the following table.

T	Amount	256
	Range	T0 --- T255
	Resolution	T0 --- T3: 1ms T4 --- T19: 10ms T20 --- T255: 100ms
	Max timing	32767* Resolution
C	Amount	256
	Range	C0 --- C255
	Max counting value	32767
RS	Amount	32
	Range	RS0 --- RS31
SR	Amount	32
	Range	SR0 --- SR31

Table 3-7 FB Instances Memory Ranges

Chapter IV How to Use KincoBuilder ... Basic Functions

This chapter describes the components of KincoBuilder detailedly, including their functions and operating steps. Based on the basic concepts in the previous chapters, this chapter can help you get a further and comprehensive understanding of KincoBuilder.

LD editor and IL editor may involve IEC61131-3 grammar, which will be introduced in the next chapter.

4.1 Configuring General Software Options

You need to configure some general options for KincoBuilder, e.g. the default programming language and the default CPU type for new projects. KincoBuilder will save your configuration automatically, so you just need configure them once before the next modification

Select the [Tools]>[Options...] menu command, and then the following dialogue box will popup:

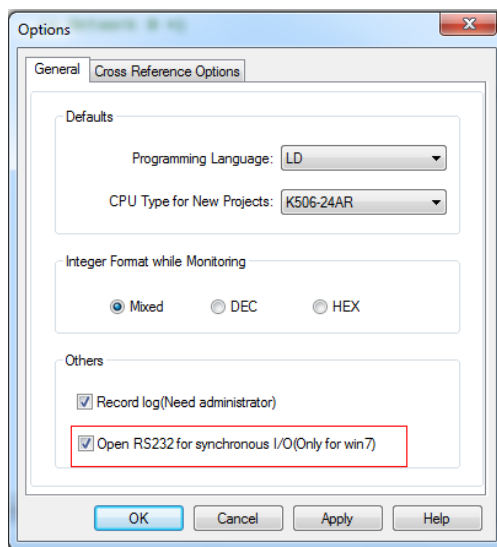


Figure 4-1 The “Options” Dialog Box

① **General Tab**

➤ **Defaults**

• **Programming Language:**

Choose the default programming language for new programs, IL or LD.

• **CPU Type for New Projects:**

Choose the CPU type that new projects always default to use.

➤ **Integer Format While Monitoring**

Choose the display format for the integer values while monitoring.

Mixed: The INT and DINT values are displayed in decimal format;

In addition, the BYTE, WORD and DWORD values are displayed in hexadecimal format.

DEC: All the integer values are displayed in decimal format.

HEX: All the integer values are displayed in hexadecimal format.

➤ **Others**

• **Compile the project before downloading:**


If this is checked, KincoBuilder will automatically compile the current project before downloading.

• **Compile the project before monitoring:**


If this is checked, KincoBuilder will automatically compile the current project before monitoring.

4.2 About Docking Windows

In KincoBuilder, the Manager Window, the Instructions Window, the Output Window and the PLC Catalog Window are designed as docking windows. A docking window has two display modes: floating or docked. In floating mode, a window can appear anywhere on your screen. In docked mode, a window is fixed to a dock along any of the four borders of the main KincoBuilder window.

- To change a docked window to a floating window
 - Double-click in the window border.
 - Point to the title bar and drag the window out of its dock area.
- To dock a floating window
 - Double-click the window title bar to return the window to its previous docked location.
 - Point to the title bar and drag the window to a dock area.
- To switch a docking window to auto-hide mode
 - Click the icon  located on the top-right corner of the window.

In auto-hide mode, it shall hide automatically and shrink into an icon and stay at the border of the main KincoBuilder window; Point to this icon for a moment, the window shall appear.

 - To cancel the auto-hide mode of a docking window
 - Click the icon  to return the window to its previous docked location.

4.3 Configuring Hardware

In a project, you are recommended to finish configuring hardware at first. When a new project has been created, a default CPU assigned in the “Options” dialog box shall be added automatically and you can modify it at will. KincoBuilder provides you with a complete, flexible and convenient hardware configuration environment where you can configure all the parameters for each PLC module. The “Hardware” window is shown as Figure 4-2. We can see that this window is composed of two parts:

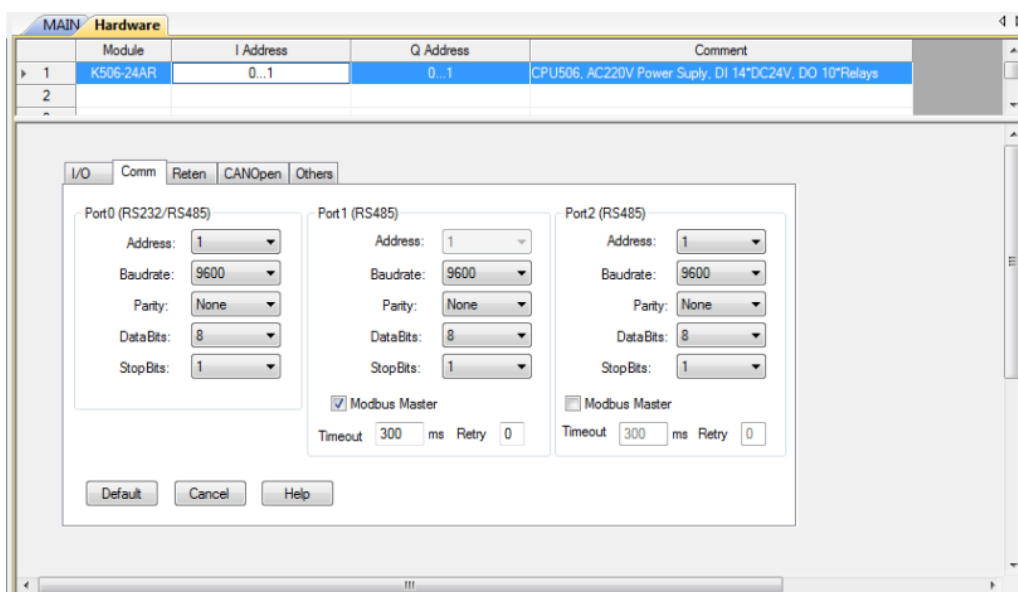


Figure 4-2 the Hardware Window

➤ **The Configuration Table**

The upper part of the hardware window shows a detailed list of the PLC modules in table form, and we call it Configuration Table. The Configuration Table represents the real configuration: you arrange your modules in the Configuration Table just as you do in a real control system.

➤ **The Parameters Window**

The lower part of the hardware window shows all the parameters of the selected module in the Configuration Table, and we call it Parameters Window.

4.3.1 How to open the Hardware window

You can open the “Hardware” window by using one of the following ways:

- Double-click the [**Hardware**] node in the **Manager** window.
- Right-click the [**Hardware**] node, and then select the [**Open**] command on the pop-up menu.

4.3.2 Copy and paste the hardware configuration in different projects

In KincoBuilder, users are allowed to copy and paste [Hardware Configuration] in different projects. The [Hardware Configuration] refers to configuration of of CANOpen, communication port, etc. and will not copy the information of CPU, which means it can be executed between CPUs. All pending projects must be opened by KincoBuilder. You may use the copy and paste function the same time with that of LD or IL.

This function will be helpful if you would like to transplant the configuration of CANOpen of old projects.

You may use this function as follows:

- Click [Copy Hardware Configuration], [Paste Hardware Configuration] in the [Edit] menu;
- Right-click [PLC Hardware Configuration] in the [Project Manager] tree and execute [Copy Hardware Configuration] and [Paste Hardware Configuration]

4.3.3 Add/Remove Modules

➤ **Add a module**

You can add a module using the following steps:

- In the Configuration Table, click a row to place the focus on it. If there exists a module in this row, it must be removed before adding a new module.

- In the PLC Catalog Window, double-click a module to add it to the row with the current focus in the Configuration Table.

Row 1 can only be added into with a CPU module, and other rows can only be added into with the expansion modules. There shall not be any null rows between each two modules. If a null row exists, KincoBuilder will not allow continuing to add modules after it, and an error message-box will popup when saving or compiling the project.

- **Remove a module**

You can remove a module by using the following ways:

- Click the module to be removed in the Configuration Table, then use **Del** key to remove it.
- Right-click the module to be removed, and then select the [**Remove**] command on the pop-up menu.

4.3.4 Configuring Module Parameters

Once you have arranged your modules in the Configuration Table, you can continue to assign their parameters. KincoBuilder allows you define all of the parameters of a module.

In the Configuration Table, click a PLC module to place the focus on it, and then the Parameters Window of this module shall appear below. You can assign a module's parameters in its Parameters Window. Of course, you can use **Up** and **Down** arrow key to move the focus in the Configuration Table

On the right hand of the Parameters Window, there are two public buttons: [**Default**] and [**Cancel**].

- [**Default**]: If you click this button, KincoBuilder will assign default parameters for the current module.
- [**Cancel**]: If you click this button, the original configuration of the current module will be restored.



Notice: The addresses of the modules in the same memory area (I, Q, AI or AQ) cannot overlap!

4.3.4.1 Parameters of the CPU

① [I/O Configuration] tab

Here you can assign the I/O parameters of the CPU module, as shown in the following figure.

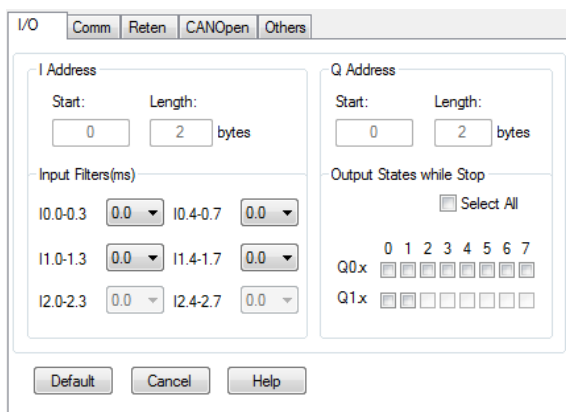


Figure 4-3 I/O Parameters of the CPU

- **Input:** Here, you can configure the DI channels on the CPU body.
- **I Address:** the start byte address of the DI channels in I area. It is fixed to be 0.
- **Input Filters:** Select an input filter (ms) that defines a delay time for DI channels. This delay is helpful to filter the input noise and enhance the anti-interference capacity of the control system. When an input state changes, it won't be accepted as valid unless it remains for the duration of the filter time.
- **Output:** Here, you can configure the DO channels on the CPU body.
- **Q Address:** the start byte address of the DO channels in Q area. It is fixed to be 0.
- **Output States while STOP:** Set the digital outputs in a known state while the CPU stops. If the checkbox for an output is checked, the output shall be set to ON (1) while the CPU stops. The default state of an output while the CPU stops is OFF (0). This function is very significant for safety interlock requirements after a RUN-to-STOP transition.

② [Communication Ports] tab

Here you can assign the serial communication parameters for Port0 and Port1 on the CPU module.

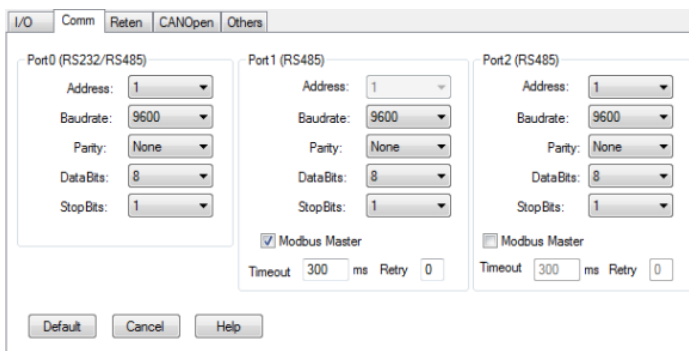


Figure 4-4 Serial Communication Parameters

■ Port0

- **Address:** Choose the desired station address of Port0. This address also acts as a Modbus RTU slave number, and it must be exclusive in the network.
- **Baudrate:** Select the desired baud rate. (2400, 4800, 9600, 19200, 38400, 57600or115200bps)
- **Parity:** Select the desired parity scheme. (No parity, Odd, or Even)
- **DataBits:** Select the number of bits in the bytes transmitted and received. (8)
- **StopBits:** Select the number of stop bits. (1)

■ Port1 and Port2

Port1 and Port2 are RS485 ports.

- **Modbus Master:** If the checkbox is checked, Port1 will work as a Modbus RTU master.
- **Timeout:** Enter a timeout value for this Modbus master.
- **Retry:** Enter the value of retry times. When the master receives a wrong frame from a slave, it will retry to communicate with the slave for '**Retry**' times.
- **Baudrate:** Select the desired baud rate. (2400, 4800, 9600, 19200or38400bps)
- **Parity:** Select the desired parity scheme. (No parity, Odd, or Even)
- **DataBits:** Select the number of bits in the bytes transmitted and received. (8)
- **StopBits:** Select the number of stop bits. (1)

③ **[Retentive Ranges]** tab

Here you can define four retentive ranges to select the ranges of the RAM you want to retain on power loss. If the CPU loses power, the instantaneous data in the RAM will be maintained by the super capacitor, and only the data in the retentive ranges will be left unchanged at next power on.

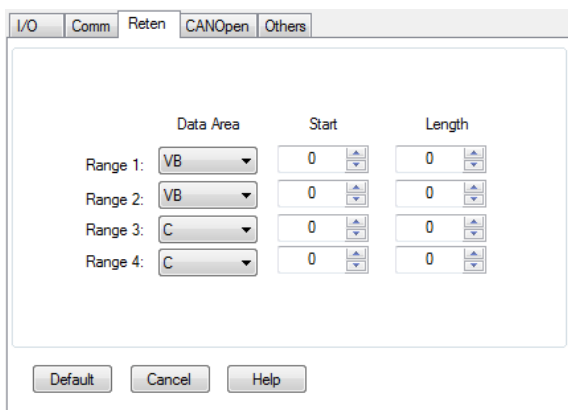


Figure 4-5 Retentive Ranges

■ **Range 1**

• **Data area**

Select the memory area for retentive Range 1. (V area or Counter area)

For counters, only the current count values can be retentive.

• **Start**

Assign the start byte address of Rang 1.

• **Length**

Assign the length of Rang 1, unit: byte.

■ **Range 2**

■ **Range 3**

■ **Range 4**

Please refer to the information described above.

As shown in Figure 4-5, the data stored in Range 1 (%VB0 to %VB9), Range 2 (%VB100 to %VB199), Range 3 (C0 to C9) and Range 4 (C20 to C49) will be retentive on power loss.

4.3.4.2 Parameters of the DI Module

You can set the parameters of a DI module as follows:

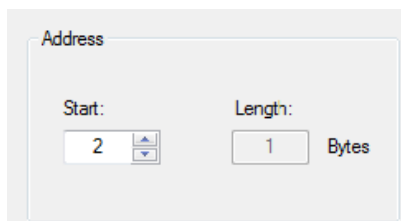


Figure 4-6 Parameters of the DI Module

■ Address

• Start

Enter the start byte address of the address range of this module in I area. The addresses for this module's channels are based on this start address.

• Length

The length of this module's address range. This value is fixed, and it depends on the number of this module's DI channels.

As shown in Figure 4-6, the module has 8 DI channels, and its start address is %IB2, so the addresses of its channels are %I2.0 to %I2.7.

4.3.4.3 Parameters of the DO Module

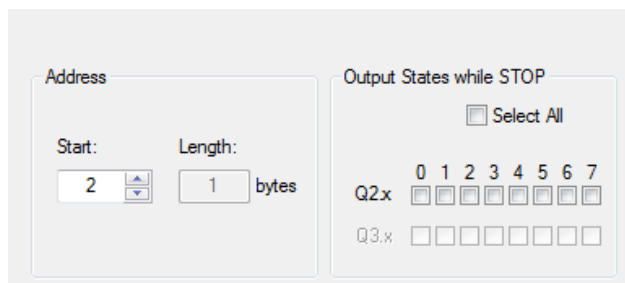


Figure 4-7 Parameters of the DO Module

■ Address

• Start

Enter the start byte address of the address range of this module in Q area. The addresses for this module's channels are based on this start address.

• Length

The length of this module's address range. This value is fixed, and it depends on the number of this module's DO channels.

As shown in Figure 4-7, the module has 16 DO channels, and its start address is %QB2, so the addresses of its channels are %Q2.0 to %Q3.7.

■ Output States while STOP

- Here you can set the digital outputs in a known state while the CPU stops. If the checkbox for an output is checked, the output shall be set to ON (1) while the CPU stops. The default state of a output while the CPU stops is OFF (0).

4.3.4.4 Parameters of the AI Module

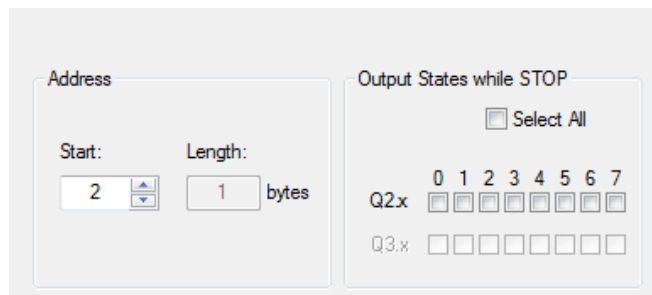


Figure 4-8 Parameters of the AI Module

■ Address

• Address

Enter the start byte address (address of the first channel) of this module in AI area; the addresses for the other channels are based on this start address, each addresses occupies two bytes. This numerical value must be even.

• Length

The length of this module's address range. This value is fixed, and it depends on the number of this module's AI channels.

As shown in Figure 4-8, the module has 4 AI channels, and its start address is %AIW0, so the addresses of the other channels are %AIW2, %AIW4 and %AIW6.

■ Inputs

• Function

Select a measurement type for a channel, e.g. 4-20mA, 1-5V, etc.

Please refer to [6.1.4 Internal Presentation Format of the Measured Values of Signals](#) in “Hardware Manual” for the representation of the measured value.

• Filter

Select a software filter for a channel. As for the analogue signal with rapid changes, a filter can be helpful to stabilize the measured value. *Notice: If the control system requires responding to an AI signal quickly, the software filter of the corresponding channel should be disabled.*

You can assign one of the following filters for a channel:

No --- The software filter is disabled.

Arithmetic Mean --- The filtered value is the arithmetic mean value of a number of samples of the input.

Sliding Mean --- The filtered value is the sliding mean value of a number of samples of the input.

4.3.4.5 Parameters of the AO Module

Address:	8	Length:	8	bytes
Channel 0:	[0,20]mA	Filter:	None	
Channel 1:	[0,20]mA	Filter:	None	
Channel 2:	[0,20]mA	Filter:	None	
Channel 3:	[0,20]mA	Filter:	None	

Figure 4-9 Parameters of the AO Module

■ Address

• Address

Enter the start address (address of the first channel) of this module in AQ area; the addresses for the other channels are based on this start address, each addresses occupies two bytes. This numerical value must be even.

• Length

The length of this module's address range. This value is fixed, and it depends on the number of this module's AO channels.

As shown in Figure 4-9, the module has 2 AQ channels, and its start address is %AQW0, so the address of

another channel is %AQW2.

■ Outputs

• Function

Select a type of output signal for a channel, e.g. 4-20mA, 1-5V, etc.

Please refer to [7.1.4 Internal Presentation Format of Signal Value](#) in “Hardware Manual” for the representation of the output value.

• Freeze Output while STOP

Select whether to set the analog output to a known value (**Freeze Value**) while the CPU stops. If the checkbox for an output is checked, the output shall keep at the freeze value while the CPU stops.

• Freeze Value

Here you can enter a value which the analog output shall keep at while the CPU stops.

4.4 The Initial Data Table

In the Initial Data Table, you can assign initial numerical values for BYTE, WORD, DWORD, INT, DINT and REAL variables in V area. The CPU module processes the Initial Data once at power on and then starts the scan cycle. The Initial Data Table is as Figure 4-10.

Initial Data					
	Address	Value	Value	Value	Value
1	%VB0	B#1	B#11	B#11	
2	%VW12	2	22	222	
3	%VD122	DI#3	DI#33	DI#33333333	
4	%VR322	4.4	4.44	4.444	

Figure 4-10 the Initial Data Table



NOTE: The memory area of “Initial Data”, “Data Maintain” and “Data Backup” should be placed to avoid overlap. The data should be recovered after CPU powered on. The sequence is: recover the memory data defined in “Data Maintain”, assign initial value to the memory are of “Initial Data” and recover the data permanently saved by commands.

4.4.1 Opening the Initial Data Table

- Double-click the [**Initial Data**] node in the **Manager** window.
- Right-click the [**Initial Data**] node, and then select the [**Open**] command on the pop-up menu.

4.4.2 Editing a Cell

Click on a cell to make it change to the editing mode, and now you can type the desired data. Besides, you can use the **UP**, **DOWN**, **LEFT** and **RIGHT** arrow keys to move the focus from one cell to another, and the cell that gets the focus shall change to the editing mode.

When a cell loses focus, its contents are confirmed. Besides, you can use the **ENTER** key to confirm your work and move the focus to the next cell.

The illegal data shall turn red.

4.4.3 Making Initial Data Assignments

The table has 5 columns: an **Address** column and 4 **Value** columns.

- Enter a direct variable, i.e. a direct address in the **Address** column.
- Enter numerical values in the **Value** columns. You can enter one value or multiple values. If you enter multiple values, KincoBuilder shall make an implicit address assignment.

As shown in Figure 4-10, Row 1 indicates that B#1 is assigned to %VB0 and B#2 is assigned %VB1; Row 2 indicates that 2, 3 and 4 are assigned to %VW10, %VW12 and %VW14 respectively; Row 3 indicates that DI#100, DI#200, DI#2000 and DW#2456 are assigned to %VD100, %VD104, %VD108 and %VD112 respectively.

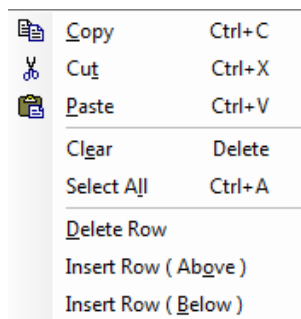
4.4.4 Editing the Initial Data Table

- Sorting

Click the **Address** column header to sort the table.

- The Pop-up Menu

Right-click on any cell in the table, the following menu will popup:



- **Delete Row:** Delete the row in which the focus is located.
- **Insert Row (Above):** Insert a new blank row above the row in which the focus is located.
- **Insert Row (Below):** Insert a new blank row below the row in which the focus is located.

Please pay attention when using the paste command: it will not work between different types of table' neither between different rows.

4.5 The Global Variable Table

The Global Variable Table is composed of two parts: the **Global Variable** tab and the **FB Instance** tab.

➤ The **Global Variable** tab

It can be used to define the Global Variable, which accesses PLC memory address.

The Global Variable in the programme can replace PLC memory address to ensure the readability of the programme. Each memory address can be assigned one symbolic variable name; similarly, one symbolic variable name will have one corresponding memory address only.

Please refer to [3.3.1 How to define an identifier](#) to see the defining rule;

Please refer to [3.5 Variables](#) to see more information about the global variables.

You can declare global symbolic variables here, as shown in Figure 4-11.

In this manual, “the Global Variable Table” usually indicates this tab.

	Symbol	Address	Data Type	Comment
1	bOpenDoor	%V0.0	BOOL	Open the door
2	fWenDu	%VR44	REAL	Weather
3	iCnt	%VW66	INT	Step counter
4				
5				
6				

Figure 4-11 the Global Variable tab

➤ The **FB Instance** tab

	Instance	FB	Position
1	C55	CTU	MAIN
2	C66	CTU	MAIN
3	C255	CTU	MAIN
4	T11	TON	MAIN
5	T55	TON	MAIN
▶ 6	T254	TON	MAIN

Figure 4-12 the FB Instance tab

As mentioned in [3.6.6 Using of FB Instances](#), the FB instances are declared by KincoBuilder automatically to facilitate the users. So all the information here is only for reference and you cannot modify them.

4.5.1 Opening the Global Variable Table

There are three ways to open the Global Variable Table:

- Double-click the [**Global Variable**] node in the **Manager** window.
- Right-click the [**Global Variable**] node, and then select the [**Open**] command on the pop-up menu.
- Select the [**Project**]>[**Global Variable**] menu command.

4.5.2 Declaring the Global Variables

The table has 4 columns: **Symbol**, **Address**, **Data Type** and **Comment**.

- Open the Global Variable Table window and select the **Global Variable** tab.

- Enter the symbol name in the **Symbol** column and confirm it.
- Enter the direct address in the **Address** column and confirm it.
- Choose a data type from the drop list in the **Data Type** column.
- (Optional) Enter a **Comment**.

If you declare a global variable in the Global Variable Table, you can use it in any POU, and a direct address is equivalent to its symbolic name in the user program.

Please refer to [3.5 Variables](#) for more information about the global variable.

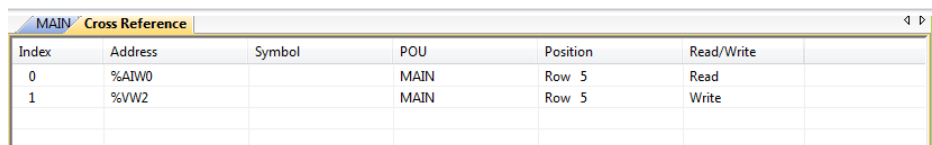
You can operate the Global Variable Table just as the Initial Data Table. Please refer to [4.4 The Initial Data Table](#) for more information.

4.6 The Cross Reference Table

The Cross Reference Table shows all the variables used in the project, and identifies the POU, network or line location, and how to access the operands (read or write to). The Cross Reference Table is helpful when you want to know if a symbolic name or an address is already in use, and where it is used.

Information in the Cross Reference Table only be generated after the first compilation, and will refresh automatically after each compilation.

The Cross Reference Table is as the following figure:



Index	Address	Symbol	POU	Position	Read/Write
0	%AIW0		MAIN	Row 5	Read
1	%VW2		MAIN	Row 5	Write


Figure 4-13 the Cross Reference Table

- **Address** Display all the memory addresses used in the project.
- **Symbol** Display the global symbolic name of the **Address**.
- **POU** Indicate the POU where the **Address** is used.
- **Position** Indicate the line or network where the **Address** is used.
- **Read/Write** Indicate whether the **Address** is read or written to here.

As shown in Figure 4-13, the first row in the table indicates that **%M1.3** is used once in **Network 0** of the **Main** program, and it is read this time.

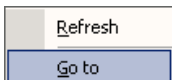
Double-click on a row in the Cross Reference Table, and you shall go to the corresponding part of your program.

4.6.1 Opening the Cross Reference Table

- Select the [**Project**]>[**Cross Reference**] menu command.
- Click the icon  in the toolbar.
- Use the **Alt+C** shortcut key.

4.6.2 The Pop-up Menu

Right-click on any row in the table, the following menu shall popup.



- **Refresh:** Refresh the table and display the latest cross-reference information.
- **Go to:** Go to the corresponding part of your program.

4.7 The Status Chart

You can use the Status Chart to monitor and force any direct variable used in the project after you have downloaded the project to the PLC. The Status Chart is shown as Figure 4-14.

➤ Memory Monitor Chart

Can be used to detect any memory address of PLC;

This function is available on K3 and K5 PLC; on K5 all memories are available while on K3 only I, Q, AI, AQ, M, V areas.

The Monitor Chart is shown as follows:

Initial Data VAR_GLOBAL MAIN * Status Chart							
	Address	Number	Format	Value 1	Value 2	Value 3	
1	%V0.0	3	DEC	FALSE	FALSE	FALSE	
2							
3	%MB6	7	DEC	B#0	B#0	B#0	
4	%MB9			B#0	B#0	B#0	
5	%MB12			B#0			
6							
7	%VR44	9	DEC	0	0	0	
8	%VR56			0	0	0	
9	%VR68			0	0	0	
10							
11	%SMW66	5	DEC	0	0	0	
12	%SMW72			0	0		
13							
14							
▶ 15	<input type="text"/>						
16							
17							
18							

You may find in the chart:

[Memory Address]: Input the initial address of the memory area to be detected;

[Monitor Length]: Input the total number of data to be detected from the memory addresses. Maximum number is 150.

For each row maximum number of data shown is 3. If more than 3, data will be separate into different rows.

[Display Format]: Select the display format, including decimal and hexadecimal system.

[Memory Value]: Show the memory detected. If bit memory is detected, only TRUE or FALSE will be shown.

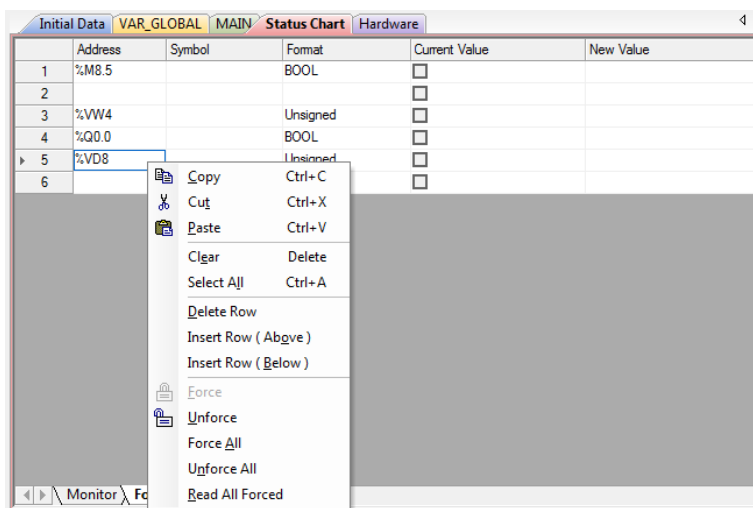


Figure 4-14 the Status Chart

- **Address** Enter the initial address to be monitored and forced.
- **Symbol** Display the global symbolic name of the **Address**.
- **Format** Choose a display format for the current value and new value.
(BOOL; REAL; Signed, Unsigned, Hexadecimal or Binary)
- **Current value** Display current values of the **Address** from the PLC.
- **New Value** Enter the value to be forced for the **Address** when monitoring

You can open a Status Chart to edit it, but no status information is displayed in the **Current Value** column unless you select the **[Monitor]** command from the **[Debug]** menu or toolbar.


In order to be efficient, KincoBuilder only allows monitoring and forcing the variables used in the project. If you enter the variables that are not used, the **Current Value** and **New Value** won't take effect.

4.7.1 Opening the Status Chart

- Double-click the [**Status Chart**] node in the **Manager** window.
- Right-click the [**Status Chart**] node, and then select the [**Open**] on the pop-up menu.
- Select the [**Debug**]>[**Status Chart**] menu command.

4.7.2 Monitoring the Variable Value

You may monitor the variable value by Status Chart as follows:

- Input the memory address to be monitored in [Address]
- Enter into online monitoring status via following means:
 - Execute [**Debug**]→[**Online Monitor**] command;
 - Click the icon  on the toolbar;
 - Use shortcutkey **F6**
- [**Display Format**] of the monitor value can be changed at any time.

4.7.3 The Force Function

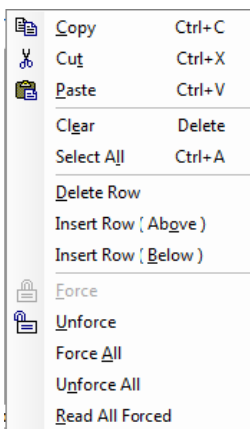
You can use the Force Function to edit the variable value in I area, Q area, M area, V area, AI are, AQ area, among which variables in I area, Q area, M area and V area can be forced by bit, byte, word or double byte; variables in AI area and AQ area can be forced by word. When CPU is rebooted, all force status will be canceled.

K5 allows maximum 32 variables. Immediate commands cannot be forced.

In at certain time of the scanning period, one variable may have following possibilities: exterior input signal (I, AI) or user program result (Q, AQ, M, V). Variable valuing principles are as follows:

- Variables in M area and V area, force value has the same priority of the program execution result.
- Variables in I and AI area, force value is prior to exterior signal input value.
- Variables in Q and AQ area, program execution result will overwhelm.


4.7.4 Right-click Menu




- Force: Input [Force Value] the direct memory (address) of PLC.
- Cancel Force: Cancel the Force status of single being clicked.
- Force All: Input all forced value of [Force Values] to the corresponding direct memories (address) of PLC.
- All Cancel Force: Cancel all the Force status.
- Read All Forced: Read all forced variables and show them in the Variable Status Table.

4.7.5 Force and Cancel Force

You may force or cancel force a variable in the Variable Status Table as follows:

- Input the direct memory address to be forced in [Address];
- Select the [Display Format] of the value and you may verify it at any time (optional);
- Input the force value. You may input a round number in decimal and Kincobuilder will adjust it accordingly;
- You may force a variable as follows:
- Right-click the row and execute [Force] command;
- Click the row and click icon  in the tool bar;

- Click the row and execute [Debug] → [Force] command;
- You may cancel forcing a direct [Address] in a row as follows:
- Right-click the row and execute [Cancel Force] command;
- Click the row and click the icon  in the toolbar
- Click the row and execute [Debug] → [Cancel Force]

The edition of Variable Status Table is the same as Initial Data, please refer to that part.

4.8 Password Protection

The Kinco-K5 provides password protection for you to encrypt the CPU for restricting access to specific functions. If a CPU is encrypted, the password will be required to enter when you try to access the restricted functions. Here, if a correct password is entered, the CPU will permit the corresponding operation; if a wrong password is entered, the CPU will refuse the corresponding operation. The password is only valid for current operation. If you try to access the restricted functions again, then you have to enter the password again.

A password is a string of letters, digits, and underline characters, and it is case-sensitive. The maximum length of a password is 8 bits.

If you upgrade the PLC protect level to “Grade 3: Highest Protection” and set password or enable “Forbid Upload”, the user programme in PLC will be encrypt

4.8.1 Protection Privileges

The Kinco-K5 provides the following 3 protection privileges:

- **Level 1:** Full access. No restriction to access all the functions. This is the default level.
- **Level 2:** Partial access. Password is required while downloading.
- **Level 3:** Minimum access. Password is required while downloading and uploading.

4.8.2 How to change the password and the protection level

Select [PLC]>[Password...] menu command to open the 'Password' window. See the following figure:

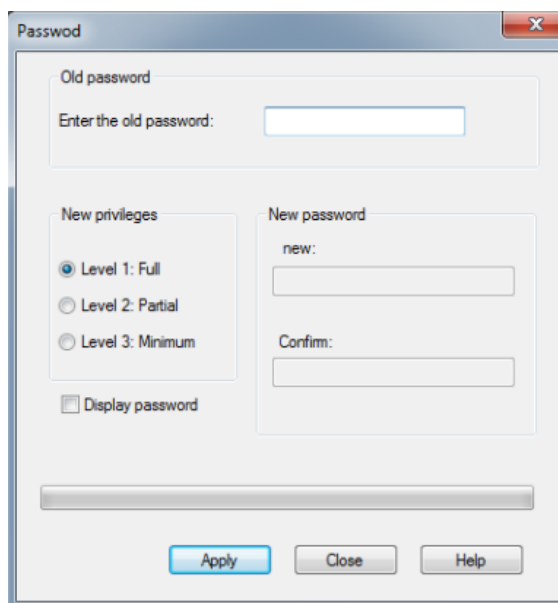


Fig. 4-15 the 'Password' Window

■ Old password

If the connected CPU has been set with password protection, then the original old passwords has to be entered here for verification. If no password protection has ever been set, then just leave the edit box empty.

➤ New Privileges

Here, you can set the new protection levels and passwords for the connected CPU.

- **New Privileges:** You can choose any one from level 1, level 2, and level 3.
- **New password:** You can enter a new password here.
- **Confirm:** You need to enter the new password again here.

After finishing the settings above, you can click on the [**Apply**] button to write the new settings into the connected CPU, and then the new settings will be efficient.

4.8.3 How to recover from a lost password

If you forget the password, you have to clear the memory of the CPU for continuing to use it. Select **[PLC]>[Clear...]** menu command to clear the memory of the CPU.

After clearing, all the data in the CPU, including the user program, the configuration data, and the password, will be lost, and the CPU is restored to the factory-set defaults, except for the RTC. Here, the communication parameters are the following: the station number 1, the baudrate is 9600, no parity, 8 data bits, 1 stop bit.

Chapter V How to Use KincoBuilder ... Programming

KincoBuilder presently supports IL and LD programming languages, and so two editors are provided for programming: the IL editor and the LD editor. This chapter will detailedly describes the two editors and meanwhile represents the relevant syntaxes and rules of IL and LD languages.

IEC61131-3 defines three textual languages and three graphical languages. The textual languages include: Instruction List (IL), Structured Text (ST) and Sequential Function Chart (SFC, textual version); and the graphical languages include: Ladder Diagram (LD), Function Block Diagram (FBD) and Sequential Function Chart (SFC, graphical version).

KincoBuilder presently provides two editors for programming: the IL editor and the LD editor. You can write a POU in IL or LD language, i.e. you can write a POU with the IL or LD editor. With some restrictions, a POU written in a program editor can be viewed and modified in another program editor. You just select the **[Project]>[IL]** or **[Project]>[LD]** menu command to switch the editor for the current POU.

5.1 Programming in IL

5.1.1 Overview

IL is a low level language that is very similar with the assembly language, and it is based on similar instruction list languages from well-known PLC manufacturers around the world.

IL is close to a machine code, and so it is an efficient language. IL is very appropriate for experienced programmers. Sometimes you can use IL to solve the problems that you cannot solve easily using LD.

5.1.2 Rules

5.1.2.1 Instructions

IL is a line-oriented language. An IL program consists of a sequence of instructions. Each instruction shall begin on a new line and contains an operator. Operands are optional, and they are separated by commas or spaces. A comment can be entered at the end of the line using parentheses and asterisks. Blank lines are allowable in an instruction list.

The following figure shows the typical format of an IL statement:

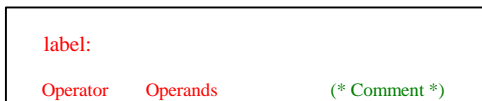


Figure 5-1 The Typical Format of an IL Statement

➤ **label**

Optional. Jump is used to jump to a line of the IL program. In this case, a label in front of the destination line is used. The name format of a label is identical with that of an identifier.

➤ **Operator**

➤ **Operands**

Please refer to instructions set for the detailed descriptions.

➤ **Comment**

Optional. Only one comment is allowable in a line; nesting is not permitted.

The following is an example:

(* NETWORK 0 *)

begin: (* a label,used at jump *)

LD %I1.0

TP T2, 168 (* if %I1.0 is true, the timer T2 is started. T2 is an instance of TP. *)

5.1.2.2 Current Result

IL provides a universal accumulator called the “Current Result (CR)”, and the current result of logical operation is stored in the CR. The CR will be refreshed after the execution of each statement, and it may act as the execution condition or one of the operands for the next statement.

All the operators in KincoBuilder can be grouped according to their influence on the CR as shown in the following table. Please refer to the instruction set for further details.

Group	Influence on the CR	Examples
C	Create the CR	LD, LDN
P	Set the CR to be the result of operation	Bit logic, Compare instructions, etc.
U	Leave the CR unchanged	ST, R, S, JMP, etc.

Table 5-1 The Operator Groups



IEC61131-3 does not define the above groups. As a result, these groups in different programming systems may be different.

5.1.2.3 Network

In KincoBuilder, a POU is composed of the following parts:

- POU type and POU name
- Variable declaration part
- Code part containing the instructions

Network can be taken as the basic code segment; the code part of the POU is composed of several networks.

Networks make it easier to view an IL program. A typical network includes:

- Network label
- Network comment.
- Instructions

5.1.3 The IL Editor in KincoBuilder

When a new program in IL language is being established, the IL editor will be ready for programming; if an IL program is opening, the IL editor will also be ready. The IL editor is shown as follows.

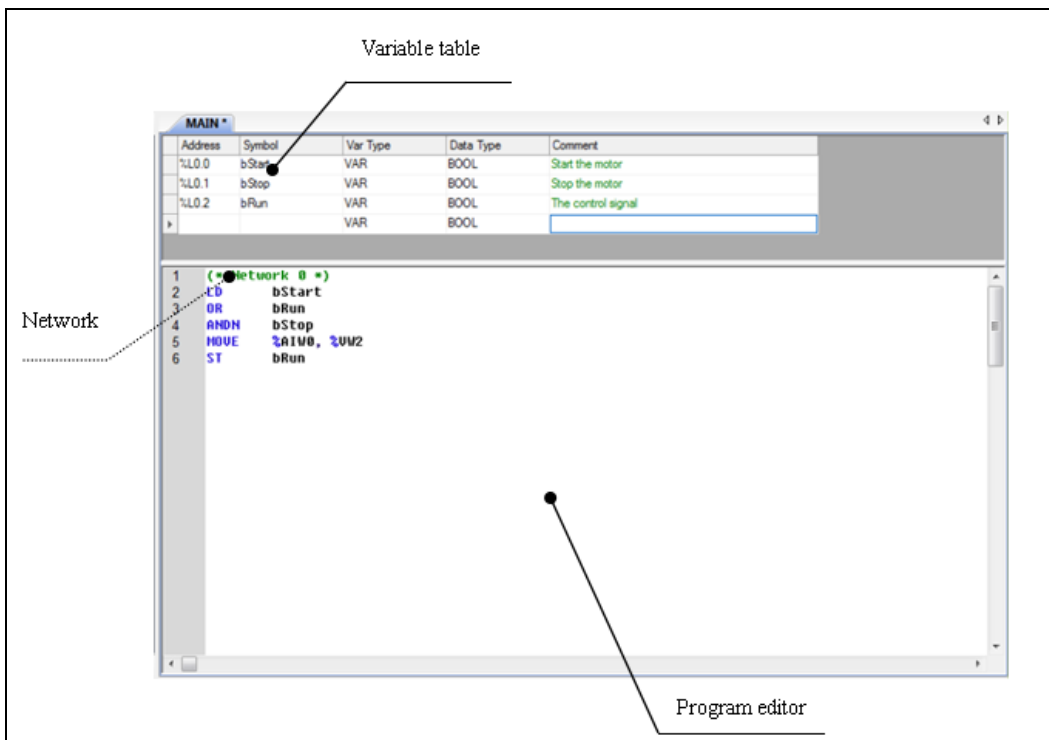


Figure 5-2 the IL Editor

The IL editor is composed of two parts:

- The Variable Table: you can declare the local variables and input/output parameters of the POU here.
- The Program Editor: you can edit your control program here.

5.1.3.1 Adding a Network

Use one of the following ways to add a network:

- Use **Ctrl+Q** shortcut key

- Right-click the Program Editor and select the **[Insert Network]** on the pop-up menu.

5.1.3.2 Allowable Instructions Format in a Network

- There can be only one statement label in a network. For example:

```
(* NETWORK 0 *)
```

```
MRun:    (* There can be only one statement label *)
```

- A network can contain some statements.

In [5.2.2.2 Current Result](#), we divide all the instructions three groups (“C”, “P” and “U”).

The network must begin with one of the instructions in group “C”, and end with one of the instructions in group “P” or “U”. For example:

```
(* NETWORK 0 *)
```

```
LD  %M3.5  (*Begin with LD instruction *)
```

```
... ..    (*you can enter other instructions *)
```

```
ST  %Q2.3  (*End with the allowable instruction *)
```

- A network can contain some statement labels and some statements.

The network must begin with a label or one of the instructions in group “C”, and end with one of the instructions in group “P” or “U”. For example:

```
(* NETWORK 0 *)
```

```
MRun:
```

```
LD  %M3.5  (*Begin with LD instruction *)
```

```
... ..    (*You can enter other instructions*)
```

```
ST  %Q2.3  (*End with the allowable instruction *)
```

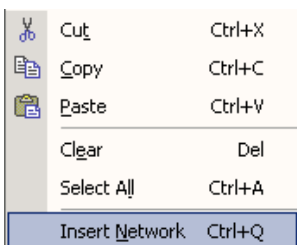
5.1.3.3 Other Operations

The IL editor can automatically format the statements. It can also check the statements automatically, and a red question mark (?) before a line indicates that there is something wrong with this line.

The IL editor is similar with a text editor and supports common keyboard operations.


All commands in the [Edit] menu are applicable in the IL editor.

Right-click on the Program Editor, the following menu will popup:



5.1.3.4 Online Monitoring

After the [Debug]>[Monitor] menu command is selected, the IL editor will change to the online monitoring mode. In this mode, you are not allowed to edit the program.

In the online monitoring mode, the original Program Editor area is divided into two columns by a vertical line in the middle, with the right column displaying the program and left column displaying the corresponding variables. When moving the cursor onto the vertical line, it will turn into . Then drag the line to the left or right to change the sizes of the columns.

5.1.3.5 Example

```
(* NETWORK 0 *)  
LDN      %M0.0
```

```
TON    T0, 1000    (*Start T0 with the output of T1, timing: 1000*1ms *)
ST     %M0.1
LD     %M0.1
TON    T1, 1000    (*Start T1 with the output of T0, timing: 1000*1ms *)
ST     %M0.0

LD     %M0.1
ST     %Q0.0      (* Output square wave with 2s period at %Q0.0 *)
```

5.1.4 Converting IL Program to LD Program

You can select the [Project]>[LD] menu command to change the editor to the LD editor; at the same time, the current IL program shall be converted to LD format.


Not all IL programs can be converted to LD format; the successful conversion must satisfy the following conditions:

- There is no error in the source IL program.
- The source IL program must be strictly in line with the following rules:
- Each network must begin with one of the instructions in group “C”; or there must be only one statement label in a network.
- The instruction which the network begins with must be used only once in the network.
- Each network must end with one of the instructions in group “P” or “U”.

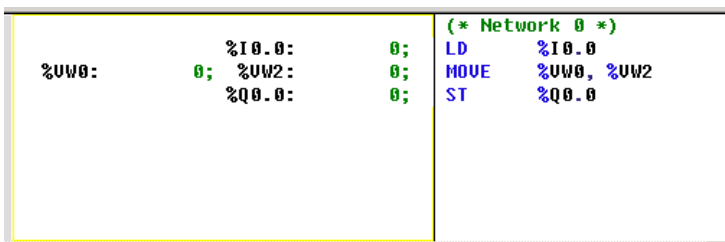
5.1.5 Debug and Monitor the Program

5.1.5.1 Online Monitor IL

You may enter Online Monitor Status with the IL editor by any means below:

- Execute [Debug] → [Online Monitor] commands;
- Click icon  in the tool bar;
- Shortcut key F6.

In the Online Monitor Status, the edition area will be divided into two columns; the right column is the program and left is corresponding variables. The columns are separated by a line, which can be dragged to change the space of each column.

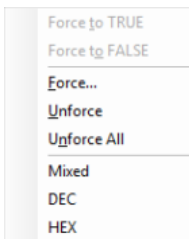


NOTE: The program cannot be edited when in the Online Monitor Status.

5.1.5.2 Force Specific Variables

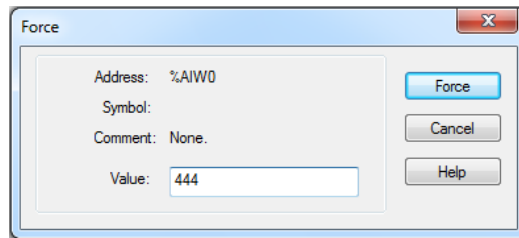
You may find detailed description in [4.7. 3 The Force Function](#).

When online monitoring IL, you may execute force or cancel force to specific variables in IL editor; right-click any variable and the menu will pop out (if a Non On/Off Variable is right-clicked, commands [**Force to be TRUE**] and [**Force to be FALSE**] will be invalid):



- Force to be TRUE: Force the value of the variable (On/Off Variable) to be 1 (TRUE)
- Force to be FALSE: Force the value of the variable (On/Off Variable) to be 0 (FALSE)

- Force to be ...: If you select this command, a dialogue will pop out



You can input the value into the **[Force Value]** box and click **[Force]**. You may refer to [3.5 Variable](#).

5.2 Programming in LD

Some definitions are from IEC 61131-3 standard.

5.2.1 Overview

LD (Ladder Diagram) is one of the most frequently used graphical languages in PLC programming. LD language is based on the traditional relay ladder logic. In addition, the IEC LD language allows the use of user defined function blocks and functions and so can be used in a hierarchical design. LD allows you to program by means of standardized graphic symbols, so it is easy to learn and use. LD shows great advantages in handling Boolean logic. The following is a simple program segment in LD.



Figure 5-3 A Sample in LD

5.2.2 Network

When you write a program in LD, you can use standardized graphic symbols and arrange them to construct a network of logic. LD network shall be delimited **错误!未找到引用源。** on the left by a vertical line known as the *left power rail*, and on the right by a vertical line known as the *right power rail*. The state of the left rail **错误!未找到引用源。** shall be considered ON all along. No state is defined for the right rail.

5.2.3 Standardized graphic symbols

➤ Link

Horizontal link and vertical link are used in LD, corresponding to serial connection and parallel connection respectively. The link state may be ON or OFF, corresponding to the Boolean values **错误!未找到引用源。** 1 or 0 respectively. The term *link state* shall be synonymous with the term *power flow* **错误!未找到引用源。** .

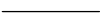

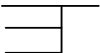
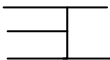
Symbol	Name	Description
	Horizontal link	A horizontal link element shall be indicated by a horizontal line. It transmits the state of the element on its immediate left to the element on its immediate right.
	Vertical link (With attached horizontal links)	<p>The vertical link element shall consist of a vertical line intersecting with one or more horizontal link elements on each side.</p> <p>The vertical link state shall represent the inclusive OR of the ON states of the horizontal links on its left side, that is, the vertical link state shall be:</p> <ul style="list-style-type: none"> - OFF if the states of all the attached horizontal links to its left are OFF; - ON if the state of one or more of the attached horizontal links to its left is ON. <p>The state of the vertical link shall be copied to all of the attached horizontal links on its right.</p>
		
		

Table 5-2 Link elements

➤ Contact

A *contact* is an element which imparts a state to the horizontal link on its right side which is equal to the Boolean AND **错误!未找到引用源。** of the state of the horizontal link at its left side with an appropriate function of an associated Boolean **错误!未找到引用源。** **错误!未找到引用源。** variable **错误!未找到引用源。**

A contact does not modify the value of the associated Boolean variable.

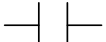
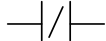
Symbol	Name	Description
<p>***</p> 	Normally open contact	The state of the left link is copied to the right link if the state of the associated Boolean variable (indicated by "***") is ON. Otherwise, the state of the right link is OFF.
<p>***</p> 	Normally closed contact	The state of the left link is copied to the right link if the state of the associated Boolean variable is OFF. Otherwise, the state of the right link is OFF.

Table 5-3 Contacts

➤ Coil

A *coil* writes the state of the left link into the associated Boolean variable.

Symbol	Name	Description
*** —()—	Coil	The state of the left link is copied to the associated Boolean variable and to the right link.
*** —(/)—	Negated coil	The inverse of the state of the left link is copied to the associated Boolean variable, that is, if the state of the left link is OFF, then the state of the associated variable is ON, and vice versa.
*** —(S)—	SET (latch) coil	The associated Boolean variable is set to the ON state when the left link is in the ON state, and remains set until reset by a RESET coil.
*** —(R)—	RESET (unlatch) coil	The associated Boolean variable is reset to the OFF state when the left link is in the ON state, and remains reset until set by a SET coil.

Table 5-4 Coils

➤ Execution control elements

Transfer of program control in the LD language shall be represented by the graphical elements shown in the following table.

Symbol	Name	Description
┆—(1)—<RETURN>	Conditional Return	Program execution 错误!未找到引用源。 shall be transferred back to the invoking entry when the horizontal link state to its left is 1 (TRUE), and shall continue in the normal fashion when the Boolean input is 0 (FALSE).
┆>> Label	Unconditional Jump	Program execution 错误!未找到引用源。 shall be transferred to the designated network label 错误!未找到引用源。 错误!未找到引用源。 unconditionally.

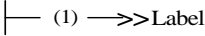
	Conditional Jump	Program execution 错误!未找到引用源。 shall be transferred to the designated network label 错误!未找到引用源。 when the horizontal link state to its left is 1 (TRUE), and shall continue in the normal fashion when the Boolean input is 0 (FALSE).
-----------------------------------------------------------------------------------	---------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 5-5 Execution control elements



Notice: (1) indicates that here is the graphical code whose result is Boolean.

➤ Functions and function blocks

A function or a function block shall be represented with a rectangular block, and its actual variable connections can be shown by writing the appropriate variable outside the block adjacent to the formal variable name on the inside. At least one Boolean input and one Boolean output shall be shown on each block to allow for power flow 错误!未找到引用源。 through the block.

The function shall have a Boolean input named *EN* and a Boolean output named *ENO*. *EN* is used to control the execution of this function. If *EN* is true, the function will be executed and *ENO* will be set as true. If *EN* is false, the function will not be executed and *ENO* is to be set as false.



Figure 5-4 Functions and Function Blocks

5.2.4 The LD Editor in KincoBuilder

When a new program in LD language is being established, the LD editor will be ready for programming; if an LD program is opening, the LD editor will also be ready. The LD editor is shown as follows.

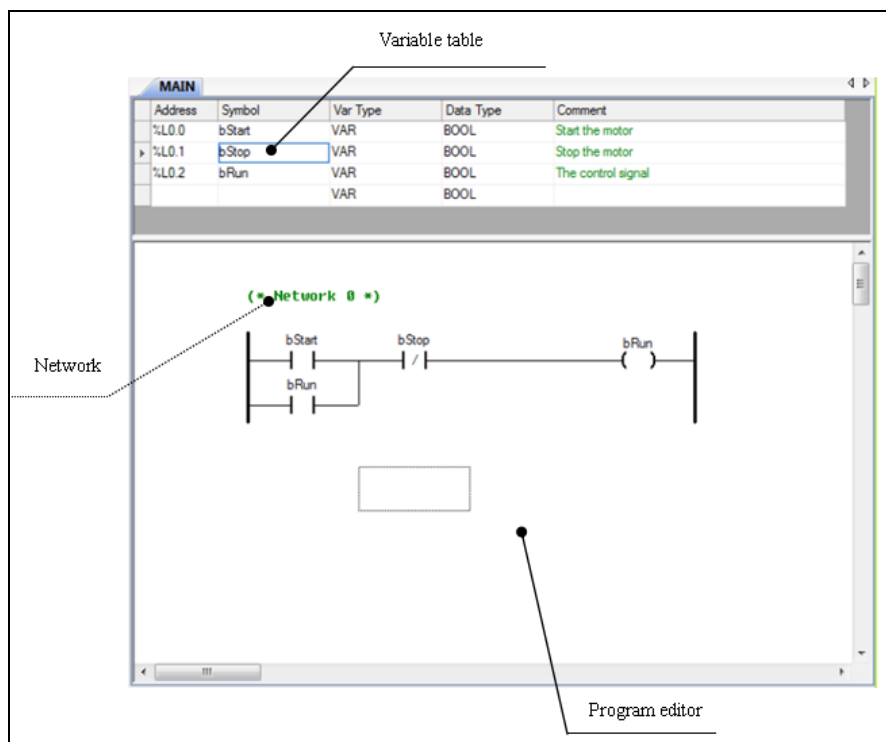


Figure 5-5 the LD Editor

5.2.4.1 LD Program Limits

Max. 200 networks are allowed in a LD program.

You can regard the Program Editor window as a canvas divided into cells. Inside that canvas, a network can extend max. 32 cells horizontally and max. 16 cells vertically. So the maximum number of the elements horizontally in a network are as follows: if there are only coils and contacts, up to 31 contacts and 1 coil; if only with functions/function blocks, up to 12 blocks, 1 coil and 1 contact. In addition, in a network, the branches shall not exceed 16 in a parallel connection.

Parallel connection of two or more independent functions/function blocks is forbidden.

5.2.4.2 Common Operations

The LD editor supports common mouse operations:


- Click an element, then it shall be selected and the focus moves on it (a rectangular frame appears on the element);
- Double-click an element, then its property dialog box shall pop up, and there you can modify the element's properties;
- Right-click an element, then the context menu shall pop up, and you can select the menu command to execute the corresponding function.

In addition, the LD editor supports keyboard operations:

- Use **UP**, **DOWN**, **LEFT** and **RIGHT** arrow keys to move the focus.
- Press **ENTER** key to select the element's parameter area for entering.
- Press **Del** key to delete the element on which the focus is located.
- There is a shortcut key corresponding to each menu command.

5.2.4.3 LD Programming Steps

The following description will focus on mouse operations.

- Use one of the following ways to add a network:
- Select the **[LD]>[Network]** menu command
- Click the icon  on the toolbar
- Use the shortcut key **Ctrl+W**
- Right-click any element, and select the **[Network]** command on the pop-up menu

The network just added is as follows.



Figure 5-6 A New Network

Double-click the network label to open the comment dialog box, and you can enter some comments here to give a description for this network.

➤ When you add an instruction, its variables are initially denoted by red question marks (???). These question marks indicate that the variable is undefined, and you must define it before compiling the program.

When you click a variable, a box appears to indicate the variable area, and you can enter the desired variable or constant in this box. You can also press **ENTER** key to select the variable area for the element on which the focus is located. The LD Editor shall automatically format the direct address after you enter it, so you need not enter the percent mark if you enter a direct address.

In addition, you can double-click a contact or coil element to open its property dialog box to modify its type and parameters. The following figure shows a contact property dialog box.

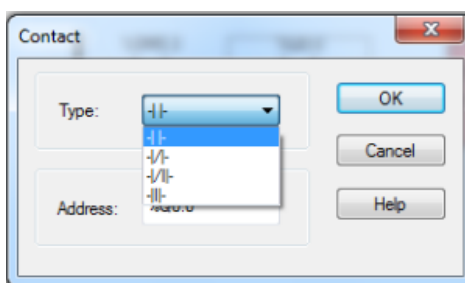











Figure 5-7 A Contact Property Dialog Box

➤ Click an element and select it as the reference, then continue to add other elements using one of the following ways:

➤ Use the **[LD]** menu commands or shortcut keys:

LD	PLC	Debug	Tools	Window
	Network		Ctrl+W	
	Left Contact		Ctrl+L	
	Right Contact		Ctrl+E	
	Parallel Contact		Ctrl+U	
	Block		Ctrl+B	
	Coil		Ctrl+D	
	Branch		Ctrl+H	
	Delete			
	Delete Network		Ctrl+Delete	

Left Contact: Add a contact on the left of the reference element.

Right Contact: Add a contact on the right of the reference element.

Parallel Contact: Add a contact parallel to the reference contact.

Block: Add a serial block (Function/FB/Subroutine).

Coil: Add a coil parallel with the reference coil.













Branch: Draw a branch parallel to other elements.

Delete: Delete the selected element.

Delete Network: Delete the network where the selected element is located.

➤ Use the context menu commands:

Right-click an element, then the following context menu pops up. Please refer to the above descriptions.

	Cu <u>t</u>	Ctrl+X
	C <u>o</u> py	Ctrl+C
	P <u>a</u> ste	Ctrl+V
	Select A <u>l</u> l	Ctrl+A
	Insert N <u>e</u> twork	Ctrl+W
	L <u>e</u> ft Contact	Ctrl+L
	R <u>i</u> ght Contact	Ctrl+E
	P <u>a</u> rallel Contact	Ctrl+T
	B <u>l</u> ock	Ctrl+B
	C <u>o</u> il	Ctrl+D
	B <u>r</u> anch	Ctrl+H
	D <u>e</u> lete	Del
	D <u>e</u> lete Network	Ctrl+Del

- Use the toolbar buttons:



Click the appropriate toolbar button to add a corresponding element.

- Double-click from the LD Instructions tree:

In the LD Instructions tree, expand the tree, find the desired instruction, and double-click on it, then the instruction shall appear in the LD Editor.

Assume that a “MOVE” block is added. Then the network is as follows:

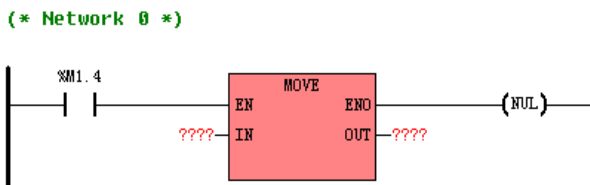


Figure 5-8 Adding other Elements

- Continue to use the mouse or the **ENTER** key to select the variable area to modify the variables of the new elements. In addition, you can double-click on the block elements in the program to open the parameters dialog box to modify the block’s properties.

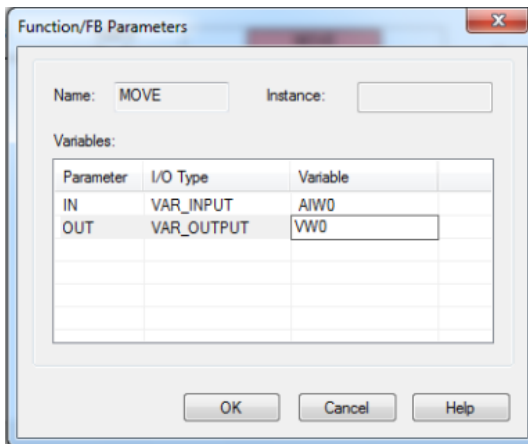


Figure 5-9 The Block Parameters Dialog Box

You can double-click any variable in the [Variable] list to modify it, and then press **Enter** key to confirm the typing. In addition, you can also use **Up** or **Down** arrow keys to select a variable, and press **Enter** key to begin editing, then press **ENTER** key to confirm the typing.

KincoBuilder will strictly check the syntax of your typing, wrong variable shall be denied.

The modified network is shown as follows:

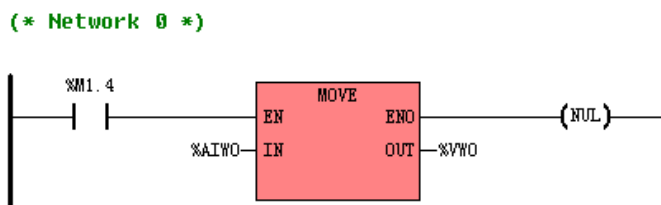


Figure 5-10 The Modified Network

➤ After this network is complete, continue to add and modify new networks until this POU is finished. When adding a new network, if the current network label is selected as the reference, then the new network shall be added above the current network; otherwise, the new network shall be added below the current network. Here the current network means the network where the selected element is located.

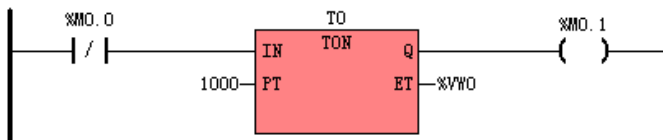
5.2.4.4 Online Monitoring

After the [Debug]>[Monitor] menu command is selected, the LD editor will change to the online monitoring mode.

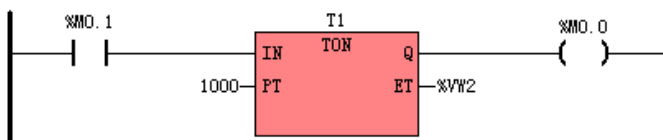
In this mode, all the PLC data status is displayed in the LD Editor window, and you are not allowed to edit the program.

5.2.4.5 Example

(* Network 0 *)
(* Start T0 with the output of T1, timing: 1000*1ms *)



(* Network 1 *)
(* Start T1 with the output of T0, timing: 1000*1ms *)



(* Network 2 *)
(* Output square wave with 2s period at %Q0.0 *)




5.2.5 Monitoring and Debugging the Program

5.2.5.1 Online Monitor of LD Program











NOTE: Online monitor status does not allow any edition to the program.

You can enter into the online monitor status if LD editor is opened.

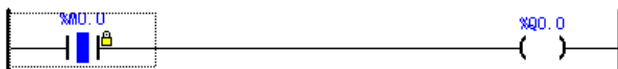
- [Debug]→[Online Monitor]
- Click the icon .
- Shortcut key F6.

In online monitor status variables are shown as:

-   Contacts disconnect,   Contacts connect
-   Coils disconnect,   Coils connect
- Non ON/OFF variables will be shown at the right side.

Example:

(* Network 0 *)

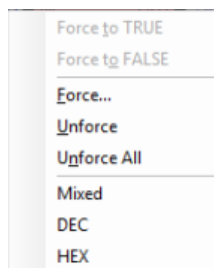


NOTE: Yellow lock is the force value

5.2.5.2 Force Specific Variables

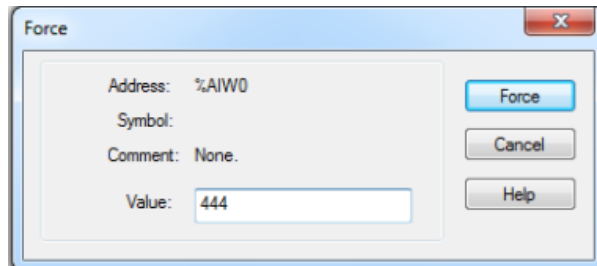
You may find detailed description in [4.7.3 The Force Function](#).

When online monitoring IL, you may execute force or cancel force to specific variables in IL editor; right-click any variable and the menu will pop out (if a Non On/Off Variable is right-clicked, commands [**Force to be TRUE**] and [**Force to be FALSE**] will be invalid):



- Force to be TRUE: Force the value of the variable (On/Off Variable) to be 1 (TRUE)
- Force to be FALSE: Force the value of the variable (On/Off Variable) to be 0 (FALSE)

- Force to be ...: If you select this command, a dialogue will pop out



You can input the value into the **[Force Value]** box and click **[Force]**. You may refer to [3.5 Variable](#).

Chapter VI Kinco-K5 Instruction Set

Kinco-K5 instruction set accords with IEC 61131-3 standard for programming, the basic instructions and most of the standard functions/function blocks are provided. In addition, some non-standard instructions are available to satisfy different users and actual application requirements.

6.1 Summary

In this chapter, detailed introduction and specific application examples of all instructions shall be given. Instructions for LD and IL are to be described.

For LD, *EN* and *ENO* operands are not described in the following sections, because both of them are the same for all the instructions. *EN* and *ENO* are both connected with power flow. *EN* (Enable) is a BOOL input for most of the blocks, and power flow must be valid at this input for the block to be executed. *ENO* (Enable Out) is a BOOL output for most of the blocks; if the block gets the power flow at the *EN* input and the block is executed right, then the *ENO* is set to be “1” and passes power flow to the next element, otherwise power flow shall be terminated here.

For IL, as mentioned in [5.1.2.2 Current Result](#) in the software manual, the CR will be refreshed after the execution of each statement, and it may act as the execution condition or one of the operands for the next statement. This is described detailedly, and the abbreviations of the operator groups are used in this chapter.

6.2 Bit Logic Instructions

6.2.1 Standard Contact

➤ Description

➤

	Name	Usage	Group	
LD	Normally open contact	<i>bit</i> — —		
	Normally closed contact	<i>bit</i> — /—		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX
IL	LD	LD <i>bit</i>	C	<input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
	AND	AND <i>bit</i>	P	<input checked="" type="checkbox"/> CPU508
	OR	OR <i>bit</i>		
	LDN	LDN <i>bit</i>	C	
	ANDN	ANDN <i>bit</i>	P	
	ORN	ORN <i>bit</i>		

Operand	Input/Output	Data Type	Acceptable Memory Areas
<i>bit</i>	Input	BOOL	I, Q, V, M, SM, L, T, C, RS, SR, constant

➤ **LD**

When the *bit* is equal to 1, the Normally Open contact is closed (on) and then power flow is passed to the next element.

When the *bit* is equal to 0, the Normally Closed contact is closed (on) and then power flow is passed to the next

element.

➤ **IL**

The Normally Open contacts are represented by the *LD*, *AND*, and *OR* instructions.

The *LD* instruction loads the *bit* and sets the CR equal to the result.

The *AND* instruction is used to AND the *bit* with the CR, and set the CR equal to the operation result.

The *OR* instruction is used to OR the *bit* with the CR, and set the CR equal to the operation result.

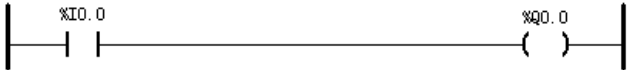
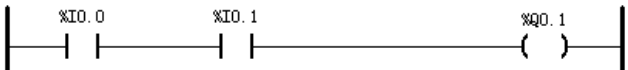

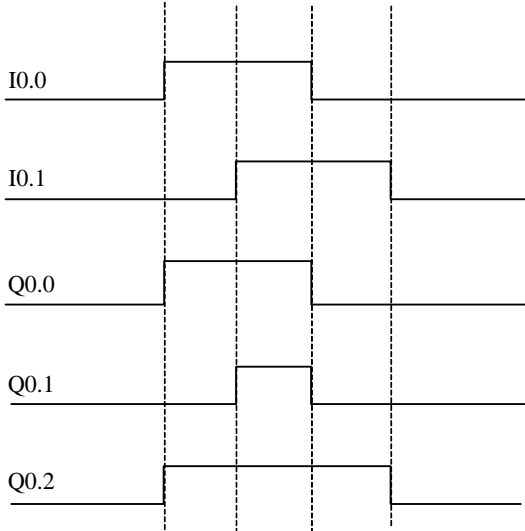
The Normally Closed contacts are represented by the *LDN*, *ANDN*, and *ORN* instructions.

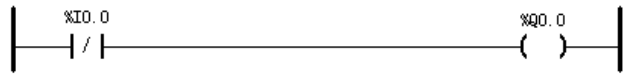

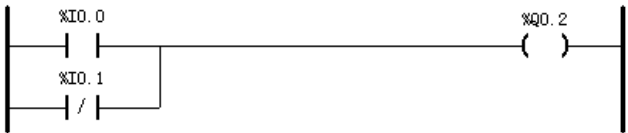
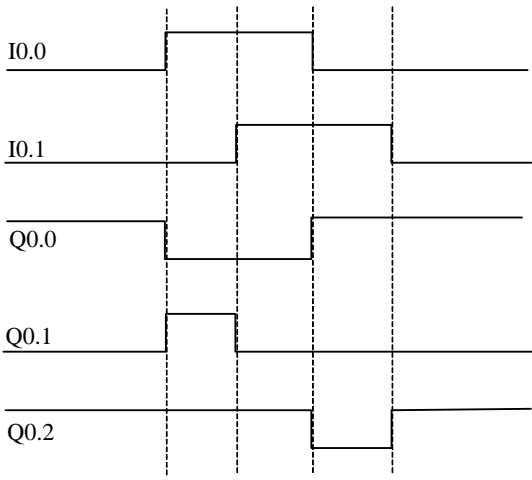
The *LDN* instruction loads the logical NOT of the *bit* value and sets the CR equal to the operation result.

The *ANDN* instruction is used to AND the logical NOT of the *bit* value with the CR, and set the CR equal to the operation result.

The *ORN* instruction is used to OR the logical NOT of the *bit* value with the CR, and set the CR equal to the operation result.

➤ Examples

LD	IL
	LD %IO.0 ST %Q0.0
	LD %IO.0 AND %IO.1 ST %Q0.1
	LD %IO.0 OR %IO.1 ST %Q0.2
Time Sequence Diagram	
	

LD	IL
	LDN %I0.0 ST %Q0.0
	LD %I0.0 ANDN %I0.1 ST %Q0.1
	LD %I0.0 ORN %I0.1 ST %Q0.2
Time Sequence Diagram	
	

6.2.2 Immediate Contact

➤ Description

	Name	Usage	Group	
LD	Normally open immediate contact	<i>bit</i> — I —		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	Normally closed immediate contact	<i>bit</i> — /I —		
IL	LDI	LDI <i>bit</i>	C	
	ANDI	ANDI <i>bit</i>	P	
	ORI	ORI <i>bit</i>		
	LDNI	LDNI <i>bit</i>	C	
	ANDNI	ANDNI <i>bit</i>	P	
	ORNI	ORNI <i>bit</i>		

Operand	Input/Output	Data Type	Acceptable Memory Areas
<i>bit</i>	Input	BOOL	I (CPU body)

When the immediate instruction is executed, it obtains the physical value of the input channel immediately, but the corresponding input image register is not updated.

The immediate instructions can only be used for the DI channels on the CPU body, and are not influenced by the input filter time configured in the [Hardware].

In contrary to a standard contact, an immediate contact does not rely on the scan cycle to update and so it can respond to the input signal more quickly.

➤ LD

When the physical input value (*bit*) is equal to 1, the Normally Open Immediate contact is closed (on) and then

power flow is passed to the next element.

When the physical input value (*bit*) is equal to 0, the Normally Closed Immediate contact is closed (on) and then power flow is passed to the next element.

➤ **IL**

The Normally Open Immediate contacts are represented by the *LDI*, *ANDI*, and *ORI* instructions.

The *LDI* instruction loads the the physical input value (*bit*) and sets the CR equal to the result.

The *ANDI* instruction is used to AND the physical input value (*bit*) with the CR, and set the CR equal to the operation result.

The *ORI* instruction is used to OR the physical input value (*bit*) with the CR, and set the CR equal to the operation result.

The Normally Closed Immediate contacts are represented by the *LDNI*, *ANDNI*, and *ORNI* instructions.

The *LDNI* instruction loads the logical NOT of the physical input value (*bit*) and sets the CR equal to the operation result.

The *ANDNI* instruction is used to AND the logical NOT of the physical input value (*bit*) with the CR, and set the CR equal to the operation result.

The *ORNI* instruction is used to OR the logical NOT of the physical input value (*bit*) with the CR, and set the CR equal to the operation result.

6.2.3 Coil

➤ Description

	Name	Usage	Group	
LD	Set Coil	$\overline{-(\textit{bit})}$		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	Reset Coil	$\overline{-(\textit{bit})}$		
	Null coil	$\overline{-(\textit{NUL})}$		
IL	ST	ST <i>bit</i>	U	
	STN	STN <i>bit</i>		

Operand	Input/Output	Data Type	Acceptable Memory Areas
<i>bit</i>	Output	BOOL	Q, V, M, SM, L

■ LD

The Coil instruction writes the power flow to the output image register for the *bit*.

The Negated Coil instruction writes the inverse of the power flow to the output image register for the *bit*.

The function of the Reset Coil is: if the power flow is 1, the output image register for the *bit* is set equal to 0, otherwise the register remains unchanged.

The function of the Set Coil is: if the power flow is 1, the output image register for the *bit* is set equal to 1, otherwise the register remains unchanged.

The function of the Null Coil is to indicate the end of a network, so this instruction is only to facilitate you in programming, but doesn't execute any particular operation.

■ IL

The coils are represented by the *ST*, *STN*, *R* and *S* instructions.

The *ST* instruction writes the CR to the output image register for the *bit*.

The *STN* instruction writes the inverse of the CR to the output image register for the *bit*.

The function of the *R* instruction is: if the CR is equal to 1, the output image register for the *bit* is set equal to 0, otherwise the register remains unchanged.

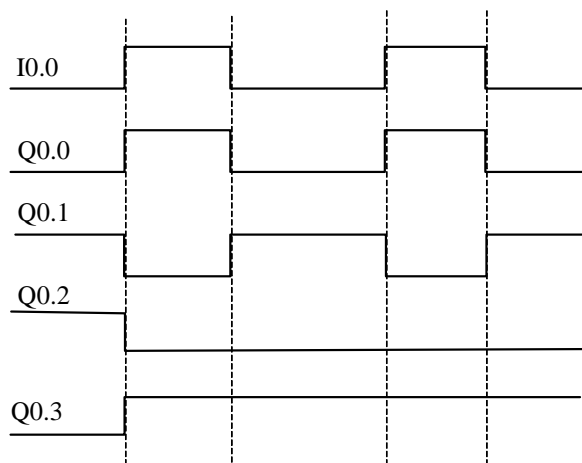
The function of the *S* instruction is: if the CR is equal to 1, the output image register for the *bit* is set equal to 1, otherwise the register remains unchanged.

ST, *STN*, *R* and *S* instructions don't influence the CR.

➤ Examples

LD	IL
	<pre>LD %I0.0 ST %Q0.0 STN %Q0.1 R %Q0.2 S %Q0.3</pre>
	<pre>LD %M0.0 MOVE %VW0, %VW2</pre>
Time Sequence Diagram	

Assume that the values of Q0.1 and Q0.2 are 1 and 0 respectively before these statements are executed.



6.2.4 Immediate Coil

➤ Description

	Name	Usage	Group	
LD	Set Immediate Coil	$\text{---} \begin{matrix} bit \\ \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \\ \text{---} \text{---} \end{matrix} \text{---}$		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	STI	STI <i>bit</i>	U	<input checked="" type="checkbox"/> CPU508

Operand	Input/Output	Data Type	Acceptable Memory Areas
<i>bit</i>	Output	BOOL	Q (CPU body)

These immediate instructions can only be used for the DO channels on the CPU body.

■ **LD**

When the Immediate Coil instruction is executed, it immediately writes the power flow to both the physical output (*bit*) and the corresponding output image register.

When the Reset Immediate Coil instruction is executed, if the power flow is 1, both the physical output (*bit*) and the corresponding output image register are set equal to 0 immediately, otherwise they remain unchanged.

When the Set Immediate Coil instruction is executed, if the power flow is 1, both the physical output (*bit*) and the corresponding output image register are set equal to 1 immediately, otherwise they remain unchanged.

■ **IL**

The immediate coils are represented by the *STI*, *RI* and *SI* instructions.

When the *STI* instruction is executed, it immediately writes the CR to both the physical output (*bit*) and the corresponding output image register.

When the *RI* instruction is executed, if the CR is equal to 1, both the physical output (*bit*) and the corresponding output image register are set equal to 0 immediately, otherwise they remain unchanged.

When the *SI* instruction is executed, if the CR is equal to 1, both the physical output (*bit*) and the corresponding output image register are set equal to 1 immediately, otherwise they remain unchanged.

STI, *RI* and *SI* instructions don't influence the CR.

6.2.5 Set And Reset Coil

➤ Description

	Name	Usage	Group
LD	Reset	$\overline{\text{bit}} \text{ (R)}$	<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
	Set	$\text{bit} \text{ (S)}$	

IL	Reset	R <i>bit</i>	U	☑ CPU508
	Set	S <i>bit</i>		

Operand	Input/Output	Data Type	Acceptable Memory Areas
<i>bit</i>	Output	BOOL	Q (CPU body)

■ **LD**

The function of the Reset Coil is: if the power flow is 1, the output image register for the bit is set equal to 0, otherwise the register remains unchanged.

The function of the Set Coil is: if the power flow is 1, the output image register for the bit is set equal to 1, otherwise the register remains unchanged.

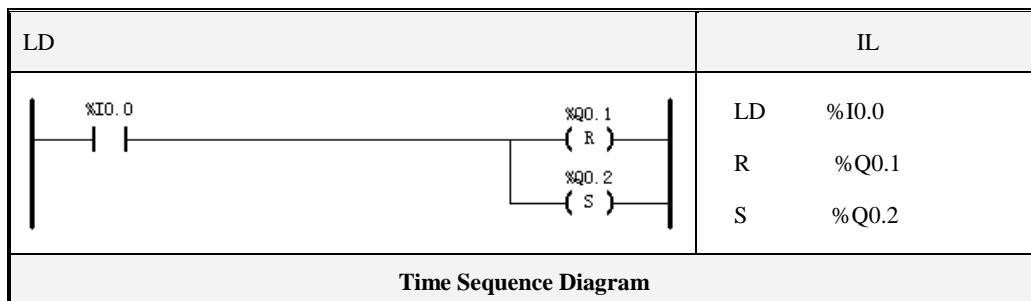
■ **IL**

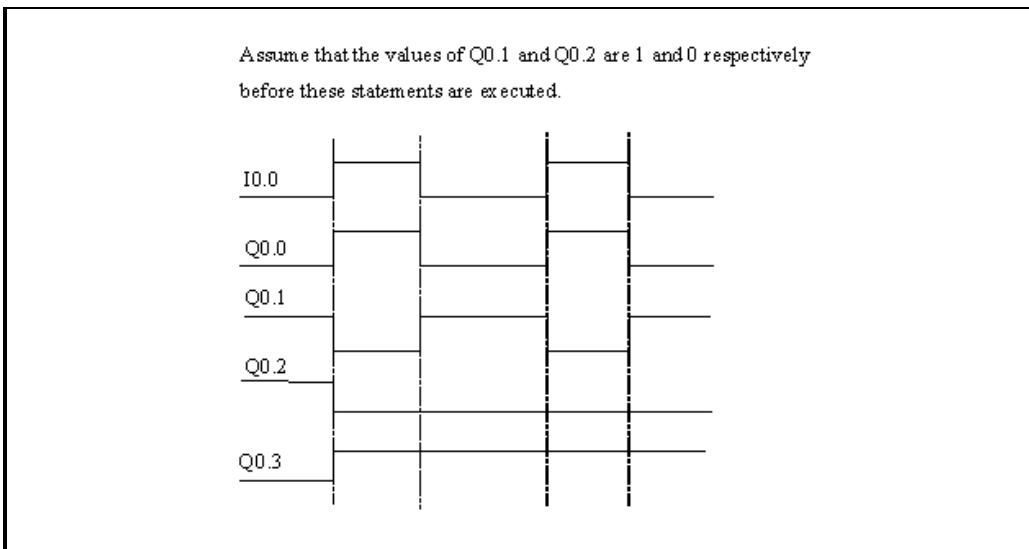
The function of the R instruction is: if the CR is equal to 1, the output image register for the bit is set equal to 0, otherwise the register remains unchanged.

The function of the S instruction is: if the CR is equal to 1, the output image register for the bit is set equal to 1, otherwise the register remains unchanged.

R and S instructions don't influence the CR.

➤ **Examples**





6.2.6 Set and Reset Block Coil

➤ Description

	Name	Usage	Group	
LD	Reset Block Coil			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	Set Block Coil			
IL	R_BLK	R_BLK N,Q	U	
	S_BLK	S_BLK N,Q		

Operand	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	BOOL	Power flow
N	Input	INT	L, M, V, constant
Q	Output	BOOL	Q, V, M, SM, L

■ **LD**

Effect of R_BLK: if the IN value is 1, it will set consecutive bits from address Q to 0, otherwise remain unchanged.

Effect of S_BLK: if the IN value is 1, it will set consecutive bits from address Q to 1, otherwise remain unchanged.

■ **IL**

Effect of R_BLK: if the CR value is 1, it will set consecutive bits from address Q to 0, otherwise remain unchanged.

Effect of S_BLK: if the CR value is 1, it will set consecutive bits from address Q to 1, otherwise remain unchanged.

The execution of R_BLK and S_BLK will not affect the CR value.

The max amount of Parameter N is 1024.

Parameter Q is a start address of memory block with variable length. Please be noted that the whole memory block is forbidden to be input in illegal memory zones, otherwise the consequences may be terrible.

6.2.7 Set And Reset Immediate Coil

➤ Description

	Name	Usage	Group	
LD	Reset Immediate Coil	$\overset{bit}{-(RI)-}$		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	Set Immediate Coil	$\overset{bit}{-(SI)-}$		
IL	RI	RI <i>bit</i>	U	
	SI	SI <i>bit</i>		

Operands	Input/Output	Data Type	Acceptable Memory Areas
bit	Output	BOOL	Q (CPU body)

These immediate instructions can only be used for the DO channels on the CPU body.

■ LD

When the Reset Immediate Coil instruction is executed, if the power flow is 1, both the physical output (bit) and the corresponding output image register are set equal to 0 immediately, otherwise they remain unchanged.

When the Set Immediate Coil instruction is executed, if the power flow is 1, both the physical output (bit) and the corresponding output image register are set equal to 1 immediately, otherwise they remain unchanged.

■ IL

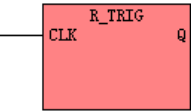
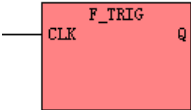
When the RI instruction is executed, if the CR is equal to 1, both the physical output (bit) and the corresponding output image register are set equal to 0 immediately, otherwise they remain unchanged.

When the SI instruction is executed, if the CR is equal to 1, both the physical output (bit) and the corresponding output image register are set equal to 1 immediately, otherwise they remain unchanged.

RI and SI instructions don't influence the CR.

6.2.8 Edge detection

➤ Description

	Name	Usage	Group	
LD	Rising edge detector			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	Falling edge detector			
IL	R_TRIG	R_TRIG	P	
	F_TRIG	F_TRIG		

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>CLK</i> (LD)	Input	BOOL	Power flow
<i>Q</i> (LD)	Output	BOOL	Power flow

■ LD

The function of the *R_TRIG* instruction is to detect the rising edge of the *CLK* input: following a 0-to-1 transition of the *CLK* input, the *Q* output is set to 1 for one scan cycle and then returns to 0.

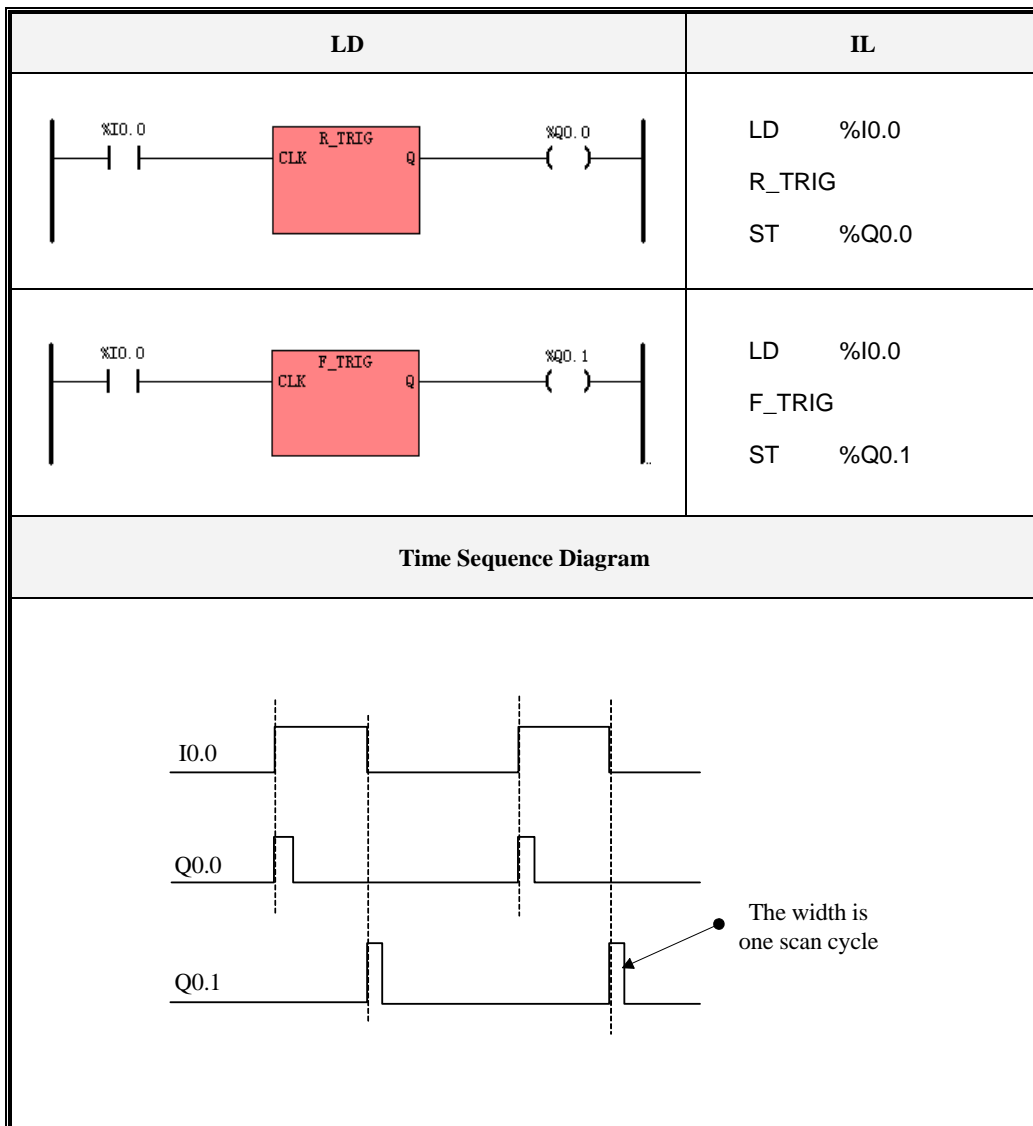
The function of the *F_TRIG* instruction is to detect the falling edge of the *CLK* input: following a 1-to-0 transition of the *CLK* input, the *Q* output is set to 1 for one scan cycle and then returns to 0.

■ IL

The function of the *R_TRIG* instruction is to detect the rising edge of the CR: following a 0-to-1 transition of the CR, the *Q* output is set to 1 for one scan cycle and then returns to 0.

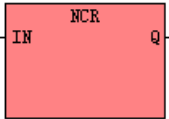
The function of the *F_TRIG* instruction is to detect the falling edge of the CR: following a 1-to-0 transition of the CR, the *Q* output is set to 1 for one scan cycle and then returns to 0.

➤ Examples



6.2.9 NCR (NOT)

➤ Description

	Name	Usage	Group	
LD	NCR			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	NCR	NCR	P	<input checked="" type="checkbox"/> CPU508

Parameter	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BOOL	Power flow
<i>Q</i>	Output	BOOL	Power flow

➤ **LD**

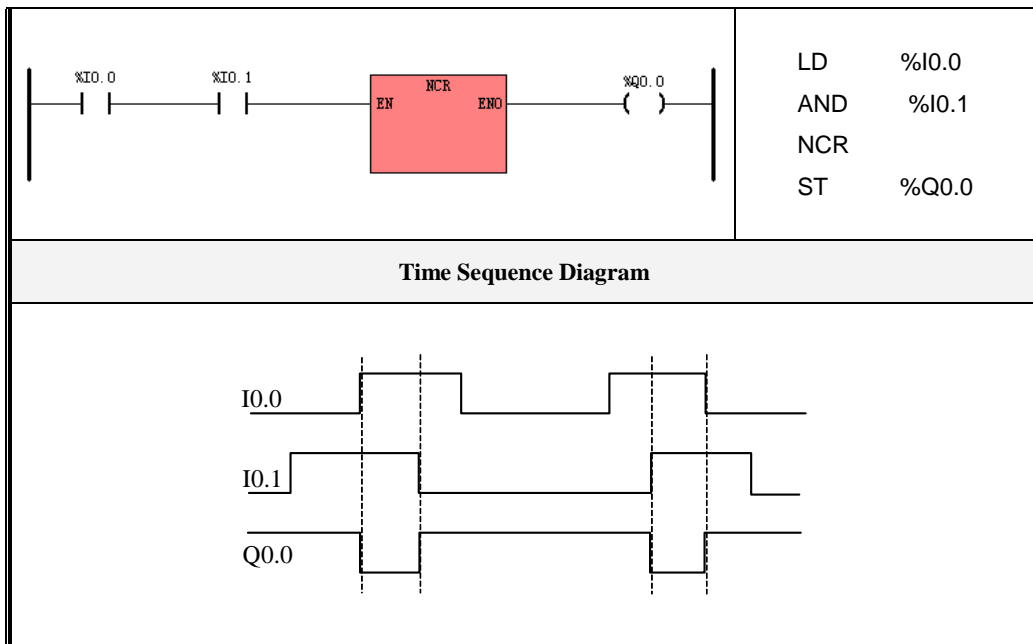
The NCR instruction changes the state of the power flow from 1 to 0 or from 0 to 1.

➤ **IL**

The NCR instruction changes the CR from 1 to 0 or from 0 to 1.

➤ Examples

LD	IL
-----------	-----------



6.2.10 Bistable elements

The Bistable element is one of the function blocks defined in the IEC61131-3 standard, totally in two types, i.e. the Set Dominant Bistable (SR) and the Reset Dominant Bistable (RS).

Please refer to [2.6.4 Function Block and Function Block Instance](#) for more detailed information.

6.2.10.1 SR (Set Dominant Bistable)

➤ Description

	Name	Usage	Group	
LD	SR			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506

IL	SR	LD <i>SI</i> SR <i>SRx, R</i>	P	<input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
----	----	----------------------------------	---	--------------------------------------------------------------------------------------------

Parameter	Input/Output	Data Type	Acceptable Memory Areas
<i>SRx</i>	-	SR instance	SR
<i>SI</i>	Input	BOOL	Power flow
<i>R</i>	Input	BOOL	I, Q, V, M, SM, L, T, C, RS, SR
<i>QI</i>	Output	BOOL	Power flow

The Set Dominant Bistable (*SR*) is a bistable element where the set input dominates. If the set (*SI*) and reset (*R*) inputs are both 1, both the output *QI* and the status value of *SRx* will be 1.

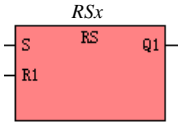
The following is a Truth Table for the *SR* Instruction:

<i>SI</i>	<i>R</i>	<i>QI, SRx</i>
0	0	Previous value
0	1	0
1	0	1
1	1	1

6.2.10.2 RS (Reset Dominant Bistable)

➤ Description

	Name	Usage	Group	<input checked="" type="checkbox"/> CPU504
--	-------------	--------------	--------------	--------------------------------------------

LD	RS		<input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	RS	LD <i>S</i> RS <i>RSx, R1</i>	P

Parameter	Input/Output	Data Type	Acceptable Memory Areas
<i>RSx</i>	-	RS instance	RS
<i>S</i>	Input	BOOL	Power flow
<i>R1</i>	Input	BOOL	I, Q, V, M, SM, L, T, C, RS, SR
<i>Q1</i>	Output	BOOL	Power flow

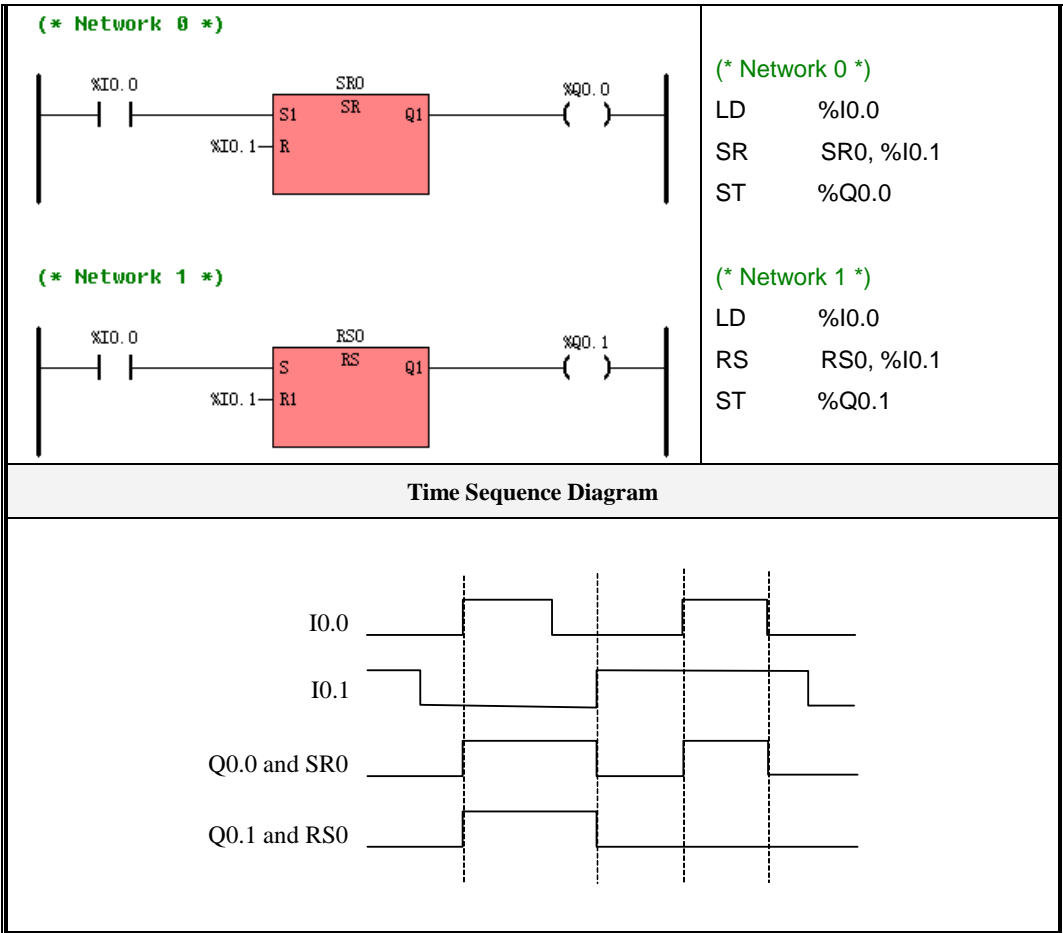
The Reset Dominant Bistable (*RS*) is a bistable element where the reset input dominates. If the set (*S*) and reset (*R1*) inputs are both 1, both the output *Q1* and the status value of *RSx* will be 0.

The following is a Truth Table for the *RS* Instruction:

<i>R1</i>	<i>S</i>	<i>Q1, SRx</i>
0	0	Previous value
0	1	1
1	0	0
1	1	0

6.2.10.3 Examples

LD	IL
-----------	-----------



6.2.11 ALT (Alternate)

➤ Description

	Name	Usage	Group	
LD	ALT			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA

IL	ALT	ALT <i>Q</i>	U	<input checked="" type="checkbox"/> CPU508
----	-----	--------------	---	--------------------------------------------

Parameter	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i> (LD)	Input	BOOL	Power flow
<i>Q</i>	Output	BOOL	Q, V, M, SM, L

➤ **LD**

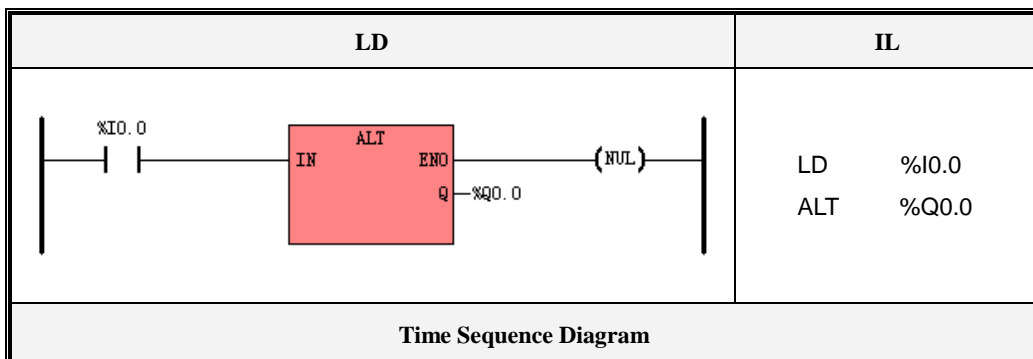
The ALT instruction changes the value of *Q* from 1 to 0 or from 0 to 1 on the rising edge of the *IN* input.

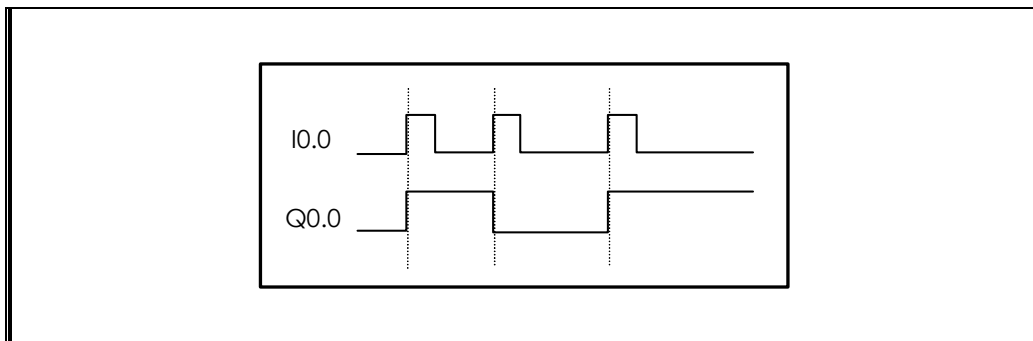
➤ **IL**

The ALT instruction changes the value of *Q* from 1 to 0 or from 0 to 1 on the rising edge of the CR.

This instruction does not influence the CR.

➤ Examples





6.2.12 NOP (No Operation)

➤ Description

	Name	Usage	Group	
LD	Dummy			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	Dummy	NOP <i>N</i>	U	<input checked="" type="checkbox"/> CPU508

Parameter	Input/Output	Data Type	Acceptable Memory Areas
<i>N</i>	Input	INT	Constant (Positive)

The NOP instruction does nothing and has no effect on the user program execution. The program Execution continues with the next instruction.

The NOP instruction is typically used to generate delays in the program execution. The operand *N* is a positive integer constant.

6.2.13 Bracket Modifier

➤ Description

	Name	Usage	Group	
IL	AND(AND(U	<input checked="" type="checkbox"/> CPU504
	OR(OR(<input checked="" type="checkbox"/> CPU504EX
))	P	<input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508

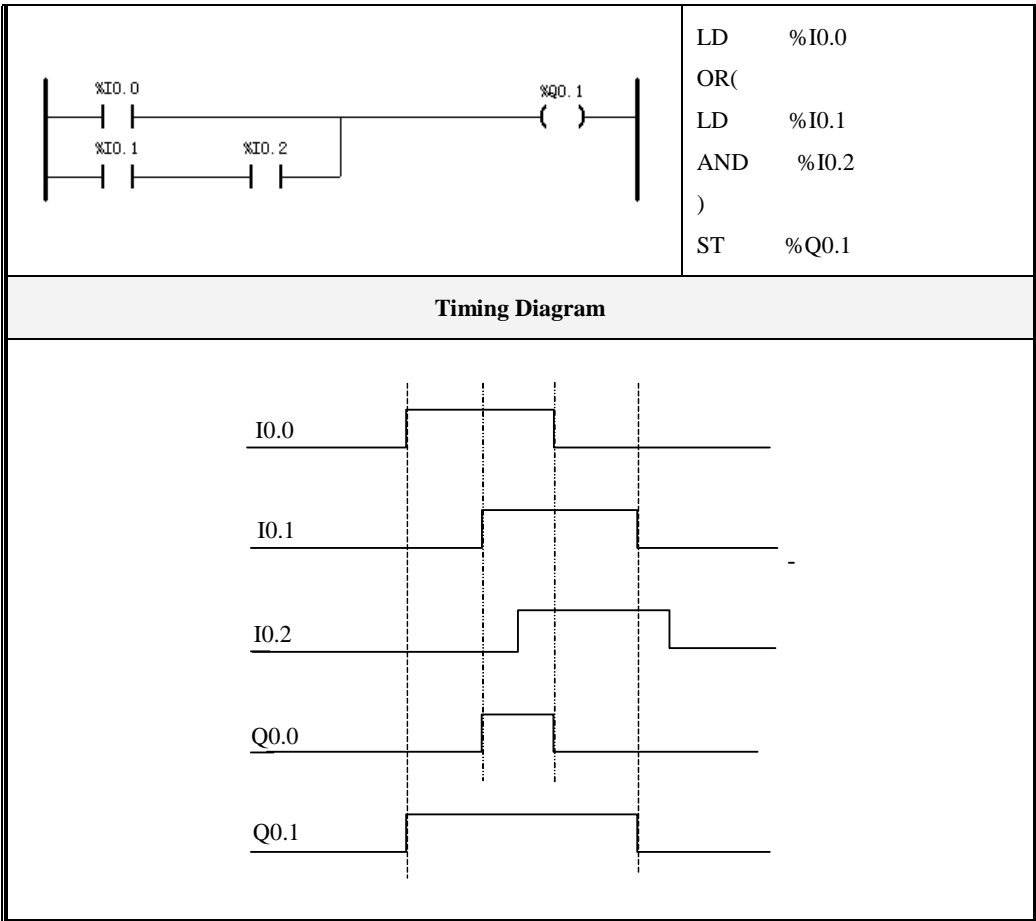
The Bracket modifier is only represented in IL. LD, ST and so on can take complicated expressions as operands, but IL only provides simple expressions. Therefore, the IEC61131-3 standard defines bracket modifier for IL to deal with some complicated expressions. Either “AND(” or “OR(” is paired with “)”.

In an IL program, before executing the statements between “AND(” and “)”, the CR is temporarily stored at first; then the statements in the brackets are executed, and the execution result is ANDed with the temporarily stored CR, and finally the CR is set equal to the operation result.

Similarly, before executing the statements between “OR(” and “)”, the CR is temporarily stored at first; then the statements in the brackets are executed, and the execution result is ORed with the temporarily stored CR, and finally the CR is set equal to the operation result.

➤ Examples

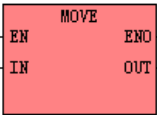
LD	IL
	<pre>LD %I0.0 AND(LD %I0.1 OR %I0.2) ST %Q0.0</pre>



6.3 Move Instructions

6.3.1 MOVE

➤ Description

	Name	Usage	Group	
LD	MOVE			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	MOVE	MOVE IN, OUT	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE, WORD, DWORD, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>OUT</i>	Output	BYTE, WORD, DWORD, INT, DINT, REAL	Q, M, V, L, SM, AQ, pointer

The MOVE instruction moves the value of *IN* to the address *OUT*. This instruction executes an assignment operation, and the *IN* and *OUT* must be of the same data type.

■ **LD**

If *EN* is 1, this instruction is executed..

■ **IL**

If the *CR* is 1, this instruction is executed, and it doesn't influence the *CR*.

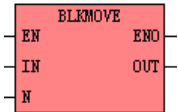
➤ Examples

LD		<p>%SM0.0 is always ON, therefore the MOVE is always executed: B#45 is assigned to %VB0.</p>
		<p>If %IO.0 is 0, the MOVE is not executed. If %IO.0 is 1, the value of %VB10 is assigned to %VB11.</p>
IL	<p>LD %SM0.0 (* The CR is created with %SM0.0 *)</p> <p>MOVE B#45, %VB0 (* B#45 is assigned to %VB0 *)</p>	
	<p>LD %IO.0 (* The CR is created with %IO.0 *)</p> <p>MOVE %VB10, %VB11 (* If the CR is 1, the value of %VB10 is assigned to %VB11. *)</p> <p style="text-align: center;">(*Otherwise, this statement is not executed, %VB11 remains unchanged *)</p>	

6.3.2 BLKMOVE (Block Move)

➤ Description

	Name	Usage	Group	
				<input checked="" type="checkbox"/> CPU504

LD	BLKMOVE			<input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	BLKMOVE	BLKMOVE <i>IN, OUT, N</i>	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE, WORD, DWORD, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC
<i>N</i>	Input	BYTE	I, Q, M, V, L, SM, constant
<i>OUT</i>	Output	BYTE, WORD, DWORD, INT, DINT, REAL	Q, M, V, L, SM, AQ

The *IN* and *OUT* must be of the same data type.

The BLKMOVE instruction moves the *N* number of variables from the successive range that begins with the address *IN* to the successive range that begins with the address *OUT*.

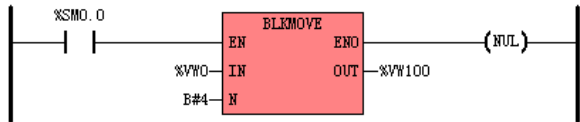
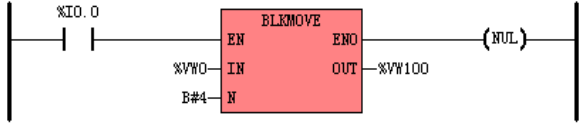
■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

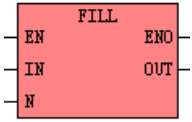
If the *CR* is 1, this instruction is executed, and it does not influence the *CR*.

➤ Examples

LD		<p>%SM0.0 is always ON, therefore the BLKMOVE is always executed: the data in %VW0 - %VW6 are moved into %VW100 - %VW106.</p>																
LD		<p>If %I0.0 is 1, the data in %VW0 - %VW6 are moved into %VW100 - %VW106. Otherwise, the BLKMOVE is not executed.</p>																
IL	<p>LD %SM0.0 (* The CR is created with the% SM0.0 *)</p> <p>BLKMOVE %VW0, %VW100, B#4 (* The data in VW0 - VW6 are moved into %VW100 - %VW106 *)</p>																	
IL	<p>LD %I0.0 (* The CR is created with %I0.0 *)</p> <p>BLKMOVE %VW0, %VW100, B#4 (* If the CR is 0, this statement isn't executed *)</p> <p style="text-align: center;">(* If the CR is 1, the data in %VW0 - %VW6 are moved into %VW100 - %VW106 *)</p>																	
RESULT	<p>The following is an example:</p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse; text-align: center;"> <tr> <td style="padding: 5px;">VW0</td> <td style="padding: 5px;">VW2</td> <td style="padding: 5px;">VW4</td> <td style="padding: 5px;">VW6</td> </tr> <tr> <td style="padding: 5px;">0</td> <td style="padding: 5px;">10</td> <td style="padding: 5px;">20</td> <td style="padding: 5px;">30</td> </tr> <tr> <td style="padding: 5px;">VW100</td> <td style="padding: 5px;">VW102</td> <td style="padding: 5px;">VW104</td> <td style="padding: 5px;">VW106</td> </tr> <tr> <td style="padding: 5px;">0</td> <td style="padding: 5px;">10</td> <td style="padding: 5px;">20</td> <td style="padding: 5px;">30</td> </tr> </table>		VW0	VW2	VW4	VW6	0	10	20	30	VW100	VW102	VW104	VW106	0	10	20	30
VW0	VW2	VW4	VW6															
0	10	20	30															
VW100	VW102	VW104	VW106															
0	10	20	30															

6.3.3 FILL (Memory Fill)

➤ Description

	Name	Usage	Group	
LD	FILL			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	FILL	FILL <i>IN, OUT, N</i>	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE	Constant
<i>N</i>	Input	BYTE	constant
<i>OUT</i>	Output	BYTE	M, V, L

The FILL instruction sets the *N* number of successive variables, beginning with the address *OUT*, to the specified constant *IN*.

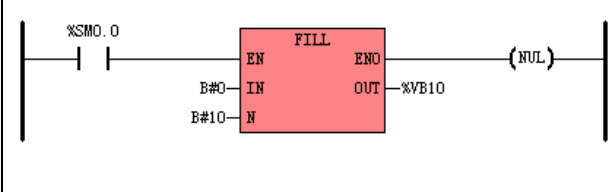
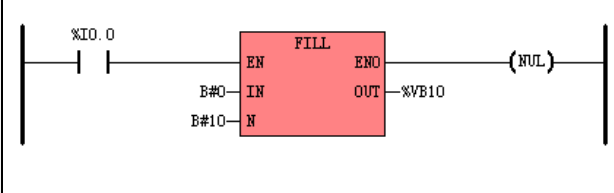
➤ **LD**

If *EN* is 1, this instruction is executed.

➤ **IL**

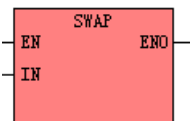
If the *CR* is 1, this instruction is executed, and it does not influence the *CR*.

➤ Examples

LD		<p>%SM0.0 is always ON, therefore the FILL is always executed: 10 variables from %VB10 to %VB19 are all set to B#0.</p>														
		<p>If %I0.0 is 0, the FILL is not executed. If %I0.0 is 1, 10 variables from %VB10 to %VB19 are all set to B#0.</p>														
IL	<p>LD %SM0.0 (* The CR is created with %SM0.0 *)</p> <p>FILL B#0, %VB10, B#10 (* 10 variables from %VB10 to %VB19 are all set to B#0 *)</p>															
	<p>LD %I0.0 (* The CR is created with %I0.0 *)</p> <p>FILL B#0, %VB10, B#10 (* If the CR is 1, 10 variables from %VB10 to %VB19 are all set to B#0 *)</p> <p style="text-align: center;">(* If the CR is 0, this statement is not executed *)</p>															
RESULT	<table border="1" style="margin: auto;"> <tr> <td>VB10</td> <td>VB11</td> <td>VB12</td> <td>VB13</td> <td>...</td> <td>VB18</td> <td>VB19</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>...</td> <td>0</td> <td>0</td> </tr> </table>		VB10	VB11	VB12	VB13	...	VB18	VB19	0	0	0	0	...	0	0
VB10	VB11	VB12	VB13	...	VB18	VB19										
0	0	0	0	...	0	0										

6.3.4 SWAP

➤ Description

	Name	Usage	Group	
LD	SWAP			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	SWAP	SWAP <i>IN</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input/Output	WORD, DWORD	Q, M, V, L, SM

The SWAP instruction exchanges the most significant byte with the least significant byte of the word (*IN*), or exchanges the most significant word with the least significant word of the double word (*IN*).

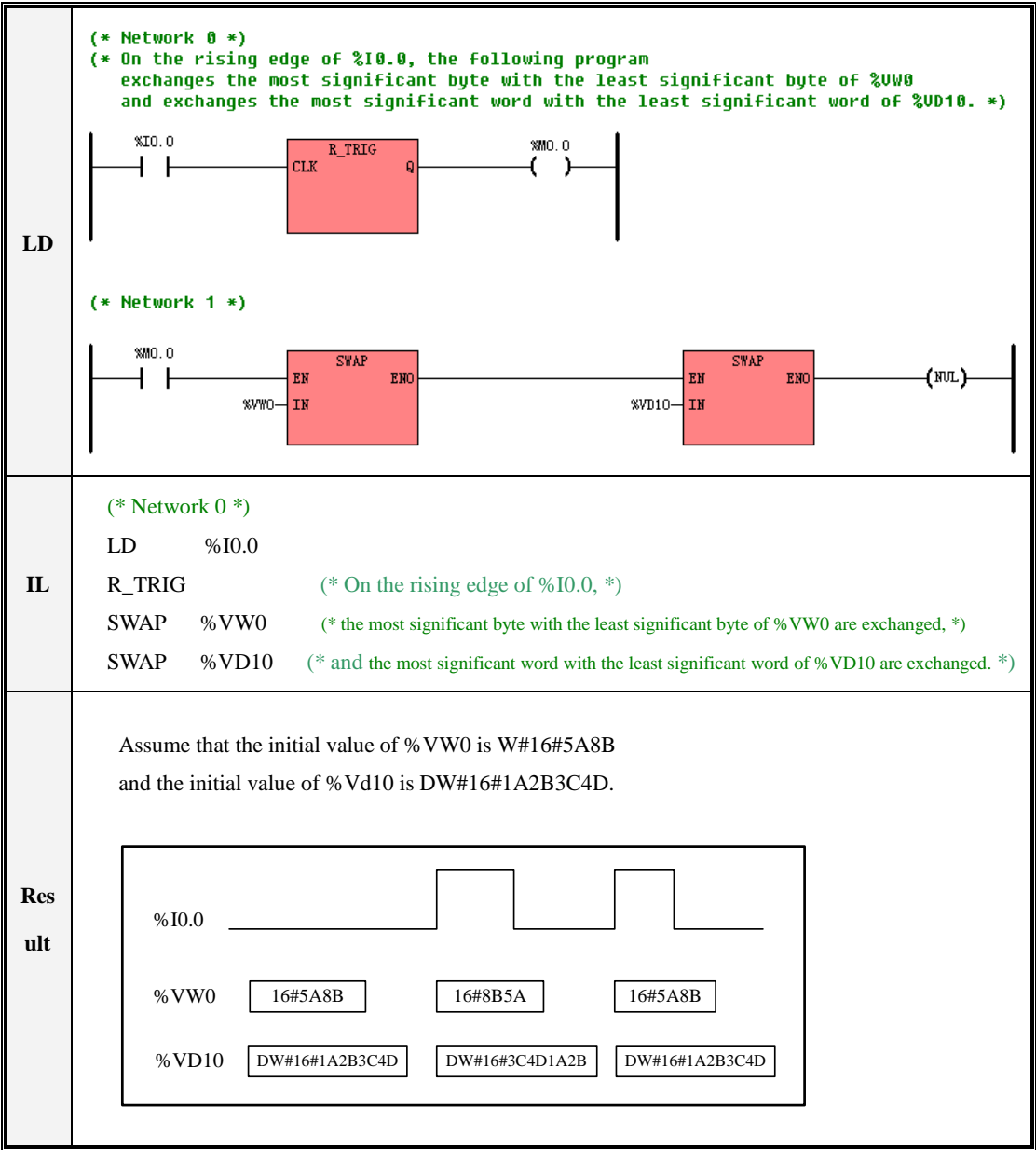
➤ **LD**

If *EN* is 1, this instruction is executed.

➤ **IL**

If the *CR* is 1, this instruction is executed, and it does not influence the *CR*.

➤ Examples

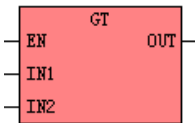


6.4 Compare Instructions

For all the compare instructions, BYTE comparisons are unsigned. INT, DINT and REAL comparisons are signed.

6.4.1 GT (Greater Than)

➤ Description

	Name	Usage	Group	
LD	GT			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	GT	GT <i>IN1, IN2</i>	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>IN2</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>OUT</i> (LD)	Output	BOOL	Power flow

The *IN1* and *IN2* must be of the same data type.

■ **LD**

If *EN* is 1, this instruction compares *IN1* greater than *IN2* and the Boolean result is assigned to *OUT*;

If *EN* is 0, this instruction is not executed, and *OUT* is set equal to 0.

■ **IL**

If CR is 1, this instruction compares *IN1* greater than *IN2* and the Boolean result is assigned to CR;


If CR is 0, this instruction is not executed, and CR remains 0.

➤ Examples

LD		<p>SM0.0 is always ON, therefore <i>GT</i> is always executed: if the value of VB0 is greater than B#200, Q0.0 is set equal to 1, otherwise Q0.0 is set equal to 0.</p>
		<p>If I0.0 is 1: if the value of VW0 is greater than that of VW2, Q0.0 is set to be 1, otherwise Q0.0 is set to be 0. If I0.0 is 0: <i>GT</i> is not executed, and Q0.0 is set to be 0.</p>
IL	LD %SM0.0	(* CR is created with SM0.0 *)
	GT %VB0, B#200	(* If VB0 is greater than B#200, CR is set to be 1, otherwise CR is set to be 0 *)
	ST %Q0.0	(* Q0.0 is set equal to CR *)
	LD %I0.0	(* CR is created with I0.0 *)
	GT %VW0, %VW2	(* If CR is 1: if VW0 is greater than VW2, CR is set to be 1, otherwise CR is set to be 0 *) (* If CR is 0: GT is not executed, CR remains 0 *)
	ST %Q0.0	(* Q0.0 is set equal to CR *)

6.4.2 GE (Greater than or Equal to)

➤ Description

	Name	Usage	Group	
LD	GE			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	GE	GE <i>IN1, IN2</i>	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>IN2</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>OUT</i> (LD)	Output	BOOL	Power flow

The *IN1* and *IN2* must be of the same data type.

■ **LD**

If *EN* is 1, this instruction compares *IN1* greater than or equal to *IN2* and the Boolean result is assigned to *OUT*;

If *EN* is 0, this instruction is not executed, and *OUT* is set equal to 0.

■ **IL**

If *CR* is 1, this instruction compares *IN1* greater than or equal to *IN2* and the Boolean result is assigned to *CR*;

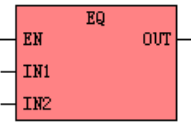
If *CR* is 0, this instruction is not executed, and *CR* remains 0.

➤ Examples

LD		<p>SM0.0 is always ON, therefore <i>GE</i> is always executed: if VB0 is greater than or equal to B#200, Q0.0 is set equal to 1, otherwise Q0.0 is set equal to 0.</p>
		<p>If I0.0 is 1: if VW0 is greater than or equal to VW2, Q0.0 is set to be 1, otherwise Q0.0 is set to be 0. If I0.0 is 0: <i>GE</i> is not executed, and Q0.0 is set to be 0.</p>
IL	<p>LD %SM0.0 (* CR is created with SM0.0 *)</p>	
	<p>GE %VB0, B#200 (* If VB0 is greater than or equal to B#200, CR is set to be 1, otherwise CR is set to be 0 *)</p>	
	<p>ST %Q0.0 (* Q0.0 is set equal to CR *)</p>	
	<p>LD %I0.0 (* CR is created with I0.0 *)</p>	
	<p>GE %VW0, %VW2 (* If CR is 1: if VW0 is greater than or equal to VW2, CR is set to be 1, *)</p> <p style="padding-left: 20px;">(* otherwise CR is set to be 0; If CR is 0: GE is not executed, CR remains 0 *)</p>	
	<p>ST %Q0.0 (* Q0.0 is set equal to CR *)</p>	

6.4.3 EQ (Equal to)

➤ Description

	Name	Usage	Group	
LD	EQ			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	EQ	EQ IN1, IN2	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>IN2</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>OUT (LD)</i>	Output	BOOL	Power flow

The *IN1* and *IN2* must be of the same data type.

■ **LD**

If *EN* is 1, this instruction compares *IN1* equal to *IN2* and the Boolean result is assigned to *OUT*;

If *EN* is 0, this instruction is not executed, and *OUT* is set equal to 0.

■ **IL**

If *CR* is 1, this instruction compares *IN1* equal to *IN2* and the Boolean result is assigned to *CR*;

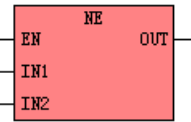
If *CR* is 0, this instruction is not executed, and *CR* remains 0.

➤ Examples

LD		<p>SM0.0 is always ON, therefore <i>EQ</i> is always executed: if the value of VB0 is equal to B#200, Q0.0 is set equal to 1, otherwise Q0.0 is set equal to 0.</p>
		<p>If I0.0 is 1: if the value of VW0 is equal to that of VW2, Q0.0 is set to be 1, otherwise Q0.0 is set to be 0. If I0.0 is 0: <i>EQ</i> is not executed, and Q0.0 is set to be 0.</p>
IL	<p>LD %SM0.0 (* CR is created with SM0.0 *)</p> <p>EQ %VB0, B#200 (* If VB0 is equal to B#200, CR is set to be 1, otherwise CR is set to be 0 *)</p> <p>ST %Q0.0 (* Q0.0 is set equal to CR *)</p>	
	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>EQ %VW0, %VW2 (* If CR is 1: if VW0 is equal to VW2, CR is set to be 1, otherwise CR is set to be 0 *)</p> <p>(* If CR is 0: EQ is not executed, CR remains 0 *)</p> <p>ST %Q0.0 (* Q0.0 is set equal to CR *)</p>	

6.4.4 NE (Not Equal to)

➤ Description

	Name	Usage	Group	
LD	NE			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	NE	NE IN1, IN2	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>IN2</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>OUT</i> (LD)	Output	BOOL	Power flow

The *IN1* and *IN2* must be of the same data type.

■ **LD**

If *EN* is 1, this instruction compares *IN1* not equal to *IN2* and the Boolean result is assigned to *OUT*;

If *EN* is 0, this instruction is not executed, and *OUT* is set equal to 0.

■ **IL**

If *CR* is 1, this instruction compares *IN1* not equal to *IN2* and the Boolean result is assigned to *CR*;

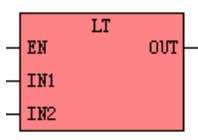
If *CR* is 0, this instruction is not executed, and *CR* remains 0.

➤ Examples

LD		<p>SM0.0 is always ON, therefore <i>NE</i> is always executed: if the value of VB0 is not equal to B#200, Q0.0 is set equal to 1, otherwise Q0.0 is set equal to 0.</p>
		<p>If I0.0 is 1: if the value of VW0 is not equal to that of VW2, Q0.0 is set to be 1, otherwise Q0.0 is set to be 0. If I0.0 is 0: <i>NE</i> is not executed, and Q0.0 is set to be 0.</p>
IL	LD %SM0.0	(* CR is created with SM0.0 *)
	NE %VB0, B#200	(* If VB0 is not equal to B#200, CR is set to be 1, otherwise CR is set to be 0 *)
	ST %Q0.0	(* Q0.0 is set equal to CR *)
	LD %I0.0	(* CR is created with I0.0 *)
	NE %VW0, %VW2	(* If CR is 1: if VW0 is not equal to VW2, CR is set to be 1, otherwise CR is set to be 0 *) (* If CR is 0: NE is not executed, CR remains 0 *)
	ST %Q0.0	(* Q0.0 is set equal to CR *)

6.4.5 LT (Less than)

➤ Description

	Name	Usage	Group	
LD	LT			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	LT	LT IN1, IN2	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>IN2</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>OUT</i> (LD)	Output	BOOL	Power flow

The *IN1* and *IN2* must be of the same data type.

■ **LD**

If *EN* is 1, this instruction compares *IN1* less than *IN2* and the Boolean result is assigned to *OUT*;

If *EN* is 0, this instruction is not executed, and *OUT* is set equal to 0.

■ **IL**

If *CR* is 1, this instruction compares *IN1* less than *IN2* and the Boolean result is assigned to *CR*;

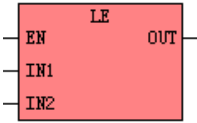
If *CR* is 0, this instruction is not executed, and *CR* remains 0.

➤ Examples

LD		<p>SM0.0 is always ON, therefore <i>LT</i> is always executed: if the value of VB0 is less than B#200, Q0.0 is set equal to 1, otherwise Q0.0 is set equal to 0.</p>
		<p>If I0.0 is 1: if the value of VW0 is less than that of VW2, Q0.0 is set to be 1, otherwise Q0.0 is set to be 0. If I0.0 is 0: <i>LT</i> is not executed, and Q0.0 is set to be 0.</p>
IL	<p>LD %SM0.0 (* CR is created with SM0.0 *)</p>	
	<p>LT %VB0, B#200 (* If VB0 is less than B#200, CR is set to be 1, otherwise CR is set to be 0 *)</p>	
	<p>ST %Q0.0 (* Q0.0 is set equal to CR *)</p>	
	<p>LD %I0.0 (* CR is created with I0.0 *)</p>	
	<p>LT %VW0, %VW2 (* If CR is 1: if VW0 is less than VW2, CR is set to be 1, otherwise CR is set to be 0 *)</p> <p>(* If CR is 0: LT is not executed, CR remains 0 *)</p>	
	<p>ST %Q0.0 (* Q0.0 is set equal to CR *)</p>	

6.4.6 LE (Less than or Equal to)

➤ Description

	Name	Usage	Group	
LD	LE			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	LE	LE IN1, IN2	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>IN2</i>	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
<i>OUT</i> (LD)	Output	BOOL	Power flow

The *IN1* and *IN2* must be of the same data type.

■ **LD**

If *EN* is 1, this instruction compares *IN1* less than or equal to *IN2* and the Boolean result is assigned to *OUT*;

If *EN* is 0, this instruction is not executed, and *OUT* is set equal to 0.

■ **IL**

If *CR* is 1, this instruction compares *IN1* less than or equal to *IN2* and the Boolean result is assigned to *CR*; If

CR is 0, this instruction is not executed, and *CR* remains 0.

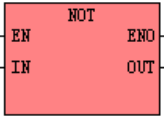
➤ Examples

LD		<p>SM0.0 is always ON, therefore <i>LE</i> is always executed: if VB0 is less than or equal to B#200, Q0.0 is set equal to 1, otherwise Q0.0 is set equal to 0.</p>
		<p>If IO.0 is 1: if VW0 is less than or equal to VW2, Q0.0 is set to be 1, otherwise Q0.0 is set to be 0.</p> <p>If IO.0 is 0: <i>LE</i> is not executed, and Q0.0 is set to be 0.</p>
IL	LD %SM0.0 (* CR is created with SM0.0 *)	
	LE %VB0, B#200 (* If VB0 is less than or equal to B#200, CR is set to be 1, otherwise CR is set to be 0 *)	
	ST %Q0.0 (* Q0.0 is set equal to CR *)	
	LD %IO.0 (* CR is created with IO.0 *)	
	LE %VW0, %VW2 (* If CR is 1: if VW0 is less than or equal to VW2, CR is set to be 1, *)	
	(* otherwise CR is set to be 0; If CR is 0: LE is not executed, CR remains 0 *)	
ST %Q0.0 (* Q0.0 is set equal to CR *)		

6.5 Logical Operations

6.5.1 NOT

➤ Description

	Name	Usage	Group	
LD	NOT			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	NOT	NOT <i>OUT</i>	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>OUT</i>	Output	BYTE, WORD, DWORD	Q, M, V, L, SM, pointer

■ **LD**

The *IN* and *OUT* must be of the same data type.

If *EN* is 1, this instruction inverts each bit of *IN* and assigns the result to *OUT*.

If *EN* is 0, this instruction is not executed.

■ **IL**

If *CR* is 1, this instruction inverts each bit of *OUT* and still stores the result in *OUT*. It does not influence *CR*;

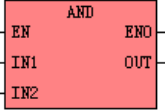
If *CR* is 0, this instruction is not executed.

➤ Examples

LD		<p>If I0.0 is 0: <i>NOT</i> is not executed.</p> <p>If I0.0 is 1: <i>NOT</i> inverts each bit of VW2 and assigns the result to VW2.</p>
IL	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>NOT %VW2 (* If CR is 1: <i>NOT</i> inverts each bit of VW2 and still stores the result in VW2 *)</p> <p style="padding-left: 100px;">(* If CR is 0: <i>NOT</i> instruction is not executed *)</p>	
Result	<p>For the LD example, if <i>NOT</i> instruction is executed, the result will be as the following:</p> <p>Address VW2</p> <p>Value W#16#5555</p> <p>Address VW20</p> <p>Value W#16#AAAA</p>	

6.5.2 AND

➤ Description

	Name	Usage	Group	
LD	AND			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	AND	AND IN1, OUT	U	<input checked="" type="checkbox"/> CPU508

Parameter	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>IN2</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>OUT</i>	Output	BYTE, WORD, DWORD	Q, M, V, L, SM, pointer

■ LD

The *IN1*, *IN2* and *OUT* must be of the same data type.

If *EN* is 1, this instruction ANDs the corresponding bits of *IN1* and *IN2* and assigns the result to *OUT*.

If *EN* is 0, this instruction is not executed.

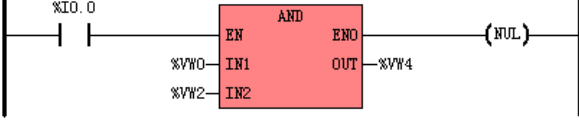
■ IL

The *IN* and *OUT* must be of the same data type.

If *CR* is 1, this instruction ANDs the corresponding bits of *IN* and *OUT* and assigns the result to *OUT*, and it does not influence *CR*.

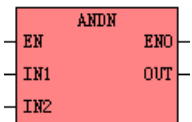
If *CR* is 0, this instruction is not executed.

➤ Examples

LD		<p>If I0.0 is 0: <i>AND</i> is not executed.</p> <p>If I0.0 is 1: The <i>AND</i> instruction ANDs the corresponding bits of VW0 and VW2, and assigns the result to VW4.</p>										
IL	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>AND %VW0, %VW2 (* If CR is 1: The <i>AND</i> instruction ANDs the corresponding bits of VW0 and VW2. *)</p> <p style="padding-left: 100px;">(* and still stores the result in VW2 *)</p> <p style="padding-left: 100px;">(* If CR is 0: The <i>AND</i> instruction is not executed *)</p>											
Result	<p>For the LD example, if <i>AND</i> instruction is executed, the result will be as the following:</p> <table style="margin-left: 40px; border: none;"> <tr> <td style="padding-right: 20px;">Address</td> <td style="padding-right: 20px;">VW0</td> <td style="padding-right: 40px;">VW2</td> </tr> <tr> <td>Value</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#129B</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#960F</td> </tr> </table> <table style="margin-left: 40px; border: none;"> <tr> <td style="padding-right: 20px;">Address</td> <td style="padding-right: 20px;">VW4</td> </tr> <tr> <td>Value</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#120B</td> </tr> </table>		Address	VW0	VW2	Value	W#16#129B	W#16#960F	Address	VW4	Value	W#16#120B
Address	VW0	VW2										
Value	W#16#129B	W#16#960F										
Address	VW4											
Value	W#16#120B											

6.5.3 ANDN

➤ Description

	Name	Usage	Group	
LD	ANDN			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	ANDN	ANDN <i>IN1, OUT</i>	U	<input checked="" type="checkbox"/> CPU508

Parameter	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>IN2</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>OUT</i>	Output	BYTE, WORD, DWORD	Q, M, V, L, SM, pointer

■ LD

The *IN1*, *IN2* and *OUT* must be of the same data type.

If *EN* is 1, this instruction ANDs the corresponding bits of *IN1* and *IN2*, then inverts each bit of the result, and assigns the final result to *OUT*. If *EN* is 0, this instruction is not executed.

■ IL

The *IN* and *OUT* must be of the same data type.

If *CR* is 1, this instruction ANDs the corresponding bits of *IN* and *OUT*, then inverts each bit of the result, and assigns the final result to *OUT*. It does not influence *CR*.

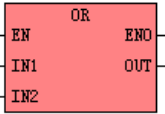
If *CR* is 0, this instruction is not executed.

➤ Examples

LD		<p>If I0.0 is 0: <i>ANDN</i> is not executed.</p> <p>If I0.0 is 1: The <i>ANDN</i> instruction ANDs the corresponding bits of VW0 and VW2, then inverts each bit of the result, and assigns the final result to VW4.</p>										
IL	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>ANDN %VW0, %VW2 (* If CR is 1: The <i>ANDN</i> instruction ANDs the corresponding bits of VW0 and VW2, *)</p> <p style="padding-left: 150px;">(* then inverts each bit of the result, and still stores the final result in VW2 *)</p> <p style="padding-left: 150px;">(* If CR is 0: The <i>ANDN</i> instruction is not executed *)</p>											
Result	<p>For the LD example, if <i>ANDN</i> instruction is executed, the result will be as the following:</p> <table style="margin-left: auto; margin-right: auto; border: none;"> <tr> <td style="padding-right: 20px;">Address</td> <td style="padding-right: 20px;">VW0</td> <td style="padding-right: 20px;">VW2</td> </tr> <tr> <td>Value</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#129B</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#960F</td> </tr> </table> <table style="margin-left: auto; margin-right: auto; border: none;"> <tr> <td style="padding-right: 20px;">Address</td> <td style="padding-right: 20px;">VW4</td> </tr> <tr> <td>Value</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#EDF4</td> </tr> </table>		Address	VW0	VW2	Value	W#16#129B	W#16#960F	Address	VW4	Value	W#16#EDF4
Address	VW0	VW2										
Value	W#16#129B	W#16#960F										
Address	VW4											
Value	W#16#EDF4											

6.5.4 OR

➤ Description

	Name	Usage	Group	
LD	OR			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	OR	OR IN1, OUT	U	<input checked="" type="checkbox"/> CPU508

Parameter	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>IN2</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>OUT</i>	Output	BYTE, WORD, DWORD	Q, M, V, L, SM, pointer

■ LD

The *IN1*, *IN2* and *OUT* must be of the same data type.

If *EN* is 1, this instruction ORs the corresponding bits of *IN1* and *IN2* and assigns the result to *OUT*.

If *EN* is 0, this instruction is not executed.

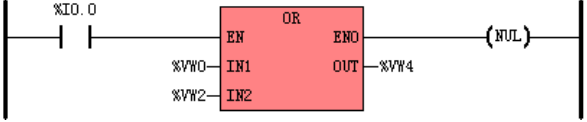
■ IL

The *IN* and *OUT* must be of the same data type.

If *CR* is 1, this instruction ORs the corresponding bits of *IN* and *OUT* and assigns the result to *OUT*, and it does not influence *CR*.

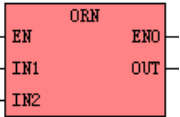
If *CR* is 0, this instruction is not executed.

➤ Examples

LD		<p>If I0.0 is 0: <i>OR</i> is not executed.</p> <p>If I0.0 is 1: The <i>OR</i> instruction ORs the corresponding bits of VW0 and VW2, and assigns the result to VW4.</p>										
IL	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>OR %VW0, %VW2 (* If CR is 1: The <i>OR</i> instruction ORs the corresponding bits of VW0 and VW2, *)</p> <p style="padding-left: 100px;">(* and still stores the result in VW2 *)</p> <p style="padding-left: 100px;">(* If CR is 0: The <i>OR</i> instruction is not executed *)</p>											
Result	<p>For the LD example, if <i>OR</i> instruction is executed, the result will be as the following:</p> <table style="margin-left: 40px; border: none;"> <tr> <td style="padding-right: 20px;">Address</td> <td style="padding-right: 20px;">VW0</td> <td style="padding-right: 40px;">VW2</td> </tr> <tr> <td>Value</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#5555</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#AAAA</td> </tr> </table> <table style="margin-left: 40px; border: none;"> <tr> <td style="padding-right: 20px;">Address</td> <td style="padding-right: 20px;">VW4</td> </tr> <tr> <td>Value</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#FFFF</td> </tr> </table>		Address	VW0	VW2	Value	W#16#5555	W#16#AAAA	Address	VW4	Value	W#16#FFFF
Address	VW0	VW2										
Value	W#16#5555	W#16#AAAA										
Address	VW4											
Value	W#16#FFFF											

6.5.5 ORN

➤ Description

	Name	Usage	Group	
LD	ORN			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	ORN	ORN <i>IN1, OUT</i>	U	<input checked="" type="checkbox"/> CPU508

Parameter	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>IN2</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>OUT</i>	Output	BYTE, WORD, DWORD	Q, M, V, L, SM, pointer

■ LD

The *IN1*, *IN2* and *OUT* must be of the same data type.

If *EN* is 1, this instruction ORs the corresponding bits of *IN1* and *IN2*, then inverts each bit of the result, and assigns the final result to *OUT*. If *EN* is 0, this instruction is not executed.

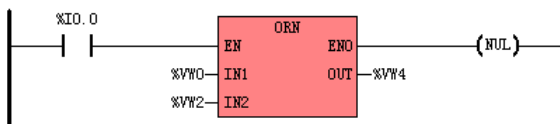
■ IL

The *IN* and *OUT* must be of the same data type.

If *CR* is 1, this instruction ORs the corresponding bits of *IN* and *OUT*, then inverts each bit of the result, and assigns the final result to *OUT*. It does not influence *CR*.

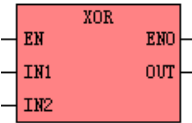
If *CR* is 0, this instruction is not executed.

➤ Examples

LD		<p>If I0.0 is 0: <i>ORN</i> is not executed.</p> <p>If I0.0 is 1: The <i>ORN</i> instruction ORs the corresponding bits of VW0 and VW2, then inverts each bit of the result, and assigns the final result to VW4.</p>										
IL	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>ORN %VW0, %VW2 (* If CR is 1: The <i>ORN</i> instruction ORs the corresponding bits of VW0 and VW2, *)</p> <p style="padding-left: 100px;">(* then inverts each bit of the result, and still stores the final result in VW2 *)</p> <p style="padding-left: 100px;">(* If CR is 0: The <i>ORN</i> instruction is not executed *)</p>											
Result	<p>For the LD example, if <i>ORN</i> instruction is executed, the result will be as the following:</p> <table style="margin-left: 40px; border-collapse: collapse;"> <tr> <td style="padding-right: 20px;">Address</td> <td style="padding-right: 20px;">VW0</td> <td style="padding-right: 40px;">VW2</td> </tr> <tr> <td>Value</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#129B</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#960F</td> </tr> </table> <table style="margin-left: 40px; border-collapse: collapse;"> <tr> <td style="padding-right: 20px;">Address</td> <td style="padding-right: 20px;">VW4</td> </tr> <tr> <td>Value</td> <td style="border: 1px solid black; padding: 2px 10px;">W#16#6960</td> </tr> </table>		Address	VW0	VW2	Value	W#16#129B	W#16#960F	Address	VW4	Value	W#16#6960
Address	VW0	VW2										
Value	W#16#129B	W#16#960F										
Address	VW4											
Value	W#16#6960											

6.5.6 XOR (Exclusive OR)

➤ Description

	Name	Usage	Group	
LD	XOR			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	XOR	XOR IN1, OUT	U	

Parameter	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>IN2</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>OUT</i>	Output	BYTE, WORD, DWORD	Q, M, V, L, SM, pointer

■ **LD**

The *IN1*, *IN2* and *OUT* must be of the same data type.

If *EN* is 1, this instruction XORs the corresponding bits of *IN1* and *IN2* and assigns the result to *OUT*.

If *EN* is 0, this instruction is not executed.

■ **IL**

The *IN* and *OUT* must be of the same data type.

If *CR* is 1, this instruction XORs the corresponding bits of *IN* and *OUT* and assigns the result to *OUT*, and it does not influence *CR*.

If *CR* is 0, this instruction is not executed.

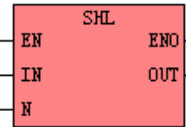
➤ Examples

<p>LD</p>		<p>If I0.0 is 0: <i>XOR</i> is not executed.</p> <p>If I0.0 is 1: The <i>XOR</i> instruction XORs the corresponding bits of VW0 and VW2, and assigns the result to VW4.</p>
<p>IL</p>	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>XOR %VW0, %VW2 (* If CR is 1: The <i>XOR</i> instruction XORs the corresponding bits of VW0 and VW2, *)</p> <p> (* and still stores the result in VW2 *)</p> <p> (* If CR is 0: The <i>XOR</i> instruction is not executed *)</p>	
<p>Result</p>	<p>For the LD example, if <i>XOR</i> instruction is executed, the result will be as the following:</p> <p>Address VW0 VW2</p> <p>Value <input type="text" value="W#16#9514"/> <input type="text" value="W#16#B9A1"/></p> <p>Address VW4</p> <p>Value <input type="text" value="W#16#2CB5"/></p>	

6.6 Shift/Rotate Instructions

6.6.1 SHL (Shift left)

➤ Description

	Name	Usage	Group	
LD	SHL			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	SHL	SHL <i>OUT, N</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>N</i>	Input	BYTE	I, Q, M, V, L, SM, constant, pointer
<i>OUT</i>	Output	BYTE, WORD, DWORD	Q, M, V, L, SM, pointer

■ LD

The *IN* and *OUT* must be of the same data type.

If *EN* is 1, this instruction shifts the value of *IN* to the left by *N* bits, and each bit is filled with a zero while it is shifted left. The result is assigned to *OUT*. If *EN* is 0, this instruction is not executed.

■ IL

If *CR* is 1, this instruction shifts the value of *OUT* to the left by *N* bits, and each bit is filled with a zero while it is shifted left. The result is still stored in *OUT*. It does not influence *CR*.

If *CR* is 0, this instruction is not executed.

➤ Examples

LD		<p>If M0.0 is 0: <i>SHL</i> isn't executed.</p> <p>If M0.0 is 1: <i>SHL</i> shifts QB0 to the left by 1 bit, and the result is still assigned to QB0.</p>
IL	<p>LD %M0.0 (* CR is created with M0.0 *)</p> <p>SHL %QB0, B#1 (* If CR is 1: <i>SHL</i> shifts QB0 to the left by 1 bit, and the result is still stored in QB0 *)</p> <p style="padding-left: 100px;">(* If CR is 0: <i>SHL</i> instruction is not executed *)</p>	
Result	<p>For the LD example, if <i>SHL</i> instruction is executed, the result will be as the following:</p> <p style="margin-left: 40px;">QB0 B#2#10000001</p> <p style="margin-left: 80px;">After 1st shift After 2nd shift After 3rd shift After 4th shift</p> <p style="margin-left: 40px;">QB0 B#2#00000010 B#2#00000100 B#2#00001000 B#2#00010000</p>	

6.6.2 ROL (Rotate left)

➤ Description

	Name	Usage	Group	
LD	ROL			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	ROL	ROL <i>OUT, N</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>N</i>	Input	BYTE	I, Q, M, V, L, SM, constant, pointer
<i>OUT</i>	Output	BYTE, WORD, DWORD	Q, M, V, L, SM, pointer

■ LD

The *IN* and *OUT* must be of the same data type.

If *EN* is 1, this instruction rotates the value of *IN* to the left by *N* bits, and the MSB is rotated to the LSB. The result is assigned to *OUT*. If *EN* is 0, this instruction is not executed.

■ IL

If *CR* is 1, this instruction rotates the value of *OUT* to the left by *N* bits, and the MSB is rotated to the LSB. The result is still stored in *OUT*. It does not influence *CR*.

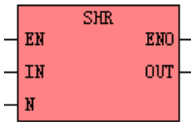
If CR is 0, this instruction is not executed.

➤ Examples

LD		<p>If M0.0 is 0: <i>ROL</i> isn't executed. If M0.0 is 1: <i>ROL</i> rotates QB0 to the left by 1 bit, and the MSB is rotated to the LSB. The result is still assigned to QB0.</p>
IL	<p>LD %M0.0 (* CR is created with M0.0 *)</p> <p>ROL %QB0, B#1 (* If CR is 1: <i>ROL</i> rotates QB0 to the left by 1 bit, and the result is still stored in QB0 *)</p> <p style="padding-left: 100px;">(* If CR is 0: <i>ROL</i> instruction is not executed *)</p>	
Result	<p>For the LD example, if <i>ROL</i> instruction is executed, the result will be as the following:</p> <p style="margin-left: 40px;">QB0 B#2#10100001</p> <p style="margin-left: 80px;">After 1st shift After 2nd shift After 3rd shift After 4th shift</p> <p style="margin-left: 40px;">QB0 B#2#01000011 B#2#10000110 B#2#00001101 B#2#00011010</p>	

6.6.3 SHR (Shift right)

➤ Description

	Name	Usage	Group	
LD	SHR			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	SHR	SHR <i>OUT, N</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>N</i>	Input	BYTE	I, Q, M, V, L, SM, constant, pointer
<i>OUT</i>	Output	BYTE, WORD, DWORD	Q, M, V, L, SM, pointer

■ **LD**

The *IN* and *OUT* must be of the same data type.

If *EN* is 1, this instruction shifts the value of *IN* to the right by *N* bits, and each bit is filled with a zero while it is shifted right. The result is assigned to *OUT*. If *EN* is 0, this instruction is not executed.

■ **IL**

If *CR* is 1, this instruction shifts the value of *OUT* to the right by *N* bits, and each bit is filled with a zero while it is shifted right. The result is still stored in *OUT*. It does not influence *CR*.

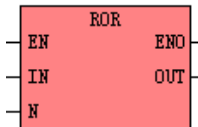
If *CR* is 0, this instruction is not executed.

➤ Examples

LD		<p>If M0.0 is 0: <i>SHR</i> isn't executed.</p> <p>If M0.0 is 1: <i>SHR</i> shifts QB0 to the right by 1 bit, and the result is still assigned to QB0.</p>
IL	<p>LD %M0.0 (* CR is created with M0.0 *)</p> <p>SHR %QB0, B#1 (* If CR is 1: <i>SHR</i> shifts QB0 to the right by 1 bit, and the result is still stored in QB0 *)</p> <p style="text-align: center;">(* If CR is 0: <i>SHR</i> instruction is not executed *)</p>	
Result	<p>For the LD example, if <i>SHR</i> instruction is executed, the result will be as the following:</p> <p style="margin-left: 40px;">QB0 B#2#10000001</p> <p style="margin-left: 80px;">After 1st shift After 2nd shift After 3rd shift After 4th shift</p> <p style="margin-left: 40px;">QB0 B#2#01000000 B#2#00100000 B#2#00010000 B#2#00001000</p>	

6.6.4 ROR (Rotate right)

➤ Description

	Name	Usage	Group	
LD	ROR			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	ROR	ROR <i>OUT, N</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE, WORD, DWORD	I, Q, M, V, L, SM, constant, pointer
<i>N</i>	Input	BYTE	I, Q, M, V, L, SM, constant, pointer
<i>OUT</i>	Output	BYTE, WORD, DWORD	Q, M, V, L, SM, pointer

■ LD

The *IN* and *OUT* must be of the same data type.

If *EN* is 1, this instruction rotates the value of *IN* to the right by *N* bits, and the LSB is rotated to the MSB. The result is assigned to *OUT*. If *EN* is 0, this instruction is not executed.

■ IL

If *CR* is 1, this instruction rotates the value of *OUT* to the right by *N* bits, and the LSB is rotated to the MSB.

The result is still stored in *OUT*. It does not influence *CR*.

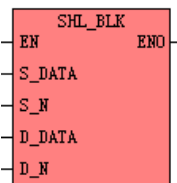
If *CR* is 0, this instruction is not executed.

➤ Examples

LD		<p>If M0.0 is 0: <i>ROR</i> isn't executed.</p> <p>If M0.0 is 1: <i>ROR</i> rotates QB0 to the right by 1 bit, and the LSB is rotated to the MSB. The result is still assigned to QB0.</p>
IL	<p>LD %M0.0 (* CR is created with M0.0 *)</p> <p>ROL %QB0, B#1 (* If CR is 1: <i>ROR</i> rotates QB0 to the right by 1 bit, and the result is still stored in QB0 *)</p> <p style="text-align: center;">(* If CR is 0: <i>ROR</i> instruction is not executed *)</p>	
Result	<p>For the LD example, if <i>ROR</i> instruction is executed, the result will be as the following:</p> <p>QB0 B#2#10100001</p> <p style="text-align: center;">After 1st shift After 2nd shift After 3rd shift After 4th shift</p> <p>QB0 B#2#11010000 B#2#01101000 B#2#00110100 B#2#00011010</p>	

6.6.5 SHL_BLK (Bit String Shift Left)

➤ Description

	Name	Usage	Group	
LD	SHL_BLK			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	SHL_BLK	SHL_BLK S_DATA, S_N, D_DATA, D_N	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>S_DATA</i>	Input	BOOL	I, Q, M, V, L
<i>S_N</i>	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constant, Pointer
<i>D_DATA</i>	Input/Output	BOOL	Q, M, V, L
<i>D_N</i>	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constant, Pointer

This instruction shifts the number *D_N* of continuous bits, beginning with *D_DATA*, to the left by *S_N* bits. Meanwhile, the number *S_N* of continuous bits, beginning with *S_DATA*, are filled into the right most bits of *D_DATA*.

■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

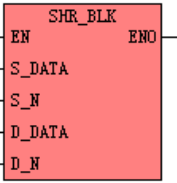
➤ Examples

LD	<pre> (* Network 0 *) (* Initialize the variables *) %SM0.1 ----- MOVE -----> (NUL) EN MOVE ENO IN OUT %VW100 16#5A6B ----- MOVE -----> (NUL) EN MOVE ENO IN OUT %VW102 16#7C8D </pre> <pre> (* Network 1 *) (* shift once at each rising edge of I0.0 *) %I0.0 ----- R_TRIG -----> CLK R_TRIG Q ----- SHL_BLK -----> (NUL) EN SHL_BLK ENO %V100.0 S_DATA 4 S_N %V102.0 D_DATA 16 D_N </pre>
IL	<pre> (* Network 0 *) (*Initialize the variables*) LD %SM0.1 MOVE 16#5A6B, %VW100 MOVE 16#7C8D, %VW102 (* Network 1 *) (*shift once at each rising edge of I0.0*) LD %I0.0 R_TRIG SHL_BLK %V100.0, 4, %V102.0, 16 </pre>

Result	The result is shown as the following:																
	VW102								VW100								
	V103.7				V102.0				V101.7				V100.0				
	Initial value	0111	1100	1000	1101												
	After the 1st execution	1100	1000	1101	1011												
After the 2nd execution	1000	1101	1011	1011													
After the 3rd execution	1101	1011	1011	1011													

6.6.6 SHR_BLK (Bit String Shift Right)

➤ Description

	Name	Usage	Group	
LD	SHR_BLK			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	SHR_BLK	SHR_BLK S_DATA, S_N, D_DATA, D_N	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>S_DATA</i>	Input	BOOL	I, Q, M, V, L
<i>S_N</i>	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constant, Pointer
<i>D_DATA</i>	Input/Output	BOOL	Q, M, V, L
<i>D_N</i>	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constant, Pointer

This instruction shifts the number *D_N* of continuous bits, beginning with *D_DATA*, to the right by *S_N* bits.

Meanwhile, the number S_N of continuous bits, beginning with S_DATA , are filled into the left most bits of D_DATA .

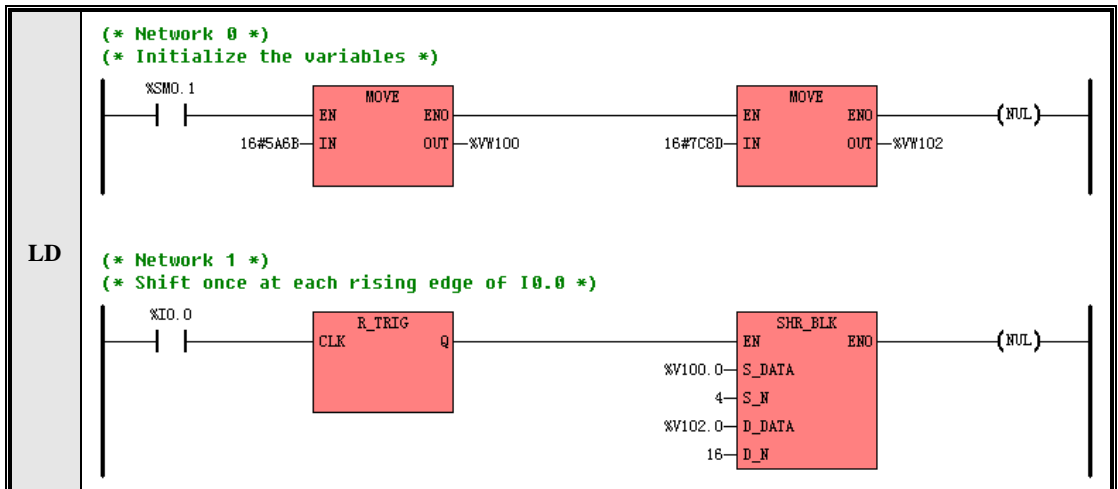
■ **LD**

If EN is 1, this instruction is executed.

■ **IL**

If CR is 1, this instruction is executed, and it does not influence CR .

➤ Examples

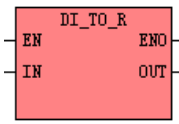


IL	<p>(* Network 0 *)</p> <p>(*Initialize the variables*)</p> <pre>LD %SM0.1 MOVE 16#5A6B, %VW100 MOVE 16#7C8D, %VW102</pre> <p>(* Network 1 *)</p> <p>(*Shift once at each rising edge of I0.0*)</p> <pre>LD %I0.0 R_TRIG SHR_BLK %V100.0, 4, %V102.0, 16</pre>																																																						
Result	<p>The result is shown as the following:</p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th></th> <th colspan="4">VW102</th> <th colspan="4">VW100</th> </tr> <tr> <th></th> <th colspan="2">V103.7</th> <th colspan="2">V102.0</th> <th colspan="2">V101.7</th> <th colspan="2">V100.0</th> </tr> </thead> <tbody> <tr> <td>Initial value</td> <td>0111</td><td>1100</td><td>1000</td><td>1101</td> <td>0101</td><td>1010</td><td>0110</td><td>1011</td> </tr> <tr> <td>After the 1st execution</td> <td>1011</td><td>0111</td><td>1100</td><td>1000</td> <td></td><td></td><td></td><td></td> </tr> <tr> <td>After the 2nd execution</td> <td>1011</td><td>1011</td><td>0111</td><td>1100</td> <td></td><td></td><td></td><td></td> </tr> <tr> <td>After the 3rd execution</td> <td>1011</td><td>1011</td><td>1011</td><td>0111</td> <td></td><td></td><td></td><td></td> </tr> </tbody> </table>		VW102				VW100					V103.7		V102.0		V101.7		V100.0		Initial value	0111	1100	1000	1101	0101	1010	0110	1011	After the 1st execution	1011	0111	1100	1000					After the 2nd execution	1011	1011	0111	1100					After the 3rd execution	1011	1011	1011	0111				
	VW102				VW100																																																		
	V103.7		V102.0		V101.7		V100.0																																																
Initial value	0111	1100	1000	1101	0101	1010	0110	1011																																															
After the 1st execution	1011	0111	1100	1000																																																			
After the 2nd execution	1011	1011	0111	1100																																																			
After the 3rd execution	1011	1011	1011	0111																																																			

6.7 Convert Instructions

6.7.1 DI_TO_R (DINT To REAL)

➤ Description

	Name	Usage	Group	
LD	DI_TO_R			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	DI_TO_R	DI_TO_R <i>IN, OUT</i>	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	DINT	I, Q, M, V, L, SM, HC, Constant
<i>OUT</i>	Output	REAL	V, L

This instruction converts a DINT value (*IN*) to a REAL value and assigns the result to *OUT*.

■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is not executed, and it does not influence *CR*.

➤ Examples

LD		<p>SM0.0 is always ON, therefore <i>DI_TO_R</i> is always executed: The value of MD0 is converted to a REAL value that is assigned to VR100.</p>						
IL	<p>LD %SM0.0 (* CR is created SM0.0 *)</p> <p>DI_TO_R %MD0, %VR100 (* The value of MD0 is converted to a REAL value that is assigned to VR100 *)</p>							
Result	<p>The result is shown as the following:</p> <table border="1" style="margin: auto; border-collapse: collapse;"> <tr> <td style="padding: 5px;">MD0</td> <td style="padding: 5px;">VR100</td> </tr> <tr> <td style="padding: 5px; text-align: center;">DI#123</td> <td style="padding: 5px; text-align: center;">123.0</td> </tr> <tr> <td style="padding: 5px; text-align: center;">DI#-9876</td> <td style="padding: 5px; text-align: center;">-9876.0</td> </tr> </table>		MD0	VR100	DI#123	123.0	DI#-9876	-9876.0
MD0	VR100							
DI#123	123.0							
DI#-9876	-9876.0							

6.7.2 R_TO_DI (REAL To DINT)

➤ Description

	Name	Usage	Group	
LD	R_TO_DI			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA

IL	R_TO_DI	R_TO_DI <i>IN, OUT</i>	U	<input checked="" type="checkbox"/> CPU508
----	---------	------------------------	---	--------------------------------------------

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	REAL	V, L, constant
<i>OUT</i>	Output	DINT	M, V, L, SM

This instruction converts a REAL value (*IN*) to a DINT value and assigns the result to *OUT*. During the conversion, the decimal fraction is cut off.

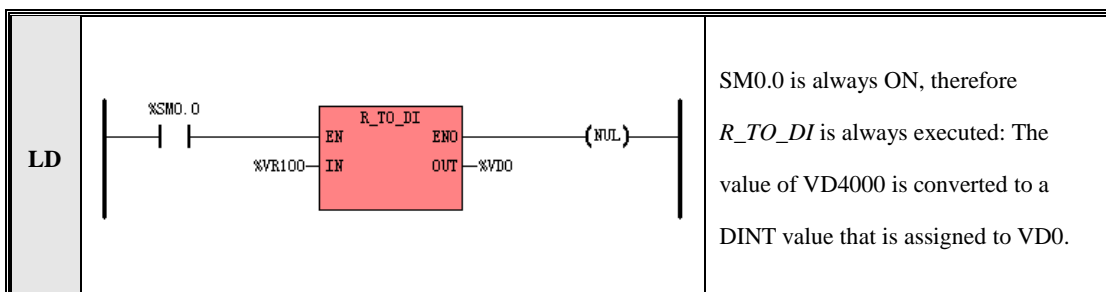
■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

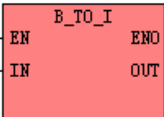
➤ Examples



IL	<p>LD %SM0.0 (* CR is created SM0.0 *)</p> <p>R_TO_DI %VD4000, %VD0 (* The value of VD4000 is converted to a DINT value that is assigned to VD0 *)</p>						
Result	<p>The result is shown as the following:</p> <div style="border: 1px solid black; padding: 10px; margin: 10px auto; width: fit-content;"> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center; width: 50%;">VR100</td> <td style="text-align: center; width: 50%;">VD0</td> </tr> <tr> <td style="text-align: center; border: 1px solid black; padding: 2px;">123.4</td> <td style="text-align: center; border: 1px solid black; padding: 2px;">DI#123</td> </tr> <tr> <td style="text-align: center; border: 1px solid black; padding: 2px;">5213.6</td> <td style="text-align: center; border: 1px solid black; padding: 2px;">DI#5214</td> </tr> </table> </div>	VR100	VD0	123.4	DI#123	5213.6	DI#5214
VR100	VD0						
123.4	DI#123						
5213.6	DI#5214						

6.7.3 B_TO_I (BYTE To INT)

➤ Description

	Name	Usage	Group	
LD	B_TO_I			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA

IL	B_TO_I	B_TO_I <i>IN, OUT</i>	U	<input checked="" type="checkbox"/> CPU508
----	--------	-----------------------	---	--------------------------------------------

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE	I, Q, M, V, L, SM, Constant
<i>OUT</i>	Output	INT	Q, M, V, L, SM, AQ

This instruction converts the input byte *IN* to an integer value and assigns the result to *OUT*.

■ **LD**

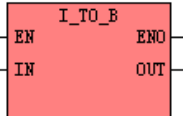
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.7.4 I_TO_B (INT To BYTE)

➤ Description

	Name	Usage	Group
LD	I_TO_B		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA

IL	I_TO_B	I_TO_B <i>IN, OUT</i>	U	<input checked="" type="checkbox"/> CPU508
----	--------	-----------------------	---	--------------------------------------------

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	INT	I, Q, M, V, L, SM, AI, AQ, T, C, Constant
<i>OUT</i>	Output	BYTE	Q, M, V, L, SM

This instruction assigns the least byte of the input *IN* to the *OUT*.

■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

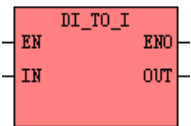
If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

<p>LD</p>		<p>SM0.0 is always 1, so I_TO_B is always be executed: assigns VB0 (the least byte of VW0) to VB10.</p>								
<p>IL</p>	<pre>LD %SM0.0 I_TO_B %VW0, %VB10</pre>									
<p>Result</p>	<p>The result is shown as the following:</p> <table border="1" data-bbox="312 769 868 1064"> <thead> <tr> <th>VW0</th> <th>VB10</th> </tr> </thead> <tbody> <tr> <td>24</td> <td>B#24</td> </tr> <tr> <td>255</td> <td>B#255</td> </tr> <tr> <td>I#16#FFFD</td> <td>B#16#FD</td> </tr> </tbody> </table>		VW0	VB10	24	B#24	255	B#255	I#16#FFFD	B#16#FD
VW0	VB10									
24	B#24									
255	B#255									
I#16#FFFD	B#16#FD									

6.7.5 DI_TO_I (DINT To INT)

➤ Description

	Name	Usage	Group	
LD	DI_TO_I			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	DI_TO_I	DI_TO_I <i>IN, OUT</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	DINT	I, Q, M, V, L, SM, HC, Constant
<i>OUT</i>	Output	INT	Q, M, V, L, SM, AQ

This instruction assigns the least word of the input *IN* to the *OUT*.

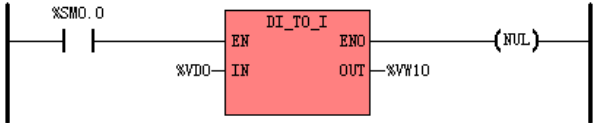
■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

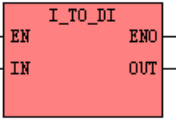
If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

<p>LD</p>		<p>SM0.0 is always 1, so DI_TO_I is always be executed: assigns VW0 (the least word of VD0) to VW10.</p>								
<p>IL</p>	<pre>LD %SM0.0 DI_TO_I %VD0, %VW10</pre>									
<p>Result</p>	<p>The result is shown as the following:</p> <table border="1" data-bbox="312 777 893 1055"> <thead> <tr> <th>VD0</th> <th>VW10</th> </tr> </thead> <tbody> <tr> <td>DI#12345</td> <td>12345</td> </tr> <tr> <td>DI#-234</td> <td>-234</td> </tr> <tr> <td>DI#16#7A8B9C1D</td> <td>I#16#9C1D</td> </tr> </tbody> </table>		VD0	VW10	DI#12345	12345	DI#-234	-234	DI#16#7A8B9C1D	I#16#9C1D
VD0	VW10									
DI#12345	12345									
DI#-234	-234									
DI#16#7A8B9C1D	I#16#9C1D									

6.7.6 I_TO_DI (INT To DINT)

➤ Description

	Name	Usage	Group	<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
LD	I_TO_DI			
IL	I_TO_DI	I_TO_DI <i>IN, OUT</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	INT	I, Q, M, V, L, SM, AI, AQ, T, C, Constant
<i>OUT</i>	Output	DINT	Q, M, V, L, SM

This instruction converts the input integer *IN* to a DINT value and assigns the result to *OUT*.

■ **LD**

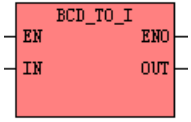
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.7.7 BCD_TO_I (BCD To INT)

➤ Description

	Name	Usage	Group	<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
LD	BCD_TO_I			
IL	BCD_TO_I	BCD_TO_I <i>IN, OUT</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>OUT</i>	Output	INT	Q, M, V, L, SM, AQ

This instruction converts the input Binary-Coded Decimal value (*IN*) to an integer value and assigns the result to the *OUT*.

Note: The 8421 codes are adopted for the BCD code. The valid range of *IN* is 0 to 9999 BCD.

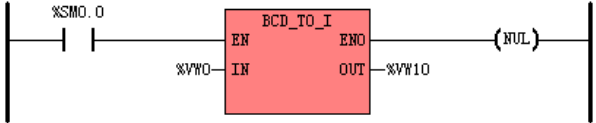
■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

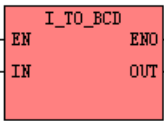
If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

LD		<p>SM0.0 is always 1, so BCD_TO_I is always be executed: converts VW0 from BCD to an integer and assigns it to VW10.</p>								
IL	<pre>LD %SM0.0 BCD_TO_I %VW0, %VW10</pre>									
Result	<p>The result is shown as the following:</p> <table border="1" data-bbox="312 786 720 1046" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">VW0</th> <th style="text-align: center;">VW10</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">16#99</td> <td style="text-align: center;">99</td> </tr> <tr> <td style="text-align: center;">16#4567</td> <td style="text-align: center;">4567</td> </tr> <tr> <td style="text-align: center;">16#9999</td> <td style="text-align: center;">9999</td> </tr> </tbody> </table>		VW0	VW10	16#99	99	16#4567	4567	16#9999	9999
VW0	VW10									
16#99	99									
16#4567	4567									
16#9999	9999									

6.7.8 I_TO_BCD (INT To BCD)

➤ Description

	Name	Usage	Group	
LD	I_TO_BCD			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	I_TO_BCD	I_TO_BCD <i>IN, OUT</i>	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	INT	I, Q, M, V, L, SM, AI, AQ, T, C, Constant
<i>OUT</i>	Output	WORD	Q, M, V, L, SM

This instruction converts the input integer value (*IN*) to a Binary-Coded Decimal value and assigns the result to the *OUT*.

Note: The 8421 codes are adopted for the BCD code. The valid range of *IN* is 0 to 9999.

■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

LD		<p>SM0.0 is always 1, so I_TO_BCD is always be executed: converts VW0 to a BCD value and assigns it to VW10.</p>								
IL	<pre>LD %SM0.0 I_TO_BCD %VW0, %VW10</pre>									
Result	<p>The result is shown as the following:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">VW0</th> <th style="text-align: center;">VW10</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">99</td> <td style="text-align: center;">16# 99</td> </tr> <tr> <td style="text-align: center;">4567</td> <td style="text-align: center;">16# 4567</td> </tr> <tr> <td style="text-align: center;">9999</td> <td style="text-align: center;">16# 9999</td> </tr> </tbody> </table>		VW0	VW10	99	16# 99	4567	16# 4567	9999	16# 9999
VW0	VW10									
99	16# 99									
4567	16# 4567									
9999	16# 9999									

6.7.9 I_TO_A (INT To ASCII)

➤ Description

	Name	Usage	Group	<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
LD	I_TO_A			
IL	I_TO_A	I_TO_A IN, OUT, FMT	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	INT	I, Q, M, V, L, SM, AI, AQ, T, C, Constant
<i>FMT</i>	Input	BYTE	I, Q, M, V, L, SM
<i>OUT</i>	Output	BYTE	Q, M, V, L, SM

This instruction converts an integer value (*IN*) to an ASCII string, then formats the string according to *FMT* and put the result into the Output Buffer beginning with *OUT*. The conversion result of a positive value does not include any sign, and the conversion result of a negative value begins with a leading minus sign (-).

The *OUT* defines the starting address of the Output Buffer, which occupies a memory range of 8 successive bytes. In the buffer, the strings are right alignment, and the free bytes are filled with spaces (whose ASCII is 32).

The *FMT* is used to format the string, and the rules are shown in the figure below:

MSB								LSB
7	6	5	4	3	2	1	0	
0	0	0	0	c	n	n	n	

- (1) *nmn* --- This field specifies the number of digits of the decimal part.
Its available rang is 0 to 5. 0 stands for no decimal part.
- (2) *c* --- This field specifies the separator between the whole number and the fraction:
0 for a decimal point (whose ASCII is 46), and 1 for a comma(whose ASCII is 44).
- (3) The upper 4 bits must be zero.

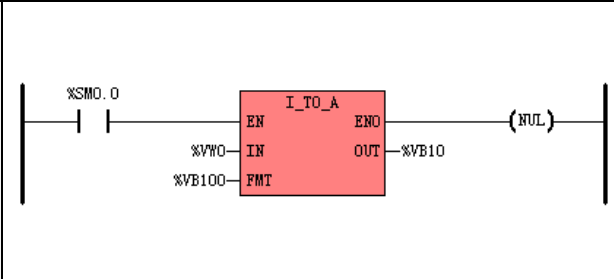
■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

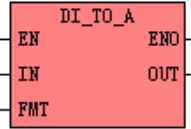
If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

LD		<p>SM0.0 is always 1, so the instruction I_TO_A is always executed: converts the value of VW0 to a string, and format the string and put the result to a buffer beginning with VB10.</p>																																																												
IL	<pre>LD %SM0.0 I_TO_A %VW0, %VB10, %VB100</pre>																																																													
Result	<p>The result is as the following:</p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="padding: 5px;">VB100</th> <th style="padding: 5px;">VW0</th> <th colspan="8" style="padding: 5px;">Result</th> </tr> <tr> <th colspan="2"></th> <th colspan="4" style="padding: 5px;">VB10</th> <th colspan="4" style="padding: 5px;">VB17</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">B#3</td> <td style="padding: 5px;">12</td> <td style="padding: 5px;">32</td><td style="padding: 5px;">32</td><td style="padding: 5px;">32</td><td style="padding: 5px;">48</td><td style="padding: 5px;">46</td><td style="padding: 5px;">48</td><td style="padding: 5px;">49</td><td style="padding: 5px;">50</td> </tr> <tr> <td colspan="2"></td> <td colspan="8" style="padding: 5px;">‘ ’ ‘ ’ ‘ ’ ‘0’ ‘.’ ‘0’ ‘1’ ‘2’</td> </tr> <tr> <td></td> <td style="padding: 5px;">-23456</td> <td style="padding: 5px;">32</td><td style="padding: 5px;">45</td><td style="padding: 5px;">50</td><td style="padding: 5px;">51</td><td style="padding: 5px;">46</td><td style="padding: 5px;">52</td><td style="padding: 5px;">53</td><td style="padding: 5px;">54</td> </tr> <tr> <td colspan="2"></td> <td colspan="8" style="padding: 5px;">‘ ’ ‘_’ ‘2’ ‘3’ ‘.’ ‘4’ ‘5’ ‘6’</td> </tr> </tbody> </table>		VB100	VW0	Result										VB10				VB17				B#3	12	32	32	32	48	46	48	49	50			‘ ’ ‘ ’ ‘ ’ ‘0’ ‘.’ ‘0’ ‘1’ ‘2’									-23456	32	45	50	51	46	52	53	54			‘ ’ ‘_’ ‘2’ ‘3’ ‘.’ ‘4’ ‘5’ ‘6’							
VB100	VW0	Result																																																												
		VB10				VB17																																																								
B#3	12	32	32	32	48	46	48	49	50																																																					
		‘ ’ ‘ ’ ‘ ’ ‘0’ ‘.’ ‘0’ ‘1’ ‘2’																																																												
	-23456	32	45	50	51	46	52	53	54																																																					
		‘ ’ ‘_’ ‘2’ ‘3’ ‘.’ ‘4’ ‘5’ ‘6’																																																												

6.7.10 DI_TO_A (DINT To ASCII)

➤ Description

	Name	Usage	Group	
LD	DI_TO_A			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	DI_TO_A	DI_TO_A IN, OUT, FMT	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	DINT	I, Q, M, V, L, SM, HC, Constants
<i>FMT</i>	Input	BYTE	I, Q, M, V, L, SM
<i>OUT</i>	Output	BYTE	Q, M, V, L, SM

This instruction converts a DINT value (*IN*) to an ASCII string, then formats the string according to *FMT* and put the result into the Output Buffer beginning with *OUT*. The conversion result of a positive value does not include any sign, and the conversion result of a negative value begins with a leading minus sign (-).

The *OUT* defines the starting address of the Output Buffer, which occupies a memory range of 12 successive bytes. In the buffer, the strings are right alignment, and the free bytes are filled with spaces (whose ASCII is 32).

The *FMT* is used to format the string, and the rules are shown in the figure below:

MSB									LSB
7	6	5	4	3	2	1	0		
0	0	0	0	c	n	n	n		

- (1) *nnn* --- This field specifies the number of digits of the decimal part.
Its available rang is 0 to 5. 0 stands for no decimal part.
- (2) *c* --- This field specifies the separator between the whole number and the fraction:
0 for a decimal point (whose ASCII is 46), and 1 for a comma(whose ASCII is 44).
- (3) The upper 4 bits must be zero.

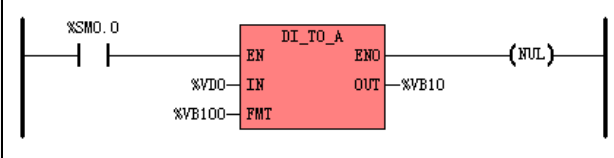
■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

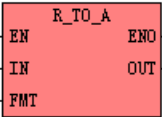
If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

LD		<p>SM0.0 is always 1, so the instruction DI_TO_A is always executed: Convert the value of VD0 to a string, and format the string and put it to a buffer beginning with VB10.</p>																																																																																
IL	<pre>LD %SM0.0 DI_TO_A %VD0, %VB10, %VB100</pre>																																																																																	
Result	<p>The result is as the following:</p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="width: 10%;">VB 100</th> <th style="width: 10%;">VD0</th> <th colspan="10" style="width: 60%;">Result</th> </tr> <tr> <td></td> <td></td> <th colspan="5">VB 10</th> <th colspan="5">VB 21</th> </tr> </thead> <tbody> <tr> <td style="border: 1px solid black; padding: 2px;">B # 3</td> <td style="border: 1px solid black; padding: 2px;">DI#12</td> <td>32</td><td>32</td><td>32</td><td>32</td><td>32</td><td>32</td><td>32</td><td>48</td><td>46</td><td>48</td><td>49</td><td>50</td> </tr> <tr> <td></td> <td></td> <td>'</td><td>'</td><td>'</td><td>'</td><td>'</td><td>'</td><td>'</td><td>'0'</td><td>'</td><td>'0'</td><td>'1'</td><td>'2'</td> </tr> <tr> <td></td> <td style="border: 1px solid black; padding: 2px;">DI#123456</td> <td>32</td><td>32</td><td>32</td><td>32</td><td>45</td><td>49</td><td>50</td><td>51</td><td>46</td><td>52</td><td>53</td><td>54</td> </tr> <tr> <td></td> <td></td> <td>'</td><td>'</td><td>'</td><td>'</td><td>'</td><td>'</td><td>'</td><td>'</td><td>'</td><td>'4'</td><td>'5'</td><td>'6'</td> </tr> </tbody> </table>		VB 100	VD0	Result												VB 10					VB 21					B # 3	DI#12	32	32	32	32	32	32	32	48	46	48	49	50			'	'	'	'	'	'	'	'0'	'	'0'	'1'	'2'		DI#123456	32	32	32	32	45	49	50	51	46	52	53	54			'	'	'	'	'	'	'	'	'	'4'	'5'	'6'
VB 100	VD0	Result																																																																																
		VB 10					VB 21																																																																											
B # 3	DI#12	32	32	32	32	32	32	32	48	46	48	49	50																																																																					
		'	'	'	'	'	'	'	'0'	'	'0'	'1'	'2'																																																																					
	DI#123456	32	32	32	32	45	49	50	51	46	52	53	54																																																																					
		'	'	'	'	'	'	'	'	'	'4'	'5'	'6'																																																																					

6.7.11 R_TO_A (REAL To ASCII)

➤ Description

	Name	Usage	Group	<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
LD	R_TO_A			
IL	R_TO_A	R_TO_A IN, OUT, FMT	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	REAL	V, L, Constants
<i>FMT</i>	Input	BYTE	I, Q, M, V, L, SM
<i>OUT</i>	Output	BYTE	Q, M, V, L, SM

This instruction converts a REAL value (*IN*) to an ASCII string, then formats the string according to *FMT* and put the result into the Output Buffer beginning with *OUT*. The conversion result of a positive value does not include any sign, and the conversion result of a negative value begins with a leading minus sign (-). If the digits of the decimal part of *IN* is larger than the *nnn* in *FMT*, which specifies the digits of the decimal part in the string, then *IN* is round off before being converted. Otherwise, if it is less than *nnn*, the missing digits of the decimal part are filled with 0 in the string.

The *OUT* defines the starting address of the Output Buffer, whose size is specified in *FMT*. In the buffer, the strings are right alignment, and the free bytes are filled with spaces (whose ASCII is 32).

The *FMT* is used to format the string, and the rules are shown in the figure below:

MSB				LSB			
7	6	5	4	3	2	1	0
s	s	s	s	c	n	n	n

(1) *nnn* --- This field specifies the number of digits of the decimal part.
Its available rang is 0 to 5. 0 stands for no decimal part.

(2) *c* --- This field specifies the separator between the whole number and the fraction:
0 for a decimal point (whose ASCII is 46), and 1 for a comma(whose ASCII is 44).

(3) *ssss* --- This field specifies the size of the buffer.
Its available rang is 3 to 15, and it must be greater than *nnn*.

Note: If the length of the resulting string exceeds the length of the Output Buffer, then the whole buffer will be filled with spaces (whose ASCII is 32).

■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

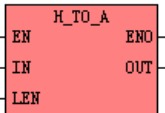
If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

LD		<p>SM0.0 is always 1, so the instruction DI_TO_A is always executed: Convert the value of VR0 to a string, and format the string and put it to a buffer beginning with VB10.</p>																																																												
IL	<pre>LD %SM0.0 R_TO_A %VR0, %VB10, %VB100</pre>																																																													
Result	<p>The result is as the following:</p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="padding: 5px;">VB100</th> <th style="padding: 5px;">VR0</th> <th colspan="8" style="padding: 5px;">Result</th> </tr> <tr> <th style="padding: 5px;"></th> <th style="padding: 5px;"></th> <th colspan="4" style="padding: 5px;">VB10</th> <th colspan="4" style="padding: 5px;">VB17</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">B#16#83</td> <td style="padding: 5px;">123.4</td> <td style="padding: 5px;">32</td><td style="padding: 5px;">49</td><td style="padding: 5px;">50</td><td style="padding: 5px;">51</td> <td style="padding: 5px;">46</td><td style="padding: 5px;">52</td><td style="padding: 5px;">48</td><td style="padding: 5px;">48</td> </tr> <tr> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> <td style="padding: 5px;">‘</td><td style="padding: 5px;">‘1’</td><td style="padding: 5px;">‘2’</td><td style="padding: 5px;">‘3’</td> <td style="padding: 5px;">‘.’</td><td style="padding: 5px;">‘4’</td><td style="padding: 5px;">‘0’</td><td style="padding: 5px;">‘0’</td> </tr> <tr> <td style="padding: 5px;"></td> <td style="padding: 5px;">-123.4567</td> <td style="padding: 5px;">45</td><td style="padding: 5px;">49</td><td style="padding: 5px;">50</td><td style="padding: 5px;">51</td> <td style="padding: 5px;">46</td><td style="padding: 5px;">52</td><td style="padding: 5px;">53</td><td style="padding: 5px;">55</td> </tr> <tr> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> <td style="padding: 5px;">‘-’</td><td style="padding: 5px;">‘1’</td><td style="padding: 5px;">‘2’</td><td style="padding: 5px;">‘3’</td> <td style="padding: 5px;">‘.’</td><td style="padding: 5px;">‘4’</td><td style="padding: 5px;">‘5’</td><td style="padding: 5px;">‘7’</td> </tr> </tbody> </table>		VB100	VR0	Result										VB10				VB17				B#16#83	123.4	32	49	50	51	46	52	48	48			‘	‘1’	‘2’	‘3’	‘.’	‘4’	‘0’	‘0’		-123.4567	45	49	50	51	46	52	53	55			‘-’	‘1’	‘2’	‘3’	‘.’	‘4’	‘5’	‘7’
VB100	VR0	Result																																																												
		VB10				VB17																																																								
B#16#83	123.4	32	49	50	51	46	52	48	48																																																					
		‘	‘1’	‘2’	‘3’	‘.’	‘4’	‘0’	‘0’																																																					
	-123.4567	45	49	50	51	46	52	53	55																																																					
		‘-’	‘1’	‘2’	‘3’	‘.’	‘4’	‘5’	‘7’																																																					

6.7.12 H_TO_A (Hexadecimal To ASCII)

➤ Description

	Name	Usage	Group	<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
LD	H_TO_A			
IL	H_TO_A	H_TO_A IN, OUT, LEN	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE	I, Q, M, V, L, SM
<i>LEN</i>	Input	BYTE	I, Q, M, V, L, SM, Constants
<i>OUT</i>	Output	BYTE	Q, M, V, L, SM

This instruction converts the number *LEN* of hexadecimal digits, beginning with *IN*, to an ASCII string, and put the string into the Output Buffer beginning with *OUT*.

Note: Every 4 binary digits makes 1 hexadecimal digit, so every input byte includes 2 hexadecimal digits, and so the size of the Output Buffer occupies is $LEN*2$ bytes.

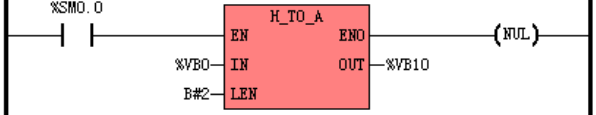
■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

LD		<p>SM0.0 is always 1, so H_TO_A is always executed: converts 2-bytes hexadecimal digits, beginning with VB0, to a string and put the result into the buffer which occupies 4 continuous bytes beginning with VB10.</p>																																																
IL	<p>LD %SM0.0 H_TO_A %VB0, %VB10, B#2</p>																																																	
Result	<p>The result is as the following:</p> <table border="1" data-bbox="300 847 856 1121" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2" style="text-align: center;">VB0</th> <th colspan="2" style="text-align: center;">VB1</th> <th colspan="4" style="text-align: center;">Result</th> </tr> <tr> <th colspan="2"></th> <th colspan="2"></th> <th colspan="2" style="text-align: center;">VB10</th> <th colspan="2" style="text-align: center;">VB13</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">B#16#1A</td> <td style="text-align: center;">B#16#2B</td> <td colspan="2"></td> <td style="text-align: center;">49</td> <td style="text-align: center;">65</td> <td style="text-align: center;">50</td> <td style="text-align: center;">66</td> </tr> <tr> <td colspan="2"></td> <td colspan="2"></td> <td style="text-align: center;">‘1’</td> <td style="text-align: center;">‘A’</td> <td style="text-align: center;">‘2’</td> <td style="text-align: center;">‘B’</td> </tr> <tr> <td style="text-align: center;">B#16#7C</td> <td style="text-align: center;">B#16#8D</td> <td colspan="2"></td> <td style="text-align: center;">55</td> <td style="text-align: center;">67</td> <td style="text-align: center;">56</td> <td style="text-align: center;">68</td> </tr> <tr> <td colspan="2"></td> <td colspan="2"></td> <td style="text-align: center;">‘7’</td> <td style="text-align: center;">‘C’</td> <td style="text-align: center;">‘8’</td> <td style="text-align: center;">‘D’</td> </tr> </tbody> </table>		VB0		VB1		Result								VB10		VB13		B#16#1A	B#16#2B			49	65	50	66					‘1’	‘A’	‘2’	‘B’	B#16#7C	B#16#8D			55	67	56	68					‘7’	‘C’	‘8’	‘D’
VB0		VB1		Result																																														
				VB10		VB13																																												
B#16#1A	B#16#2B			49	65	50	66																																											
				‘1’	‘A’	‘2’	‘B’																																											
B#16#7C	B#16#8D			55	67	56	68																																											
				‘7’	‘C’	‘8’	‘D’																																											

6.7.13 A_TO_H (ASCII to Hexadecimal)

➤ Description

	Name	Usage	Group	
LD	A_TO_H			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	A_TO_H	A_TO_H IN, OUT, LEN	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE	I, Q, M, V, L, SM
<i>LEN</i>	Input	BYTE	I, Q, M, V, L, SM, Constants
<i>OUT</i>	Output	BYTE	Q, M, V, L, SM

This instruction converts the number *LEN* of ASCII characters, beginning with *IN*, to hexadecimal digits, and put the digits into the Output Buffer beginning with *OUT*. Note: Every 4 binary digits makes 1 hexadecimal digit, so every input byte, which stands for an ASCII character, occupies 4 binary digits of memory space (i.e., a half byte) in the Output Buffer.

The valid ASCII input range is: B#16#30 to B#16#39 (stands for the characters 0 to 9), B#16#41 to B#16#46 (stands for the characters A to F).

ASCII to Hexadecimal

■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

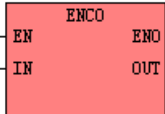
If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

LD		<p>SM0.0 is always 1, so A_TO_H is always executed: converts the 3-bytes ASCII string, beginning with VB0, to hexadecimal digits, and put the result into the Output Buffer beginning with VB100.</p>																									
IL	<pre>LD %SM0.0 A_TO_H %VB0, %VB10, B#3</pre>																										
Result	<p>The result is as the following:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>VB0</th> <th>VB1</th> <th>VB2</th> <th>VB10</th> <th>VB11</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">51</td> <td style="text-align: center;">56</td> <td style="text-align: center;">54</td> <td style="text-align: center;">B#16#38</td> <td style="text-align: center;">B#16#6x</td> </tr> <tr> <td style="text-align: center;">'3'</td> <td style="text-align: center;">'8'</td> <td style="text-align: center;">'6'</td> <td></td> <td></td> </tr> <tr> <td style="text-align: center;">55</td> <td style="text-align: center;">65</td> <td style="text-align: center;">49</td> <td style="text-align: center;">B#16#7A</td> <td style="text-align: center;">B#16#1x</td> </tr> <tr> <td style="text-align: center;">'7'</td> <td style="text-align: center;">'A'</td> <td style="text-align: center;">'1'</td> <td></td> <td></td> </tr> </tbody> </table> <p>Note: x stands for this half byte (4 bits) keeps the original value.</p>		VB0	VB1	VB2	VB10	VB11	51	56	54	B#16#38	B#16#6x	'3'	'8'	'6'			55	65	49	B#16#7A	B#16#1x	'7'	'A'	'1'		
VB0	VB1	VB2	VB10	VB11																							
51	56	54	B#16#38	B#16#6x																							
'3'	'8'	'6'																									
55	65	49	B#16#7A	B#16#1x																							
'7'	'A'	'1'																									

6.7.14 ENCO (Encoding)

➤ Description

	Name	Usage	Group	
LD	ENCO			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	ENCO	ENCO IN, OUT	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>OUT</i>	Output	BYTE	Q, M, V, L, SM

This instruction checks the input Word *IN* from the least significant bit, and writes the bit number of the first bit equal to 1 into the output byte *OUT*. Note: If the value of *IN* is 0, the result is meaningless.

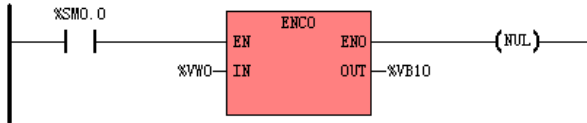
■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

<p>LD</p>		<p>SM0.0 is always 1, so ENCO is always executed: writes the bit number of the first bit equal to 1 into VB10.</p>																																																																				
<p>IL</p>	<p>LD %SM0.0 ENCO %VW0, %VB10</p>																																																																					
<p>Result</p>	<p>The result is as the following:</p> <table border="1" data-bbox="299 789 1039 1010"> <thead> <tr> <th colspan="13">VW0</th> <th colspan="2">VB10</th> </tr> <tr> <th>(MSB)</th> <th>15</th> <th>12</th> <th>9</th> <th>4</th> <th>0</th> <th>(LSB)</th> <th colspan="6"></th> <th>B#9</th> <th>B#4</th> </tr> </thead> <tbody> <tr> <td></td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td></td> </tr> <tr> <td></td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td></td> </tr> </tbody> </table>		VW0													VB10		(MSB)	15	12	9	4	0	(LSB)							B#9	B#4		0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0			0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	
VW0													VB10																																																									
(MSB)	15	12	9	4	0	(LSB)							B#9	B#4																																																								
	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0																																																					
	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0																																																					

6.7.15 DECO (Decoding)

➤ Description

	Name	Usage	Group	
LD	DECO			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	DECO	DECO IN, OUT	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE	I, Q, M, V, L, SM, Constant
<i>OUT</i>	Output	WORD	Q, M, V, L, SM

This instruction sets the bit in the output word *OUT* that corresponds to the bit number represented by the least significant “nibble” (4 bits) of the input byte *IN*. All other bits in the *OUT* are reset.

■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

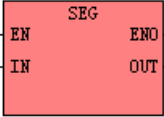
If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

<p style="text-align: center;">LD</p>		<p>SM0.0 is always 1, so DECO is always executed: sets the bit in VW10 which corresponds to the bit number represented by the least significant “nibble” of VB0.</p>																																						
<p style="text-align: center;">IL</p>	<p>LD %SM0.0 DECO %VB0, %VW10</p>																																							
<p style="text-align: center;">Result</p>	<p>The result is as the following:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: left;">VB0</th> <th style="text-align: center;">(MSB) 15</th> <th style="text-align: center;">9</th> <th style="text-align: center;">4</th> <th style="text-align: center;">0 (LSB)</th> </tr> </thead> <tbody> <tr> <td style="border: 1px solid black; padding: 2px;">B#9</td> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">B#16#D4</td> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td> </tr> </tbody> </table>		VB0	(MSB) 15	9	4	0 (LSB)	B#9	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	B#16#D4	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
VB0	(MSB) 15	9	4	0 (LSB)																																				
B#9	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0																									
B#16#D4	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0																								

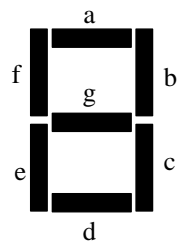
6.7.16 SEG (7-segment Display)

➤ Description

	Name	Usage	Group	
LD	SEG			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	SEG	SEG IN, OUT	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE	I, Q, M, V, L, SM, Constant
<i>OUT</i>	Output	BYTE	Q, M, V, L, SM

This instruction generates a bit pattern of a 7-segment display according to the value represented by the least significant “nibble” (4 bits) of the input byte *IN*, and then put the result into the *OUT*.

<i>IN</i> (LSD)	Display	<i>OUT</i> (- g f e d c b a)		<i>IN</i> (LSD)	Display	<i>OUT</i> (- g f e d c b a)
0	0	0 0 1 1 1 1 1 1		8	8	0 1 1 1 1 1 1 1
1	1	0 0 0 0 0 1 1 0		9	9	0 1 1 0 0 1 1 1
2	2	0 1 0 1 1 0 1 1		A	A	0 1 1 1 0 1 1 1
3	3	0 1 0 0 1 1 1 1		B	B	0 1 1 1 1 1 0 0
4	4	0 1 1 0 0 1 1 0		C	C	0 0 1 1 1 0 0 1
5	5	0 1 1 0 1 1 0 1		D	D	0 1 0 1 1 1 1 0
6	6	0 1 1 1 1 1 0 1		E	E	0 1 1 1 1 0 0 1
7	7	0 0 0 0 0 1 1 1		F	F	0 1 1 1 0 0 0 1

■ **LD**

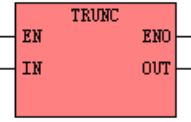
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.7.17 TRUNC (Truncate)

➤ Description

	Name	Usage	Group	<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
LD	TRUNC			
IL	TRUNC	TRUNC <i>IN, OUT</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	REAL	V, L, Constant
<i>OUT</i>	Output	DINT	M, V, L, SM

This instruction converts the REAL value *IN* to a DINT value and assigns the result to the *OUT*. The decimal part of *IN* is truncated off.

■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

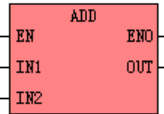
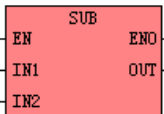
➤ Examples

<p>LD</p>		<p>SM0.0 is always 1, so TRUNC is always executed: truncates off the fraction of VR100, then converts the result to a DINT value and assigns it to VD0.</p>						
<p>IL</p>	<pre>LD %SM0.0 TRUNC %VR100,%VD0</pre>							
<p>Result</p>	<p>The result is as the following:</p> <table border="1" data-bbox="299 847 751 1038"> <thead> <tr> <th>VR100</th> <th>VD0</th> </tr> </thead> <tbody> <tr> <td>123.4</td> <td>DI#123</td> </tr> <tr> <td>5213.6</td> <td>DI#5213</td> </tr> </tbody> </table>		VR100	VD0	123.4	DI#123	5213.6	DI#5213
VR100	VD0							
123.4	DI#123							
5213.6	DI#5213							

6.8 Numeric Instructions

6.8.1 ADD and SUB

➤ Description

	Name	Usage	Group	
LD	ADD			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	SUB			
IL	ADD	ADD IN1, OUT	U	
	SUB	SUB IN1, OUT		

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	INT, DINT, REAL	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant, Pointer
<i>IN2</i>	Input	INT, DINT, REAL	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant, Pointer
<i>OUT</i>	Output	INT, DINT, REAL	Q, AQ, M, V, L, SM, Pointer

■ **LD**

The *IN1*, *IN2* and *OUT* must be of the same data type.

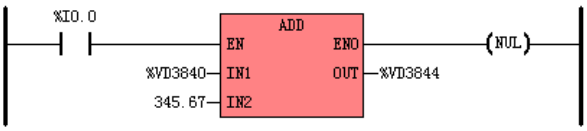
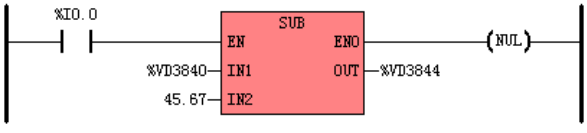
If *EN* is 1, the role that the *ADD* instruction plays is: $OUT=IN1+IN2$, and the role that the *SUB* instruction plays is: $OUT=IN1-IN2$.

■ **IL**

The *INI* and *OUT* must be of the same data type.

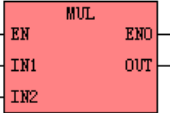
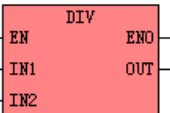
If CR is 1, the role that the *ADD* instruction plays is: $OUT=OUT+INI$, and the role that the *SUB* instruction plays is: $OUT=OUT - INI$. The *ADD* and *SUB* instructions won't influence CR.

➤ Examples

LD		<p>If I0.0 is 0: <i>ADD</i> isn't executed.</p> <p>If I0.0 is 1: The instruction adds VD3840 and 345.67, and assigns the result to VD3844.</p>
		<p>If I0.0 is 0: <i>SUB</i> isn't executed.</p> <p>If I0.0 is 1: The instruction subtracts 45.67 from VD3840, and assigns the result to VD3844.</p>
IL	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>ADD 345.67, %VD3840 (* If CR is 1: VD3840 = VD3840 +245.67 *)</p> <p style="text-align: right;"> (* If CR is 0: the instruction isn't executed *)</p>	
	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>SUB 45.67, %VD3840 (* If CR is 1: VD3840 = VD3840 - 45.67 *)</p> <p style="text-align: right;"> (* If CR is 0: the instruction isn't executed *)</p>	

6.8.2 MUL and DIV

➤ Description

	Name	Usage	Group	
LD	MUL			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	DIV			
IL	MUL	MUL IN1, OUT	U	
	DIV	DIV IN1, OUT		

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	INT, DINT, REAL	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant, Pointer
<i>IN2</i>	Input	INT, DINT, REAL	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant, Pointer
<i>OUT</i>	Output	INT, DINT, REAL	Q, AQ, M, V, L, SM, Pointer

■ LD

The *IN1*, *IN2* and *OUT* must be of the same data type.

If *EN* is 1, the role that the *MUL* instruction plays is: $OUT = IN1 \times IN2$, and the role that the *DIV* instruction plays is: $OUT = IN1 \div IN2$.

■ IL

The *IN1* and *OUT* must be of the same data type.

If *CR* is 1, the role that the *MUL* instruction plays is: $OUT = OUT \times IN1$, and the role that the *DIV* instruction

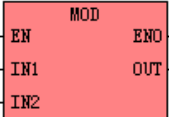
plays is: $OUT = OUT \div IN1$. The *MUL* and *DIV* instructions won't influence CR.

➤ Examples

LD		<p>If I0.0 is 0: <i>MUL</i> isn't executed.</p> <p>If I0.0 is 1: The instruction multiplies AIW0 and VW0, and assigns the result to AQW0.</p>
		<p>If I0.0 is 0: <i>DIV</i> isn't executed.</p> <p>If I0.0 is 1: The instruction divides AIW2 by VW0, and assigns the result to VW2.</p>
IL	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>MUL %AIW0, %VW0 (* If CR is 1: $VW0 = VW0 \times AIW0$ *)</p> <p> (* If CR is 0: the instruction isn't executed *)</p>	
	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>DIV %AIW2, %VW0 (* If CR is 1: $VW0 = VW0 \div AIW2$ *)</p> <p> (* If CR is 0: the instruction isn't executed *)</p>	

6.8.3 MOD (Modulo-Division)

➤ Description

	Name	Usage	Group	
LD	MOD			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	MOD	MOD <i>IN1, OUT</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN1</i>	Input	BYTE, INT, DINT	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant, Pointer
<i>IN2</i>	Input	BYTE, INT, DINT	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant, Pointer
<i>OUT</i>	Output	BYTE, INT, DINT	Q, AQ, M, V, L, SM, Pointer

■ **LD**

The *IN1*, *IN2* and *OUT* must be of the same data type.

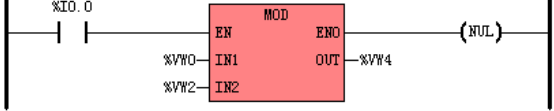
If *EN* is 1, this instruction divides *IN1* by *IN2*, and assigns the remainder to *OUT*.

■ **IL**

The *IN1* and *OUT* must be of the same data type.

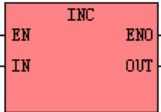
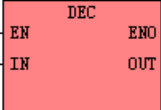
If *CR* is 1, this instruction divides *OUT* by *IN1*, and assigns the remainder to *OUT*. It does not influence *CR*.

➤ Examples

<p>LD</p>		<p>If I0.0 is 0: <i>MOD</i> is not executed. If I0.0 is 1: VW0 is divided by VW2, and the remainder is assigned to VW4.</p>												
<p>IL</p>	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>MOD %VW0, %VW4 (* If CR is 1: VW4 is divided by VW0, and the remainder is still stored in VW4 *)</p> <p>(* If CR is 0: this instruction is not executed *)</p>													
<p>Result</p>	<p>For the LD example, if <i>MOD</i> instruction is executed, the result is shown as the following:</p> <table border="0" data-bbox="312 956 820 1142"> <tr> <td>Address</td> <td>VW0</td> <td>VW2</td> </tr> <tr> <td>Value</td> <td><input type="text" value="8"/></td> <td><input type="text" value="3"/></td> </tr> <tr> <td>Address</td> <td>VW4</td> <td></td> </tr> <tr> <td>Value</td> <td><input type="text" value="2"/></td> <td></td> </tr> </table>		Address	VW0	VW2	Value	<input type="text" value="8"/>	<input type="text" value="3"/>	Address	VW4		Value	<input type="text" value="2"/>	
Address	VW0	VW2												
Value	<input type="text" value="8"/>	<input type="text" value="3"/>												
Address	VW4													
Value	<input type="text" value="2"/>													

6.8.4 INC and DEC

➤ Description

	Name	Usage	Group	
LD	INC			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	DEC			
IL	INC	INC <i>OUT</i>	U	
	DEC	DEC <i>OUT</i>		

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE, INT, DINT	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant, Pointer
<i>OUT</i>	Output	BYTE, INT, DINT	Q, AQ, M, V, L, SM, Pointer

■ LD

The *IN* and *OUT* must be of the same data type.

If *EN* is 1, the role that the *INC* instruction plays is: $OUT = IN + 1$, and the role that the *DEC* instruction plays: $OUT = IN - 1$.

■ IL

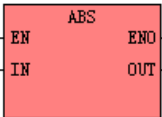
If *CR* is 1, the role that the *INC* instruction plays is: $OUT = OUT + 1$, and the role that the *DEC* instruction plays: $OUT = OUT - 1$. They do not influence *CR*.

➤ Examples

LD		<p>If I0.0 is 0: <i>INC</i> isn't executed. If I0.0 is 1: $VD4 = VD0 + DI\#1$.</p>
		<p>If I0.0 is 0: <i>DEC</i> isn't executed. If I0.0 is 1: $VB2 = VB0 - B\#1$.</p>
IL	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>INC %VD4 (* If CR is 1: $VD4 = VD4 + DI\#1$ *)</p> <p> (* If CR is 0: this instruction isn't executed *)</p>	
	<p>LD %I0.0 (* CR is created with I0.0 *)</p> <p>DEC %VB2 (* If CR is 1: $VB2 = VB2 - B\#1$ *)</p> <p> (* If CR is 0: this instruction isn't executed *)</p>	

6.8.5 ABS (Absolute Value)

➤ Description

	Name	Usage	Group	
LD	ABS			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	ABS	ABS IN, OUT	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	INT, DINT, REAL	I, Q, V, M, L, SM, T, C, AI, AQ, HC, Constant, Pointer
<i>OUT</i>	Output	INT, DINT, REAL	Q, V, M, L, SM, AQ, Pointer

The *IN* and *OUT* must be of the same data type.

This instruction calculates the absolute value of the input *IN*, and assigns the result to *OUT*, as shown in the following formula: $OUT = |IN|$.

■ **LD**

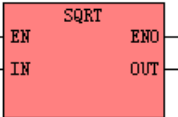
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.8.6 SQRT (Square Root)

➤ Description

	Name	Usage	Group	
LD	SQRT			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	SQRT	SQRT <i>IN, OUT</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	REAL	V, L, Constant, Pointer
<i>OUT</i>	Output	REAL	V, L, Pointer

This instruction calculates the square root of the input *IN*, and assigns the result to *OUT*, as shown in the following formula: $OUT = \sqrt{IN}$.

■ **LD**

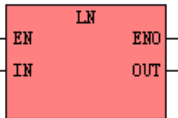
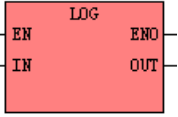
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.8.7 LN (Natural Logarithm), LOG (Common Logarithm)

➤ Description

	Name	Usage	Group	
LD	LN			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	LOG			
IL	LN	LN IN, OUT	U	
	LOG	LOG IN, OUT		

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	REAL	V, L, Constant, Pointer
OUT	Output	REAL	V, L, Pointer

The LN instruction calculates the natural logarithm of the input *IN*, and assigns the result to *OUT*, as shown in the following formula: $OUT = \log_e(IN)$.

The LOG instruction calculates the common logarithm of the input *IN*, and assigns the result to *OUT*, as shown in the following formula: $OUT = \log_{10}(IN)$.

■ **LD**

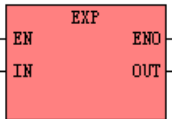
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.8.8 EXP (Exponent with the base e)

➤ Description

	Name	Usage	Group	
LD	EXP			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	EXP	EXP IN, OUT	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	REAL	V, L, Constant, Pointer
<i>OUT</i>	Output	REAL	V, L, Pointer

This instruction calculates the exponent with the base e of the input *IN*, and assigns the result to *OUT*, as shown in the following formula: $OUT = e^{IN}$.

■ **LD**

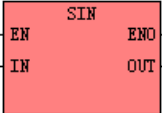
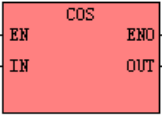
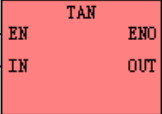
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.8.9 SIN (sine), COS (cosine), TAN (tangent)

➤ Description

	Name	Usage	Group	
LD	SIN			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	COS			
	TAN			
IL	SIN	SIN IN, OUT	U	
	COS	COS IN, OUT		
	TAN	TAN IN, OUT		

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	REAL	V, L, Constant, Pointer
<i>OUT</i>	Output	REAL	V, L, Pointer

The SIN instruction calculates the sine value of the input *IN*, and assigns the result to *OUT*, as shown in the following formula: $OUT = SIN(IN)$.

The COS instruction calculates the cosine value of the input *IN*, and assigns the result to *OUT*, as shown in the following formula: $OUT = COS(IN)$.

The TAN instruction calculates the tangent value of the input *IN*, and assigns the result to *OUT*, as shown in the

following formula: $OUT = TAN(IN)$.

- **LD**

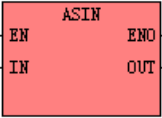
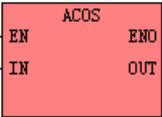
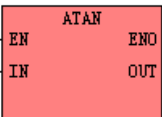
If EN is 1, this instruction is executed.

- **IL**

If CR is 1, this instruction is executed, and it does not influence CR .

6.8.10 ASIN (arc-sine), ACOS (arc-cosine), ATAN (arc-tangent)

➤ Description

	Name	Usage	Group
LD	ASIN		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	ACOS		
	ATAN		
IL	ASIN	ASIN IN, OUT	U
	ACOS	ACOS IN, OUT	
	ATAN	ATAN IN, OUT	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	REAL	V, L, Constant, Pointer
<i>OUT</i>	Output	REAL	V, L, Pointer

The ASIN instruction calculates the arc-sine value of the input *IN*, and assigns the result to *OUT*, as shown in the following formula: $OUT = \text{ARCSIN}(IN)$.

The ACOS instruction calculates the arc-cosine value of the input *IN*, and assigns the result to *OUT*, as shown in the following formula: $OUT = \text{ARCCOS}(IN)$.

The ATAN instruction calculates the arc-tangent value of the input *IN*, and assigns the result to *OUT*, as shown in the following formula: $OUT = \text{ARCTAN}(IN)$.

■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.9 Program Control

In IL, jump instructions and return instructions do not influence CR, so CR shall remain unchanged just after a jump or return instruction is executed, and you need pay more attention when using them.

6.9.1 LBL and JMP Instructions

➤ Description

	Name	Usage	Group	
LD	LBL	lbl (LBL)		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506
	JMP	lbl (JMP)		
	JMPC	lbl (JMPC)		
	JMPCN	lbl (JMPCN)		
IL	标号	lbl:		<input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	JMP	JMP <i>lbl</i>	U	
	JMPC	JMPC <i>lbl</i>		
	JMPCN	JMPCN <i>lbl</i>		

Operand	Description
<i>lbl</i>	Valid identifier

■ LD

The *LBL* instruction is used to define a label at the current position, and the label will function as the destination for the jump instructions. Redefinition of a label identifier is forbidden. This instruction is executed unconditionally, so you need not add any elements on its left. Actually, KincoBuilder will ignore all the elements on its left.

The *JMP* instruction is used to unconditionally transfer program execution **错误!未找到引用源。** to the network label **错误!未找到引用源。** **错误!未找到引用源。** specified by *lbl*.

The *JMPC* instruction is used to transfer program execution **错误!未找到引用源。** to the network label **错误!未找到引用源。** **错误!未找到引用源。** specified by *lbl* when the horizontal link state on its left is true.

The *JMPCN* instruction is used to transfer program execution **错误!未找到引用源。** to the network label **错误!未找到引用源。** **错误!未找到引用源。** specified by *lbl* when the horizontal link state on its left is false.

The jump instruction and its destination label must always exist within the same POU.

■ **IL**

The definition format of a label is *a legal identifier*:. The definition occupies an independent line. Redefinition of a label identifier is forbidden.

The *JMP* instruction is used to unconditionally transfer program execution **错误!未找到引用源。** to the label **错误!未找到引用源。** **错误!未找到引用源。** specified by *lbl*.

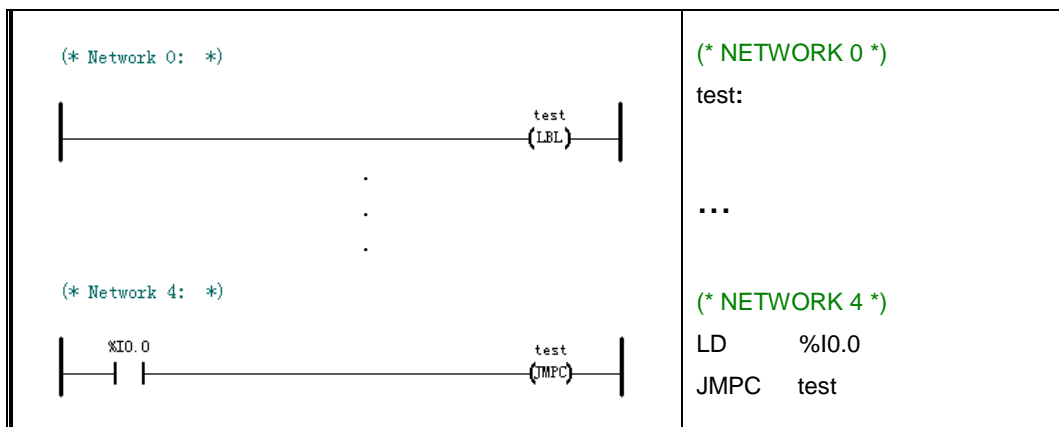
The *JMPC* instruction is used to transfer program execution **错误!未找到引用源。** to the label **错误!未找到引用源。** **错误!未找到引用源。** specified by *lbl* when CR is 1.

The *JMPCN* instruction is used to transfer program execution **错误!未找到引用源。** to the label **错误!未找到引用源。** **错误!未找到引用源。** specified by *lbl* when CR is 0.

The jump instruction and its destination label must always exist within the same POU.

➤ Examples

LD	IL
-----------	-----------



6.9.2 Return Instructions

Notice: Return instructions can only be used in subroutines and interrupt routines.

➤ Description

	Name	Usage	Group	
LD	RETC	RETC		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506
	RETCN	RETCN		
IL	RETC	RETC	U	<input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	RETCN	RETCN		

■ LD

The *RETC* instruction is used to terminate a subroutine or an interrupt routine and transfer program execution back to the calling 错误!未找到引用源。错误!未找到引用源。错误!未找到引用源。 entry when the horizontal link state on its left is true.

The *RETCN* instruction is used to terminate a subroutine or an interrupt routine and transfer program execution back to the calling 错误!未找到引用源。错误!未找到引用源。错误!未找到引用源。 entry when the horizontal

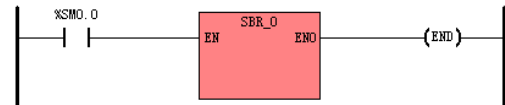
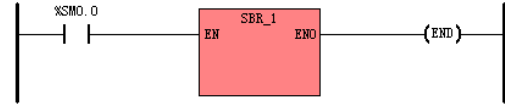

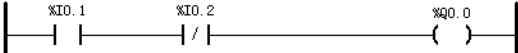
link state on its left is false.

■ **IL**

The *RETC* instruction is used to terminate a subroutine or an interrupt routine and transfer program execution back to the calling 错误!未找到引用源。错误!未找到引用源。错误!未找到引用源。 entry when CR is 1.

The *RETCN* instruction is used to terminate a subroutine or an interrupt routine and transfer program execution back to the calling 错误!未找到引用源。错误!未找到引用源。错误!未找到引用源。 entry when CR is 0.

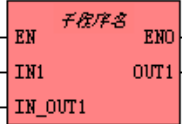
➤ Examples

LD	<p>Main program:</p> <pre>(* Network 0: *)</pre>  <pre>(* Network 1: *)</pre>  <p>Subroutine SBR_0:</p> <pre>(* Network 0: *)</pre>  <pre>(* Network 1: *)</pre> 	<p>For SBR_0:</p> <p>If IO.0 is 0, the instructions are executed sequentially.</p> <p>If IO.0 is 1, program execution is transferred back to the calling entry in the main program, and the KINCO-K5 continues to execute the instructions in Network 1.</p>
-----------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

IL	Main Program:	
	LD	%SM0.0 (* CR is created with SM0.0 *)
	CAL	SBR_0 (* Call SBR_0 *)
	CAL	SBR_1 (* Call SBR_1 *)
	SBR_0:	
	LD	%I0.0 (* CR is created I0.0 *)
	RET	(* If CR is 1, SBR_0 shall be terminate and program execution is transferred *) (* back to the calling 错误!未找到引用源。错误!未找到引用源。错误!未找到引用源。 entry in the main program. *)
	LD	%I0.1 (* If RETC is not executed, the subsequent instructions are to be executed *)
	ANDN	%I0.2
	ST	%Q0.0

6.9.3 CAL (Call a subroutine)

➤ Description

	Name	Usage	Group	
LD	CAL			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506
IL	CAL	CAL 子程序名, 子程序实参 1, 子程序实参 2, ...	U	<input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508

This instruction is used for calling and executing a subroutine with the specified *NAME*. The subroutine to be called must exist in the user program already.

You can use a CAL instruction with or without parameters. If a CAL instruction is used with parameters, the data type and the variable type of the actual parameters, must match those of the formal parameters which are defined in the Local Variable Table of the called subroutine. Also, the order of the actual parameters must be the same as

that of the the formal parameters.

➤ **LD**

All the names of the subroutines appear in the group [**SBR**] of the [**LD instructions**] tree. Double click on a name, then the corresponding subroutine is added into you program. If *EN* is 1, this subroutine is executed.

➤ **IL**

If *CR* is 1, the subroutine will be called and executed.

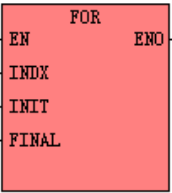
The *CAL* instruction does not influence *CR*, but *CR* may be changed in the subroutine.

➤ Examples

LD	<p>Main program:</p> <p>(* Network 0 *) (* call the subroutine 'Initialize' *)</p>																									
	<p>The Local Variable Table of the subroutine 'Initialize':</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Address</th> <th>Symbol</th> <th>Var Type</th> <th>Data Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>%LD.0</td> <td>IN1</td> <td>VAR_INPUT</td> <td>BOOL</td> <td></td> </tr> <tr> <td>%LB16</td> <td>IN2</td> <td>VAR_INPUT</td> <td>BYTE</td> <td></td> </tr> <tr> <td>%LW22</td> <td>IN_OUT1</td> <td>VAR_IN_OUT</td> <td>INT</td> <td></td> </tr> <tr> <td>▶ %LD18</td> <td>OUT1</td> <td>VAR_OUTPUT</td> <td>REAL</td> <td></td> </tr> </tbody> </table>	Address	Symbol	Var Type	Data Type	Comment	%LD.0	IN1	VAR_INPUT	BOOL		%LB16	IN2	VAR_INPUT	BYTE		%LW22	IN_OUT1	VAR_IN_OUT	INT		▶ %LD18	OUT1	VAR_OUTPUT	REAL	
Address	Symbol	Var Type	Data Type	Comment																						
%LD.0	IN1	VAR_INPUT	BOOL																							
%LB16	IN2	VAR_INPUT	BYTE																							
%LW22	IN_OUT1	VAR_IN_OUT	INT																							
▶ %LD18	OUT1	VAR_OUTPUT	REAL																							
IL	<p>Main Program:</p> <p>(* Network 0 *) (*call the subroutine 'Initialize'*)</p> <p>LD %I0.0</p> <p>CAL Initialize, %M0.0, %VBO, %VW2, %VR10</p>																									
	<p>The Local Variable Table of the subroutine 'Initialize':</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Address</th> <th>Symbol</th> <th>Var Type</th> <th>Data Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>%LD.0</td> <td>IN1</td> <td>VAR_INPUT</td> <td>BOOL</td> <td></td> </tr> <tr> <td>%LB16</td> <td>IN2</td> <td>VAR_INPUT</td> <td>BYTE</td> <td></td> </tr> <tr> <td>%LW22</td> <td>IN_OUT1</td> <td>VAR_IN_OUT</td> <td>INT</td> <td></td> </tr> <tr> <td>▶ %LD18</td> <td>OUT1</td> <td>VAR_OUTPUT</td> <td>REAL</td> <td></td> </tr> </tbody> </table>	Address	Symbol	Var Type	Data Type	Comment	%LD.0	IN1	VAR_INPUT	BOOL		%LB16	IN2	VAR_INPUT	BYTE		%LW22	IN_OUT1	VAR_IN_OUT	INT		▶ %LD18	OUT1	VAR_OUTPUT	REAL	
Address	Symbol	Var Type	Data Type	Comment																						
%LD.0	IN1	VAR_INPUT	BOOL																							
%LB16	IN2	VAR_INPUT	BYTE																							
%LW22	IN_OUT1	VAR_IN_OUT	INT																							
▶ %LD18	OUT1	VAR_OUTPUT	REAL																							

6.9.4 FOR/NEXT (FOR/NEXT Loop)

➤ Description

	Name	Usage	Group	
LD	FOR			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	NEXT	(NEXT)		
IL	FOR	FOR INDX, INIT, FINAL	U	
	NEXT	NEXT		

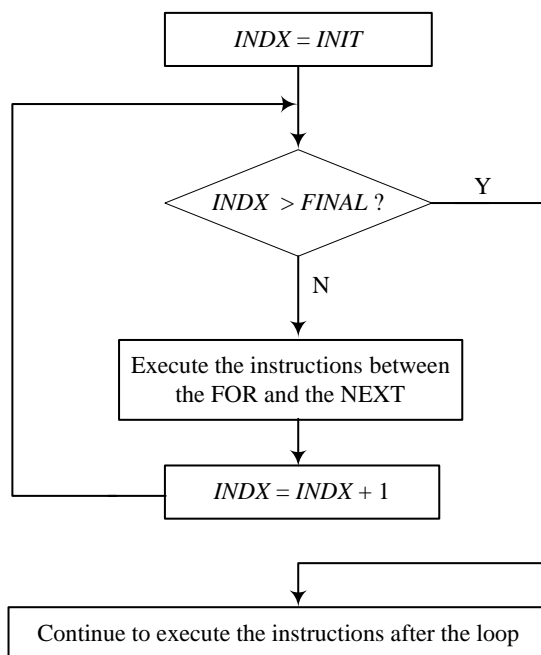
Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>INDX</i>	Input	INT	M, V, L, SM
<i>INIT</i>	Input	INT	M, V, L, SM, T, C, Constant
<i>FINAL</i>	Output	INT	M, V, L, SM, T, C, Constant

The FOR/NEXT instructions express a loop that is repeated for the specified count. You specify the loop count (*INDX*), the starting value (*INIT*), and the ending value (*FINAL*).

The NEXT instruction marks the end of the loop, and the FOR instruction executes the instructions between the FOR and the NEXT. They must be used in pairs, each FOR instruction requires a NEXT instruction.

If a FOR/NEXT loop exists within another FOR/NEXT loop, it is called a nested loop. You can nest FOR/NEXT loops to a depth of eight.

The execution process of the FOR/NEXT loop is shown in the following figure:



When using the FOR/NEXT instructions, you need to notice the following details:

- The FOR instruction must be the 2nd instruction within a Network.
- The NEXT instruction must monopolize a Network.
- You can change the final value from within the loop itself to change the end condition of the loop.
- A loop, which needs to execute for a long time that exceed the CPU's watchdog time, can leads to the CPU restarting.

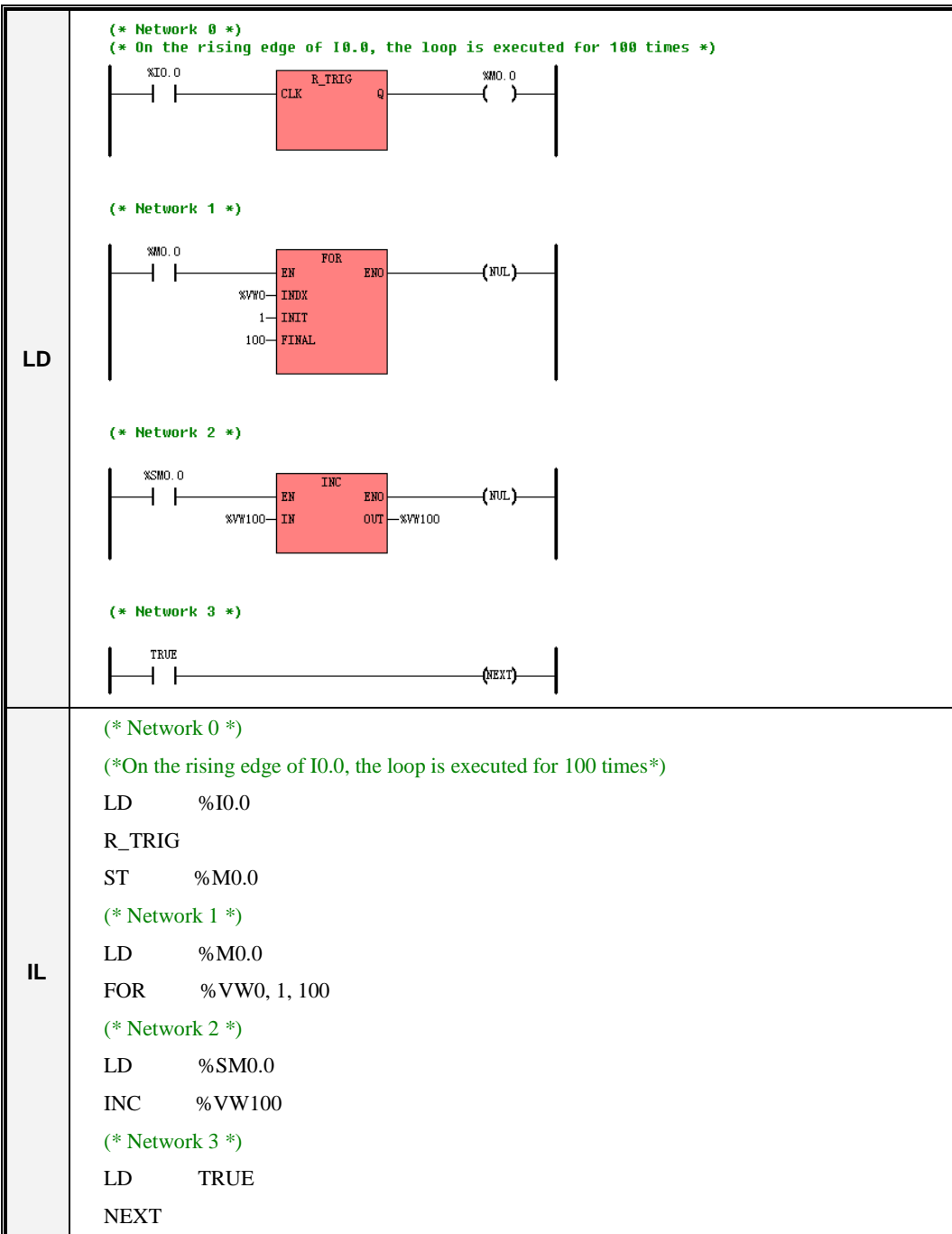
■ **LD**

If EN is 1, this instruction is executed.

■ **IL**

If CR is 1, this instruction is executed, and it does not influence CR .

➤ **Example**



6.9.5 END (Terminate the scan cycle)

➤ Description

	Name	Usage	Group	
LD	END	—(END)—		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506
IL	END	END	U	<input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508

This instruction can only be used in the main program, for terminating the current scan cycle.

At the end of the main program, KincoBuilder automatically calls the END instruction implicitly.

➤ **LD**

If the horizontal link state on its left is 1, this instruction is executed. Otherwise, this instruction does not take effect.

➤ **IL**

If CR is 1, this instruction will be executed. Otherwise, this instruction does not take effect.

This instruction does not influence CR.

6.9.6 STOP (Stop the CPU)

➤ Description

	Name	Usage	Group	
LD	STOP	STOP		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506
IL	STOP	STOP	U	<input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508

This instruction terminates the execution of your program and turns the CPU from RUN into STOP mode immediately.

➤ **LD**

If the horizontal link state on its left is 1, this instruction is executed. Otherwise, this instruction does not take effect.

➤ **IL**

If CR is 1, this instruction is executed. Otherwise, this instruction does not take effect.

This instruction does not influence CR.

6.9.7 WDR (Watchdog Reset)

➤ Description

	Name	Usage	Group	
LD	WDR	—(WDR)—		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506
IL	WDR	WDR	U	<input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508

This instruction re-triggers the system watchdog timer of the CPU.

Using the WDR instruction can increase the time that the scan cycle is allowed to take without leading to a watchdog error, so the program that needs longer time can be executed successfully. But you should use this instruction carefully, because the following processes are inhibited until the scan cycle is completed:

- CPU self-diagnosis
- Read the inputs (sample all the physical input channels and writes these values to the input image areas)
- Communication
- Write to the outputs (write the values stored in the output image areas to the physical output channels)
- Timing for the 10-ms and 100-ms timers

➤ **LD**

If the horizontal link state on its left is 1, this instruction is executed. Otherwise, this instruction does not take effect.

➤ **IL**

If CR is 1, this instruction is executed. Otherwise, this instruction does not take effect.

This instruction does not influence CR.

6.10 Interrupt Instructions

The purpose of using interrupt technique is to increase the execution efficiency of the Kinco-K5 to quickly respond to special internal or external predefined events. The Kinco-K5 supports tens of events each of which is assigned with a unique event number.

6.10.1 How the Kinco-K5 handles Interrupt Routines

An interrupt routine is executed once only on each occurrence of the interrupt event associated with it. Once the last instruction of the interrupt routine has been executed, program execution is transferred back to the main program. You can exit the routine by executing a *RETC* or *RETCN* instruction.

Interrupt technique makes the Kinco-K5 respond to special events quickly, so you should optimize interrupt routines to be short and efficient.

6.10.2 Interrupt Priority and Queue

Different events are on different priority levels. When interrupt events occur, they will queue up according to their priority levels and time sequence: the interrupt events in the same priority group are handled following the principle of “first come, first served”; the events in the higher priority group are handled preferentially. Only one interrupt routine can be executed at one point in time. Once an interrupt routine begins to be executed, it cannot be interrupted by another interrupt routine. Interrupt events that occur while another interrupt routine is being executed are queued up for later handling.

6.10.3 Types of Interrupt Events Supported by the Kinco-K5

The Kinco-K5 supports the following types of interrupt events:

- Communication Port Interrupts

This type of interrupts has the highest priority.

They are used for free-protocol communication mode. The Receive and Transmit interrupts facilitate you to fully control the communication. Please refer to the Transmit and Receive instructions for detailed information.

➤ I/O Interrupts

This type of interrupts has a medium priority.

These interrupt include rising/falling edge interrupts, HSC interrupts and PTO interrupts.

The rising/falling edge interrupts can only be trapped by the first four DI channels (%I0.0~%I0.3) on the CPU body. Each of them can be used to notify that the signal state has changed and the PLC must respond immediately.

The HSC interrupts occur when the counting value reaches the preset value, the counting direction changes or the counter is reset externally. Each of them allows the PLC respond in real time to high-speed events that cannot be responded immediately at scan speed.

The PTO interrupts occur immediately when outputting the specified number of pulses is completed. A typical application is to control the stepper motor.

➤ Time Interrupts

This type of interrupts has the lowest priority.

These interrupt include timed interrupts and the timer T2 and T3 interrupts.

The timed interrupts occur periodically (unit: ms), and they can be used for periodical tasks.

The timer interrupt occurs immediately when the current value of T2 or T3 reaches the preset value. It can be used to timely respond to the end of a specified time interval.

6.10.4 Interrupt Events List

Event No.	Description	Type
34	PORT 2: XMT complete	
33	PORT 2: RCV complete	
32	PORT 1: XMT complete	Communication Port

31	PORT 1: RCV complete	Interrupts
30	PORT 0: XMT complete	
29	PORT 0: RCV complete	
28~27	Reserved	I/O Interrupts
26	I0.0, Falling edge	
25	I0.0, Rising edge	
24	I0.1, Falling edge	
23	I0.1, Rising edge	
22	I0.2, Falling edge	
21	I0.2, Rising edge	
20	I0.3, Falling edge	
19	I0.3, Rising edge	
18	HSC0 CV=PV	
17	HSC0 direction changed	
16	HSC0 external reset	
15	HSC1 CV=PV	
14	HSC1 direction changed	
13	HSC1 external reset	
12~5	Reserved	Time Interrupts
4	Timed interrupt 1. Its period is specified in SMW16, unit: ms.	
3	Timed interrupt 0. Its period is specified in SMW12, unit: ms.	
2	Timer T3 ET=PT	
1	Timer T2 ET=PT	

Table 6-1 Interrupt Events

6.10.5 ENI (Enable Interrupt), DISI (Disable Interrupt)

➤ Description

	Name	Usage	Group	
				<input checked="" type="checkbox"/> CPU504

LD	ENI	—(ENI)—	U	<input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	DISI	—(DISI)—		
IL	ENI	ENI		
	DISI	DISI		

The ENI instruction globally enables processing all attached interrupt events.

The DISI instruction globally inhibits processing all interrupt events.

When you turn the CPU into RUN mode, interrupts are enabled being processed by default.

➤ **LD**

If the horizontal link state on its left is 1, the instruction is executed. Otherwise, the instruction does not take effect.

➤ **IL**

If CR is 1, the instruction is executed. Otherwise, the instruction does not take effect.

The instruction does not influence CR.

6.10.6 ATCH and DTCH Instructions

➤ Description

	Name	Usage	Group
LD	ATCH		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	DTCH		

IL	ATCH	ATCH INT, EVENT	U	
	DTCH	DTCH EVENT		

Operands	Input/Output	Data Type	Description
<i>INT</i>	Input		The name of an existing interrupt routine
<i>EVENT</i>	Input	INT	Constant, an interrupt event No.

■ **LD**

If *EN* is 1, the *ATCH* instruction attaches an interrupt event (specified by the event number *EVENT*) to the interrupt routine (specified by the routine name *INT*) and enables the interrupt event. After this instruction is executed, the interrupt routine shall be invoked automatically on the occurrence of the interrupt event. You can attach several events to one interrupt routine, but one event can only be attached to one interrupt routine.

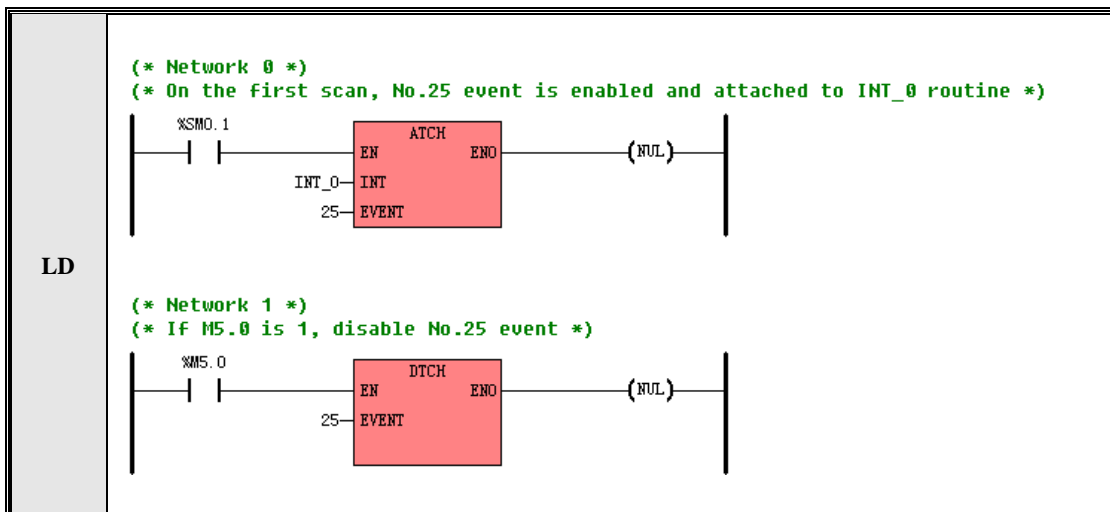
If *EN* is 1, the *DTCH* instruction breaks the attachment between the interrupt event (specified by the event number *EVENT*) and its interrupt routine, and makes the interrupt event return to be disabled.

■ **IL**

If *CR* is 1, the *ATCH* instruction attaches an interrupt event (specified by the event number *EVENT*) to the interrupt routine (specified by the routine name *INT*) and enables the interrupt event. This instruction does not influence *CR*.

If *CR* is 1, the *DTCH* instruction breaks the attachment between the interrupt event (specified by the event number *EVENT*) and its interrupt routine, and makes the interrupt event return to be disabled. This instruction does not influence *CR*.

➤ Examples



IL	(* NETWORK 0 *)		
	LD	%SM0.1	
	ATCH	INT_0, 25	(*On the first scan, No.25 event is enabled and attached to INT_0 routine *)
	LD	%M5.0	(* CR is created with M5.0 *)
	DTCH	25	(*If CR is 1, disable No.25 event *)

6.11 Clock Instructions

A real-time clock (RTC) is built in the CPU module for real-time clock/calendar indication. The real-time clock/calendar adopts BCD-format coding through second to year, automatically executes leap-year adjustment and uses the super capacitor as backup. At normal temperature, the duration of the super capacitor is 72 hours.

6.11.1 Adjusting the RTC online

You should adjust the RTC to the current actual time and date before using it. Before adjustment, the value of the RTC may be random.

Execute the [PLC]>[Time of Day Clock...] menu command to open the “Time of Day Clock...” dialog to adjust the RTC online, as shown in the following figure.

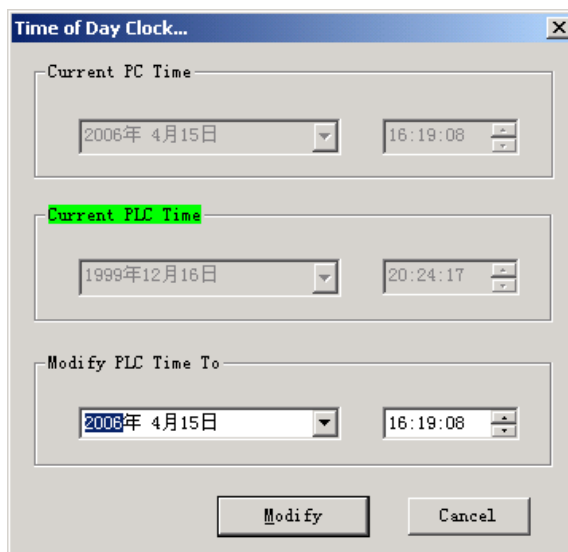


Figure 6-1 Adjusting the RTC

- **Current PC Time:** Indicate the current date and time of the current PC.
- **Current PLC Time:** Indicate the current date and time of the RTC of the online CPU module. Its background being green indicates that the CPU module communicates with the PC successfully, and its

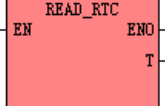
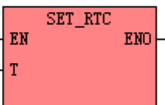
background being yellow indicates the CPU module fails to communicate with the PC.

- **Modify PLC Time To:** You can enter the desired date and time for the RTC here. Enter them through keyboard, or click the arrowhead at the right end of the relevant box to select the date or adjust the time.
- **Adopt Summer Time:** You may click this item when needed
- **Modify:** Click this button, the date and time you have entered shall be written into the CPU module, and then the RTC shall be adjusted to the desired date and time.

NOTE: If Summer Time is adopted it is required to reboot Kincobuilder to take effect.

6.11.2 READ_RTC and SET_RTC

- Description

	Name	Usage	Group	
LD	READ_RTC			<input type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	SET_RTC			
IL	READ_RTC	READ_RTC <i>T</i>	U	
	SET_RTC	SET_RTC <i>T</i>		

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>T</i>	Input (SET_RTC)	BYTE	V
	Output (READ_RTC)		

The *READ_RTC* instruction is used to read the current date and time from the RTC and write them to an 8-byte time buffer beginning with address *T*.

The *SET_RTC* instruction is used to write the date and time specified by the 8-byte time buffer beginning with address *T* to the RTC.

The storage format of the date and time in the time buffer is shown in the following table.

Note: All the values are of BCD coding.

V Byte	Meaning	Remark
T	Week	Range: 1~7, thereof 1 represents Monday, 7 represents Sunday.
T+1	Second	Range: 0~59
T+2	Minute	Range: 0~59
T+3	Hour	Range: 0~23
T+4	Day	Range: 1~31
T+5	Month	Range: 1~12
T+6	Year	Range: 0~99
T+7	Century	Fixed as 20

Table 6-2 The Time Buffer



Notice:

- You are recommended to adjust the RTC correctly using [PLC]>[Time of Day Clock...] menu command before using it.
- Because the CPU module won't check the validity of the date and time you have entered and invalid data (e.g. Feb 30) will be accepted. Therefore, you have to ensure the validity of the date/time you have entered.

■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**


If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

LD	<p>(* Network 0 *) (* Read the RTC every 1 second *)</p> <p>(* Network 1 *) (* Turn on Q0.0 during 9:00-18:00 everyday, and turn off it at other time. *)</p>
IL	<p>(* Network 0 *) (*Read the RTC every 1 second*)</p> <pre>LD %SM0.3 R_TRIG READ_RTC %VB0</pre> <p>(* Network 1 *) (*Turn on Q0.0 during 9:00-18:00 everyday, and turn off it at other time.*)</p> <pre>LD %SM0.0 GE %VB3, B#16#9 LT %VB3, B#16#18 ST %Q0.0</pre>

6.11.3 RTC_R

➤ Description

	Name	Usage	Group	
LD	RTC_R			<input type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	RTC_R	<i>RTC_R FMT, WEEK, SECOND, MINUTE, HOUR, DAY, MONTH, YEAR, CENTURY</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
FMT	Input	BYTE	L, M, V, Constants
WRRK	Output	BYTE	L, M, V
SECOND	Output	BYTE	L, M, V
MINUTE	Output	BYTE	L, M, V
HOUR	Output	BYTE	L, M, V
DAY	Output	BYTE	L, M, V
MONTH	Output	BYTE	L, M, V
YEAR	Output	BYTE	L, M, V
CENTURY	Output	BYTE	L, M, V

You may refer to the table below to see the parameters:

Operands	Description
----------	-------------

EN	Enable
FMT	Output Format. 0 represents decimal and 1 represents BCD code
WRRK	Week. 1-7 represents Mon to Sun
SECOND	Second, ranging from 0-59
MINUTE	Minute, ranging from 0-59
HOUR	Hour, ranging from 0-23
DAY	Day, ranging from 1-31
MONTH	Month, ranging from 1-12
YEAR	Year, ranging from 0-99
CENTURY	Century, a fixed value of 20

RTC_R can be used to load current time and date from real-time clock and will save each input parameter accordingly.

FMT represents the format of each parameter; 0 represents decimal and 1 represents BCD code.

➤ LD

If EN value is 1, then RTC_R command will be executed and vice versa.

➤ IL

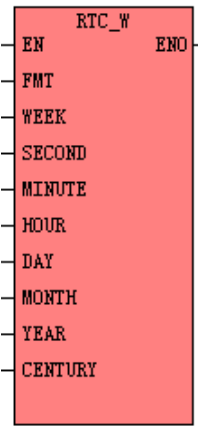
If CR value is 1, then RTC_R command will be executed and vice versa.

The RTC_R command will not affect CR value.

6.11.4 RTC_W

➤ Description

	Name	Usage	Group	<input type="checkbox"/> CPU504

LD	RTC_W		<input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	RTC_W	<i>RTC_W FMT, WEEK, SECOND, MINUTE, HOUR, DAY, MONTH, YEAR, CENTURY</i>	U

Operands	Input/Output	Data Type	Acceptable Memory Areas
FMT	Input	BYTE	L, M, V, Constants
WRRK	Input	BYTE	L, M, V, Constants
SECOND	Input	BYTE	L, M, V, Constants
MINUTE	Input	BYTE	L, M, V, Constants
HOUR	Input	BYTE	L, M, V, Constants
DAY	Input	BYTE	L, M, V, Constants
MONTH	Input	BYTE	L, M, V, Constants
YEAR	Input	BYTE	L, M, V, Constants
CENTURY	Input	BYTE	L, M, V, Constants

You may refer to the table below to see the parameters:

Operands	Description
EN	Enable

FMT	Output Format. 0 represents decimal and 1 represents BCD code
WRRK	Week. 1-7 represents Mon to Sun
SECOND	Second, ranging from 0-59
MINUTE	Minute, ranging from 0-59
HOUR	Hour, ranging from 0-23
DAY	Day, ranging from 1-31
MONTH	Month, ranging from 1-12
YEAR	Year, ranging from 0-99
CENTURY	Century, a fixed value of 20

RTC_R can be used to load current time and date from real-time clock and will save each input parameter accordingly.

FMT represents the format of each parameter; 0 represents decimal and 1 represents BCD code.

➤ LD

If EN value is 1, then RTC_R command will be executed and vice versa.

➤ IL

If CR value is 1, then RTC_R command will be executed and vice versa.

The RTC_R command will not affect CR value.

6.12 Communication Instructions

6.12.1 Free-protocol Communication

These instructions are used for free-protocol communication. Free-protocol communication mode allows your program to entirely control the communication ports of the CPU. You can use free-protocol communication mode to implement user-defined communication protocols to communicate with all kinds of intelligent devices. ASCII and binary protocols are both supported.

The CPU module is integrated with 1, 2 or 3 communication ports, each of that serves as a default Modbus RTU slave. After the communication instructions are executed, free-protocol communication mode shall be activated, involving no manual operation.

You can configure the communication parameters (such as Baudrate, Parity, etc) of each port in the Hardware Window. Please refer to [4.3.4.1 Parameters of the CPU](#) for detailed information.

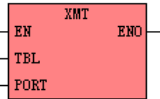
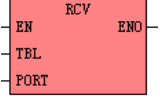
The general procedure to execute the Free-protocol Communication programming:

- Set the port communication parameter (including station number, baud rate, even-odd check, etc.) of the communication port of the **[PLC Hardware Configuration]**. For more detail please refer to [4.3.4.1 Parameters of the CPU](#).
- Set the free communication control register (see definition in following clauses). NOTE: the control register must be set ready in prior.
- Call XMT and RCV command and programme as status register and communication interruption of the free communication.

6.12.2 XMT and RCV

- Description

	Name	Usage	Influence	
				<input checked="" type="checkbox"/> CPU504

LD	XMT			<input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	RCV			
IL	XMT	XMT TBL, PORT	U	
	RCV	RCV TBL, PORT		

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>TBL</i>	Input	BYTE	I, Q, M, V, L, SM
<i>PORT</i>	Input	INT	Constant (0 to 2)

The *XMT* instruction is used to transmit the data stored in a data buffer through the communication port specified by *PORT* in free-protocol communication mode. The data buffer begins with address *TBL*, and the first byte specifies the number of bytes to be transmitted, then followed with the effective data. If *SM87.1=1*, when the CPU has transmitted the last character in the data buffer, there will automatically occur a XMT-complete interrupt event (the event number is 30 for *PORT 0*, and 32 for *PORT 1*). If the number of bytes to be transmitted is set to be 0, the *XMT* instruction won't execute any operation, and of course, the interrupt event won't occur.

The *RCV* instruction is used to receive data through the communication port specified by *PORT* in free-protocol communication mode, and the data received shall be stored in a data buffer. The data buffer begins with address *TBL*, and the first byte specifies the number of bytes received, then followed with the effective data received. You must specify a Start and an End condition for the *RCV* operation. If *SM87.1=1*, when the CPU completes receiving (disregarding normal or abnormal completion), there will automatically occur a RCV-complete interrupt event (the event number is 29 for *PORT 0*, and 31 for *PORT 1*).

In LD, the *EN* input decides whether to execute the *XMT* and *RCV* instructions. If *EN* is 1 then *XMT* and *RCV* instructions will be executed and vice versa;

In IL, CR decides whether to execute the *XMT* and *RCV* instructions. If CR is 1 then XMT and RCV instructions will be executed and vice versa; They won't influence CR.

➤ Status Registers and Control Registers in SM area for Free-protocol Communication

Besides XMT and RCV instructions, some status registers and control registers in SM area are provided for free-protocol communication. Your program can read and write to these registers to interpret the communication status and control the communication. The following is the brief summary of status bytes and control words.

➤ **SMB86 --- Receive Status Register**

Bit (read-only)			Status	Description
PORT 0	PORT 1	PORT 2		
SM86.0	SM186.0	SM286.0	1	A parity error is detected, but receive shall not be terminated.
SM86.1	SM186.1	SM286.1	1	Receive was terminated because of receiving the maximum character number.
SM86.2	SM186.2	SM286.2	1	Receive was terminated because of receiving a character Overtime.
SM86.3	SM186.3	SM286.3	1	Receive was terminated because of System Overtime.
SM86.4	SM186.4	SM286.4	-	Reserved.
SM86.5	SM186.5	SM286.5	1	Receive was terminated because of receiving the user-defined End character.
SM86.6	SM186.6	SM286.6	1	Receive was terminated because of the errors in the parameters or missing the Start or End condition.
SM86.7	SM186.7	SM286.7	1	Receive was terminated because of the user disable command.

➤ **SMB87 --- Receive Control Register**

Bit			Status	Description
PORT 0	PORT 1	PORT 2		
SM87.0	SM187.0	SM287.0	-	Reserved.
SM87.1	SM187.1	SM287.1	0	Disable XMT-complete and RCV-complete interrupts.
			1	Enable XMT-complete and RCV-complete interrupts.

SM87.2	SM187.2	SM287.2	0	Ignore SMW92/ SMW192/SMW292.
			1	Terminate receive if the time in SMW92/ SMW192/SMW292 is exceeded while receiving a character.
SM87.3	SM187.3	SM287.3	-	Reserved.
SM87.4	SM187.4	SM287.4	0	Ignore SMW90/ SMW190/ SMW290.
			1	Turn to effective receive if the time interval in SMW90/ SMW190/SMW290 is exceeded.
SM87.5	SM187.5	SM287.5	0	Ignore SMB89/ SMB189/ SMB289.
			1	Enable the user-defined End character in SMB89/ SMB189/ SMB289.
SM87.6	SM187.6	SM287.6	0	Ignore SMB88/ SMB188/ SMB288.
			1	Enable the user-defined Start character in SMB88/SMB188/ SMB288
SM87.7	SM187.7	SM287.7	0	Disable RCV function. This condition prevails over any other conditions.
			1	Enable RCV function.

➤ **Other Control Registers**

PORT 0	PORT 1	PORT 2	Description
SMB88	SMB188	SMB288	<p>To store the user-defined receive Start character.</p> <p>After executing the <i>RCV</i> instruction, the CPU turns into effective receive state when the Start character is received, and the previously received data will be rejected. CPU takes the Start character as the first effective byte received.</p> <p>SM87.6/ SM187.6/ SM287.6 should be set to be 1 to enable SMB88/ SMB188/ SMB288.</p>
SMB89	SMB189	SMB289	<p>To store the user-defined receive End character.</p> <p>The CPU will take this character as the last effective byte received. When the character is received, the CPU will immediately terminate receive disregarding any other End conditions.</p> <p>SM87.5/ SM187.5/ SM287.5 should be set to be 1 to enable SMB89/ SMB189/ SMB289.</p>

SMW90	SMW190	SMW290	<p>To store the user-defined receive Ready time (Range: 1~60,000ms). After executing the <i>RCV</i> instruction and passing through this time interval, the CPU will automatically turn into effective receive state disregarding whether the Start character is received or not. Thereafter, the data received shall be effective.</p> <p>SM87.4/SM287.4/ SM287.4 should be set to be 1 to enable SMW90/SMW190/ SMW290.</p>
SMW92	SMW192	SMW292	<p>To store the user-defined receiving a character Overtime (Range: 1~60,000ms). After executing the <i>RCV</i> instruction and turning into effective receive state, if no character is received within this time interval, the CPU will terminate receive disregarding any other End condition.</p> <p>SM87.2/SM187.2/SM287.2 should be set to be 1 to enable SMW92/SMW192/ SMW292.</p>
SMW94	SMW194	SMW294	<p>To store the maximum number of characters to be received (1~255). The CPU will immediately terminate receive as soon as the maximum effective characters are received disregarding any other End conditions.</p> <p>If this value is set to be 0, the <i>RCV</i> instruction will return directly.</p>

In free-protocol communication mode, there is a default System Receive Overtime (90 seconds). This overtime value functions as the following: After executing the *RCV* instruction, the CPU will immediately terminate receive if no data is received during this time interval. Besides, when the CPU turns into effective receive state, it will use the value of the receiving a character Overtime defined in SMW92 first, and if no valid value is in SMW92, the value of System Receive Overtime will be used as a substitute.

➤ **Communication Interruption**

Kinco-K5 offers variable interruption for free communications. If you would like to know more detail information, please refer to [6.10.1 How the Kinco K5 Handle Interruption Routines](#).

You may use SM87.1, SM187.1 or SM287.1 to allow or forbid CPU to interrupt communication. The interruption control set to 1 represent an allow routine; CPU will generate an interruption of sending completion when complete send the last word in buffer area; CPU will generate an interruption of receive completion when

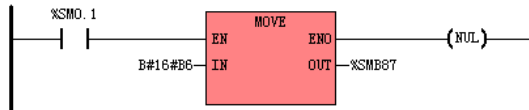
quit receiving (regardless of normal or abnormal quit).

➤ **Examples**

Examples are given below to illustrate the application of the free-protocol communication mode. In the example, the CPU will receive a character string, taking **RETURN** character as the receive End character; if receive is completed normally, the data received is transmitted back and receive is restarted, if receive is completed abnormally (e.g. because of communication errors, time out, etc), the data received will be ignored and receive will be restarted.

MAIN Program:

(* Network 0 *)
 (* The following program is to initialize free-protocol communication.
 At first,configure the Start and End conditions of the effective Receive state. *)

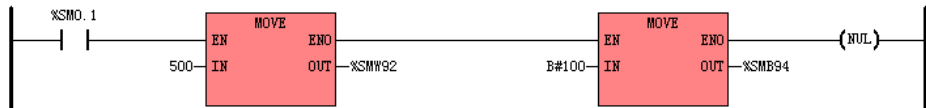


(* Network 1 *)
 (* The receive Ready time is set to be 10ms,
 The receive End character is set to be RETURN character whose ASCII is 13. *)

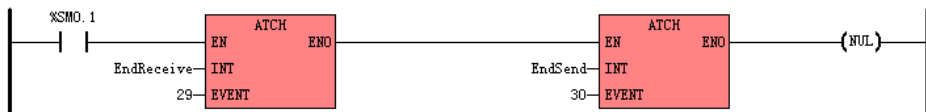


LD

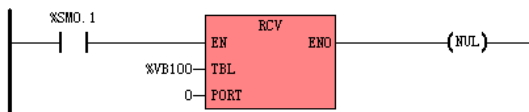
(* Network 2 *)
 (* The receiving a character Overtime is set to be 500ms,
 The maximum number of characters to be received is set to be 100. *)



(* Network 3 *)
 (* Attach the RCU-complete event to the EndReceiver routine,
 Attach the XMT-complete event to the EndSendroutine *)

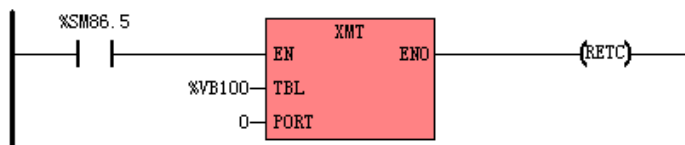


(* Network 4 *)
 (* Start the Receive task once on the first scan. *)



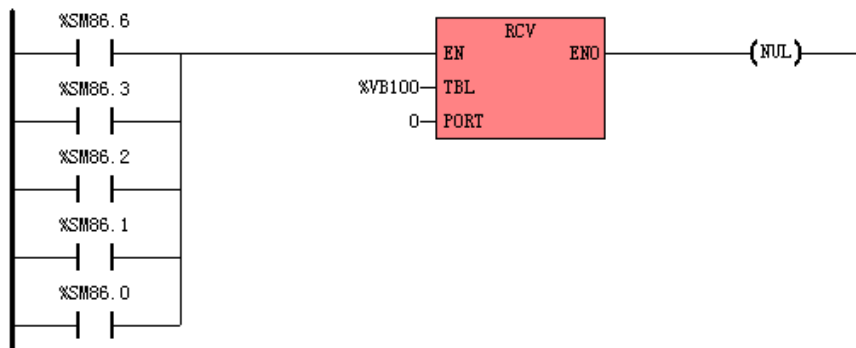
EndReceive (INT00): The RCV-complete interrupt routine

(* Network 0 *)
 (* If receiving the receive End character,
 then transmit back the data received and return. *)

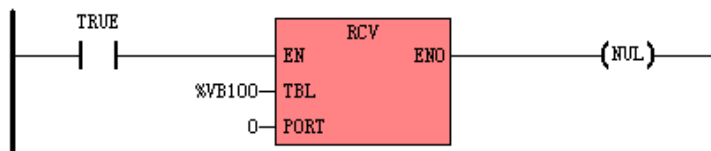


(* Network 1 *)
 (* if receive is completed abnormally, then restart receive. *)

LD

**EndSend (INT01): XMT-complete interrupt routine**

(* Network 0 *)
 (* Restart receive after the transmission is completed. *)



IL

MAIN Program:

(* Network 0 *)

(* The following program is to initialize free-protocol communication. *)

(* At first, configure the Start and End conditions of the effective Receive state. *)

```
LD      %SM0.1
MOVE    B#16#B6, %SMB87
```

(* Network 1 *)

(* The receive Ready time is set to be 10ms, *)

(* The receive End character is set to be RETURN character whose ASCII is 13. *)

```
LD      %SM0.1
MOVE    10, %SMW90
MOVE    B#16#D, %SMB89
```

(* Network 2 *)

(* The receiving a character Overtime is set to be 500ms, *)

(* The maximum number of characters to be received is set to be 100. *)

```
LD      %SM0.1
MOVE    500, %SMW92
MOVE    B#100, %SMB94
```

(* Network 3 *)

(* Attach the RCV-complete event to the EndReceiver routine, *)

(* Attach the XMT-complete event to the EndSendroutine *)

```
LD      %SM0.1
ATCH    EndReceive, 29
ATCH    EndSend, 30
```

(* Network 4 *)

(* Start the Receive task once on the first scan. *)

```
LD      %SM0.1
RCV     %VB100, 0
```

EndReive (INT00): The RCV-complete interrupt routine

(* Network 0 *)

(* If receiving the receive End character, then transmit bach the data received and return. *)

LD %SM86.5

XMT %VB100, 0

RETC

(* Network 1 *)

(* if receive is completed abnormally, then restart receive. *)

LD %SM86.6

OR %SM86.3

OR %SM86.2

OR %SM86.1

OR %SM86.0

RCV %VB100, 0

EndSend (INT01): XMT-complete interrupt routine

(* Network 0 *)

(* Restart receive after the transmittion is completed. *)

LD TRUE

RCV %VB100, 0

6.12.3 Modbus RTU Master Instructions

The Modbus RTU protocol is widely used in the industrial field. The KINCO-K5 provides the Modbus RTU Master instructions, and you can call them directly to make the KINCO-K5 as a Modbus RTU master.

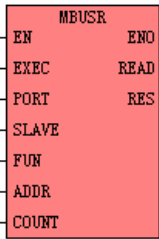
Note: these instructions are supported only by PORT1.

The general steps of the Modbus master programming are described as followings:

- Configure the communication parameters of Port1 in the **Hardware** Window. Please refer to [2.6 How to modify the CPU's communication parameters](#) and [4.3.4.1 Parameters of the CPU](#) for more details.
- Call the instructions MBUSR and MBUSW in the program.

6.12.3.1 MBUSR (Modbus RTU Master Read)

➤ Description

	Name	Usage	Group	
LD	MBUSR			<input type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	MBUSR	MBUSR EXEC, PORT, SLAVE, FUN, ADDR, COUNT, READ, RES	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
EXEC	Input	BOOL	I, Q, V, M, L, SM, RS, SR
PORT	Input	INT	Constant (0~2)
SLAVE	Input	BYTE	I, Q, M, V, L, SM, Constant
FUN	Input	INT	Constant (MODBUS function code)
ADDR	Input	INT	I, Q, M, V, L, SM, AI, AQ, Constant
COUNT	Input	INT	I, Q, M, V, L, SM, AI, AQ, Constant
READ	Output	BOOL, WORD, INT	Q, M, V, L, SM, AQ
RES	Output	BYTE	Q, M, V, L, SM

This instruction is used for reading data from a slave. The available function codes include 1 (read DO status), 2 (read DI status), 3 (read AO data) and 4 (Read AI data).

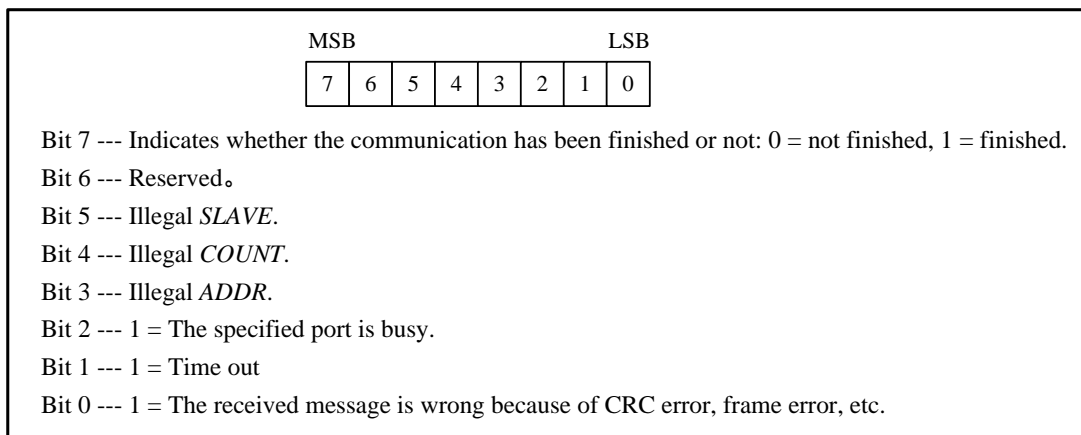
The parameter *PORT* defines the communication port used. The *SLAVE* defines the target slave address, whose available range is 1~31. The *FUN* defines a valid function code. The *ADDR* defines the starting address of the Modbus register to be read. The *COUNT* defines the number (Max. 32) of the registers to be read.

The rising edge of *EXEC* is used for starting the communication. While a MBUSR instruction is executed, it will communicate for one time on the rising edge of *EXEC*: Organize a Modbus RTU message according to the

parameters *SLAVE*, *FUN*, *ADDR* and *COUNT*, then transmit it and wait for the response of the slave; When receiving the slave's response message, check the CRC, slave number and function code to decide whether the message is correct or not, if correct, the useful data will be written into the buffer beginning with *READ*, otherwise, the received message will be discarded.

The *READ* defines the starting address of a buffer, which stores the received data. The data type of *READ* must match the function code. If the function code is of 1 or 2, the *READ* is of *BOOL* type; and if the function code is of 3 or 4, the *READ* is of *INT* or *WORD* type.

The *RES* stores the communication status and the failure information of the current execution, and it is read-only. It is described in the following figure.



■ **LD**

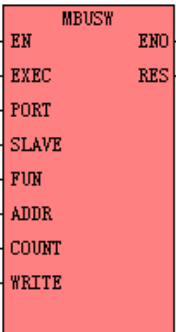
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.12.3.2 MBUSW (Modbus RTU Master Write)

➤ Description

	Name	Usage	Group	
LD	MBUSW			<input type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	MBUSW	MBUSW EXEC, PORT, SLAVE, FUN, ADDR, COUNT, READ, RES	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
EXEC	Input	BOOL	I, Q, V, M, L, SM, RS, SR
PORT	Input	INT	Constant (0~2)
SLAVE	Input	BYTE	I, Q, M, V, L, SM, Constant
FUN	Input	INT	Constant (MODBUS function code)
ADDR	Input	INT	I, Q, M, V, L, SM, AI, AQ, Constant
COUNT	Input	INT	I, Q, M, V, L, SM, AI, AQ, Constant
WRITE	Input	BOOL, WORD, INT	I, Q, RS, SR, V, M, L, SM, T, C, AI, AQ
RES	Output	BYTE	Q, M, V, L, SM

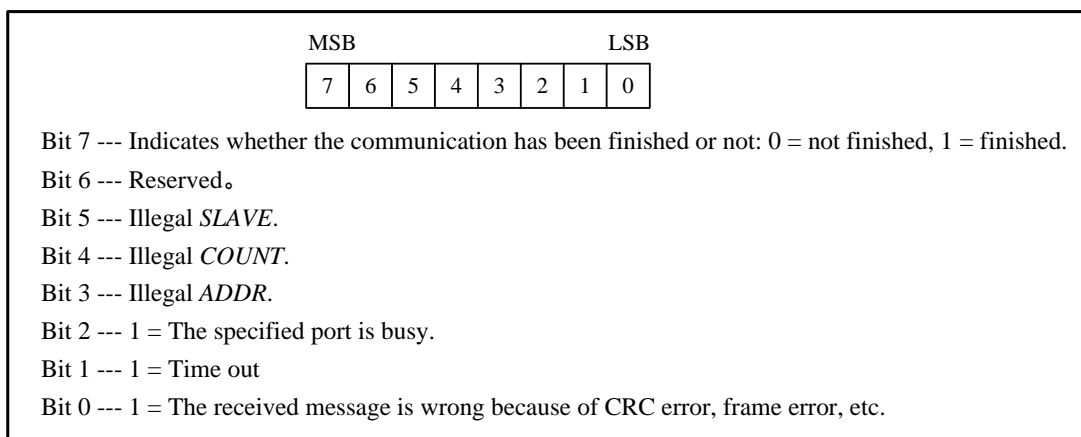
This instruction is used for writing data to a slave. The available function codes include (write to a DO), 6 (write to an AO), 15 (write to several Dos) and 16 (write to several AOs).

The parameter *PORT* defines the communication port used. The *SLAVE* defines the target slave address, whose available range is 1~31. The *FUN* defines a valid function code. The *ADDR* defines the starting address of the Modbus register to be written into. The *COUNT* defines the number (Max. 32) of the registers.

The *WRITE* defines the starting address of a buffer, which stores the data to be written into the slave. The data type of *WRITE* must match the function code. If the function code is of 5 or 15, the *WRITE* is of *BOOL* type; and if the function code is of 6 or 16, the *WRITE* is of *INT* or *WORD* type.

The rising edge of *EXEC* is used for starting the communication. While a *MBUSW* instruction is executed, it will communicate for one time on the rising edge of *EXEC*: Organize a Modbus RTU message according to the parameters *SLAVE*, *FUN*, *ADDR*, *COUNT* and *WRITE*, then transmit it and wait for the response of the slave; When receiving the slave's response message, check the *CRC*, slave number and function code to decide whether the target slave executed the command correctly or not.

The *RES* stores the communication status and the failure information of the current execution, and it is read-only. It is described in the following figure.



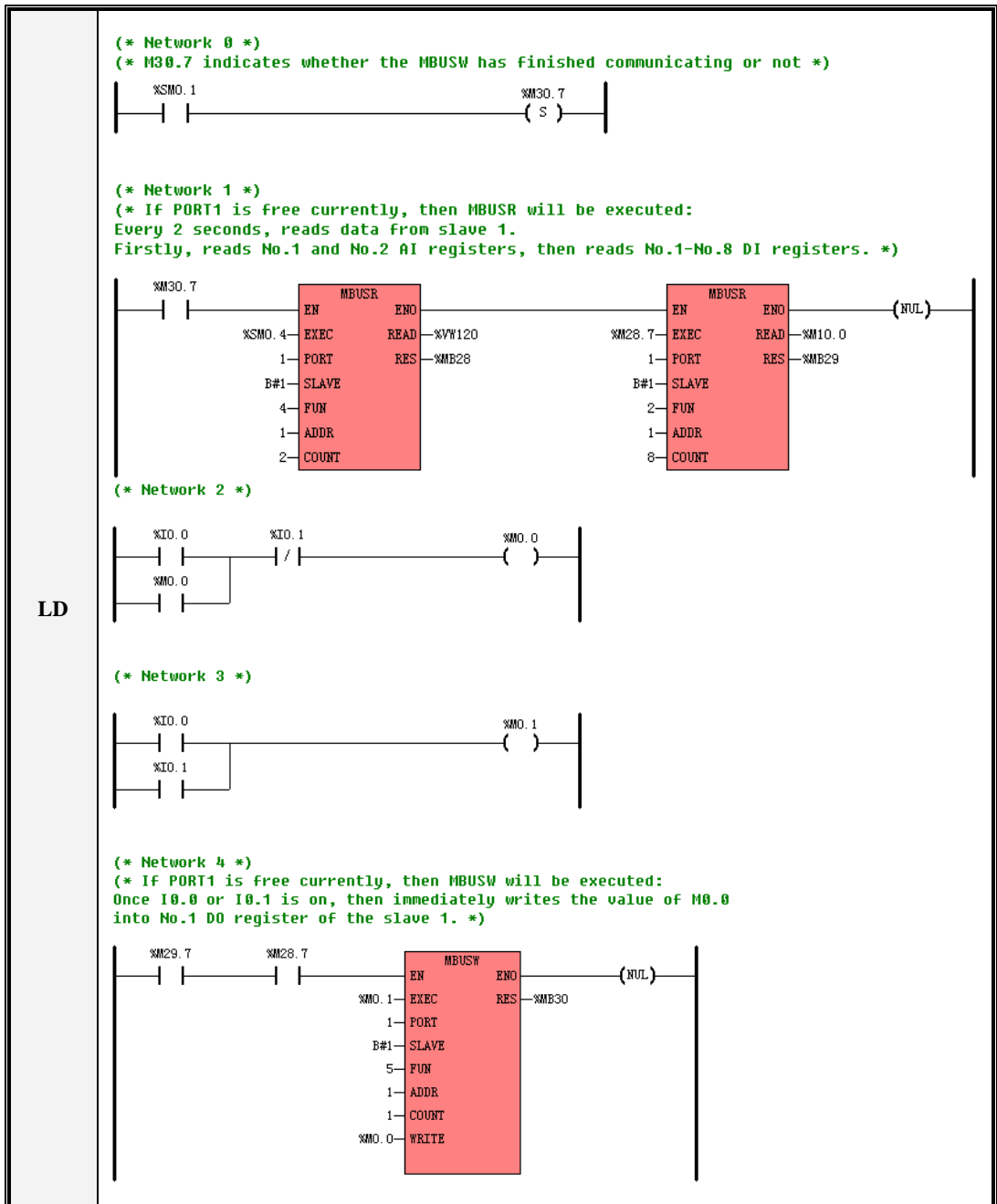
■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.12.3.3 Example for MBUSR and MBUSW



IL	<p>(* Network 0 *)</p> <p>(* M30.7 indicates whether the MBUSW has finished communicating or not*)</p> <pre>LD %SM0.1 S %M30.7</pre> <p>(* Network 1 *)</p> <p>(* If PORT1 is free currently, then MBUSR will be executed: *)</p> <p>(* Every 2 seconds, reads data from slave 1. *)</p> <p>(* Firstly, reads No.1 and No.2 AI registers, then reads No.1-No.8 DI registers.*)</p> <pre>LD %M30.7 MBUSR %SM0.4, 1, B#1, 4, 1, 2, %VW120, %MB28 MBUSR %M28.7, 1, B#1, 2, 1, 8, %M10.0, %MB29</pre> <p>(* Network 2 *)</p> <pre>LD %I0.0 OR %M0.0 ANDN %I0.1 ST %M0.0</pre> <p>(* Network 3 *)</p> <pre>LD %I0.0 OR %I0.1 ST %M0.1</pre> <p>(* Network 4 *)</p> <p>(* If PORT1 is free currently, then MBUSW will be executed: *)</p> <p>(* Once I0.0 or I0.1 is on, then immediately writes the value of M0.0 *)</p> <p>(* into No.1 DO register of the slave 1.*)</p> <pre>LD %M29.7 AND %M28.7 MBUSW %M0.1, 1, B#1, 5, 1, 1, %M0.0, %MB30</pre>
-----------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.12.4 SDO

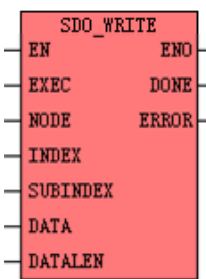
SDO is short for Service Data Object in CANOpen protocol. It accesses data in the dictionary of one device

through index and subindex. The visitor is called Client and the one to be visited is called Server. Client's request will surely be responded by Server.

SDO is mainly used to transfer low-priority data among devices. It is suitable for configuration and management to devices but not data transmission for data that requires high real-time performance.

6.12.3.1 SDO_WRITE

➤ Description

	Name	Usage	Group
LD	SDO_WRITE		<input type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	SDO_WRITE	SDO_WRITE EXEC, NODE, INDEX, SUBINDEX, DATA, DATALEN, DONE, ERROR	U

Operands	Input/Output	Data Type	Acceptable Memory Areas
EXEC	Input	BOOL	I, Q, V, M, L, SM, RS, SR
NODE	Input	BYTE	I, Q, V, M, L, SM, Constant
INDEX	Input	WORD	I, Q, V, M, L, SM, Constant
SUBINDEX	Input	BYTE	I, Q, V, M, L, SM, Constant
DATA	Input	BYTE	I, Q, V, M, L, SM
DATALEN	Input	BYTE	I, Q, V, M, L, SM, Constant
DONE	Output	BOOL	Q, M, V, L, SM
ERROR	Output	DWORD	Q, M, V, L, SM

NODE, INDEX, SUBINDEX, DATALEN should be constants or variables simultaneously.

Operands	Description
EN	Enable. If EN is 1 then this command will be executed.
EXEC	Execution. The uprise of EXEC will be adopted to SDO communication start. It is better to ensure the EN is prior to EXEC.
NODEID	Node address to be visited
Index	Index of object to be visited in the OD
SubIndex	Subindex of object to be visited in the OD
Data	Saved initial word of data to be sent
DataLen	Length of data. Unit: word
DONE	Execution result display. If SDO is being executed then DONE is 0; if SDO ends (response received or over time limit) DONE is 1
ERROR	Errors. Please see below

Code	Description
0	No errors.
1	SDO command amount out of limitation K5 allows a maximum amount of 72 regarding SDO_WRITE and SDO_READ commands in one project.
2	The main station is not in Operational Status so SDO report will not be sent.
4	Node does not exist or stops due to error so SDO report will not be sent.
5	Last similar command did not receive response.
6	Command parameter error.
8	Did not receive report due to out of time limitation. SDO time can be set in [Main Station and Whole Configuration]
9	Data length of responding report error.
10	Responding report is not as expected.

■ **LD**

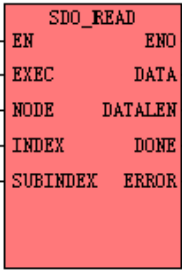
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.12.3.2 SDO_READ

➤ Description

	Name	Usage	Group	
LD	SDO_READ	 <pre> SDO_READ - EN ENO - EXEC DATA - NODE DATALEN - INDEX DONE - SUBINDEX ERROR </pre>		<input type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	SDO_READ	<i>SDO_READ EXEC, NODE, INDEX, SUBINDEX, DATA, DATALEN, DONE, ERROR</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
EXEC	Input	BOOL	I, Q, V, M, L, SM, RS, SR
NODE	Input	BYTE	I, Q, V, M, L, SM, Constant
INDEX	Input	WORD	I, Q, V, M, L, SM, Constant
SUBINDEX	Input	BYTE	I, Q, V, M, L, SM, Constant
DATA	Output	BYTE	I, Q, V, M, L, SM

DATALEN	Output	BYTE	I, Q, V, M, L, SM
DONE	Output	BOOL	Q, M, V, L, SM
ERROR	Output	DWORD	Q, M, V, L, SM

Operands	Description
EN	Enable. If EN is 1 then this command will be executed.
EXEC	Execution. The uprise of EXEC will be adopted to SDO communication start. It is better to ensure the EN is prior to EXEC.
NODEID	Node address to be visited
Index	Index of object to be visited in the OD
SubIndex	Subindex of object to be visited in the OD
Data	Saved initial word of data to be sent
DataLen	Length of data. Unit: word
DONE	Execution result display. If SDO is being executed then DONE is 0; if SDO ends (response received or over time limit) DONE is 1
ERROR	Errors. Please see below

Code	Description
0	No errors.
1	SDO command amount out of limitation K5 allows a maximum amount of 72 regarding SDO_WRITE and SDO_READ commands in one project.
2	The main station is not in Operational Status so SDO report will not be sent.
4	Node does not exist or stops due to error so SDO report will not be sent.
5	Last similar command did not receive response.
6	Command parameter error.

8	Did not receive report due to out of time limitation. SDO time can be set in [Main Station and Whole Configuration]
9	Data length of responding report error.
10	Responding report is not as expected.

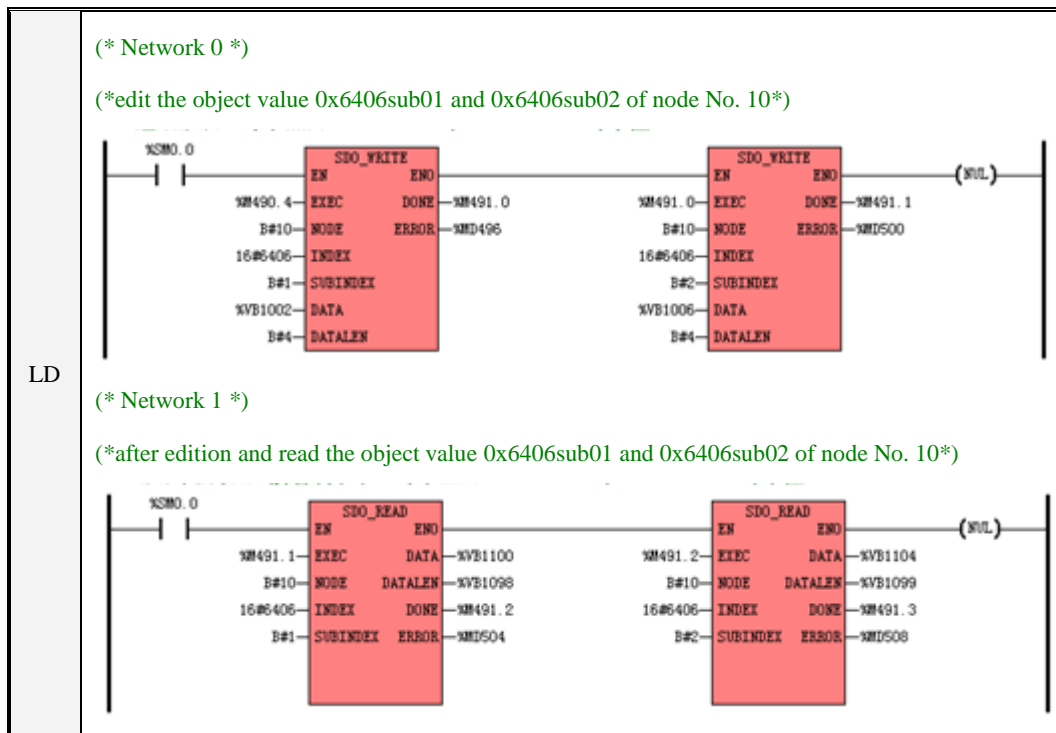
■ **LD**

If EN is 1, this instruction is executed.

■ **IL**

If CR is 1, this instruction is executed, and it does not influence CR.

6.12.3.3 Example for SDO_WRITE and SDO_READ



IL	<pre style="margin: 0;">(* Network 0 *) (*edit the object value 0x6406sub01 and 0x6406sub02 of node No. 10*) LD %SM0.0 SDO_WRITE %M490.4, B#10, 16#6406, B#1, %VB1002, B#4, %M491.0, %MD496 SDO_WRITE %M491.0, B#10, 16#6406, B#2, %VB1006, B#4, %M491.1, %MD500 (* Network 1 *) (*after edition and read the object value 0x6406sub01 and 0x6406sub02 of node No. 10*) LD %SM0.0 SDO_READ %M491.1, B#10, 16#6406, B#1, %VB1100, %VB1098, %M491.2, %MD504 SDO_READ %M491.2, B#10, 16#6406, B#2, %VB1104, %VB1099, %M491.3, %MD508</pre>
-----------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.12.5 CAN Communication Command

K5 provide the functions of CANOpen main station and CAN free communication, which should be adopted with K541 module.

The two commands can be used at the same time. Please be noted that when used at the same time the baud rate of all nodes should be the same.

CAN communication command supports CAN2.0A and CAN2.0B. The command only supports data frame but not long distance frame. The format is as follows:

ID	Word 1-8
11 digits (CAN2.0A, standard frame) or 29 digits (CAN 2.0B, extension frame)	Data length of 1-8 words

6.12.4.1 CAN_INI

➤ Description

	Name	Usage	Group	
LD	CAN_INIT			<input type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	CAN_INIT	CAN_INIT CH, BAUD	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
EN	Input	BOOL	I, Q, V, M, L, SM
CH	Input	INT	Constant(2)
BAUD	Input	INT	L, M, V, Constant
ERR	Output	BOOL	L, M, V, Constant

Operands	Description
EN	Enable.
CH	The CAN port. 2 represents K541 module.
BAUD	Baud rate CAN 8 --- 1000K 7 --- 800K 6 --- 500K 5 --- 250K 4 --- 125K 3 --- 50K 2 --- 20K 1 --- 10K
ERR	Execution of the commands. 1 represents 0 success and 1 represents error.

The rise change of EN input end will trigger this command. It can be used to CAN port(CH) to assign

initialization and set the CAN baud rate to the value BAUD represents

➤ LD

The rise change of EN will trigger this command once and vice versa.

➤ IL

The rise change of CR will trigger this command once and vice versa.

The command will not affect the CR value.

6.12.4.2 CAN_WRITE

➤ Description

	Name	Usage	Group	
LD	CAN_WRITE	<div style="border: 1px solid black; background-color: #f0f0f0; padding: 5px; display: inline-block;"> <pre> CAN_WRITE - EN ENO - CH DONE - ID ERR - FMT - DATA - LEN </pre> </div>		<input type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	CAN_WRITE	CAN_WRITE CH, ID, FMT, DATA, LEN, DONE, ERR	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
CH	Input	INT	Constant(2)
ID	Input	DWORD	L, M, V, Constant
FMT	Input	BYTE	L, M, V, Constant
DATA	Input	BYTE	L, M, V

LEN	Input	BYTE	L, M, V, Constant
DONE	Output	BOOL	L, M, V
ERR	Output	BOOL	L, M, V

Operands	Description
EN	Enable.
CH	The CAN port. 2 represents K541 module.
FMT	Report format. 0 represents standard frame and 1 represents extension frame.
DATA	Saved initial word address of sending data.
LEN	Length of data. Unit: word.
DONE	Execution result display. If executed then DONE is 0; if ends (response received or over time limit) DONE is 1.
ERR	Report status display. If sending fails (usually caused by a full sending-buffer area) then ERR will be reset to 1.

Sending CAN report will be decided by ID, format (FMT, standard frame or extension frame), data (DATA, address of memory storage) and Length (LEN).

Rise change of EN input end will trigger the command once and send the report to the sending-buffer area, then it be send via certain CAN port (CH).

If the command successfully send the report to the buffer, the execution will be done and DONE will be set to 1.

If the buffer area is full then the sending will fail, DONE and ERR will be set to 1.

➤ LD

The rise change of EN will trigger this command once and vice versa.

➤ IL

The rise change of CR will trigger this command once and vice versa.

The command will not affect the CR value.

6.12.4.3 CAN_READ

➤ Description

	Name	Usage	Group	
LD	CAN_READ	<div style="border: 1px solid black; background-color: #f0f0f0; padding: 5px; display: inline-block;"> CAN_READ EN ENO CH DONE TIME ERR ID FMT DATA LEN </div>		<input type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	CAN_READ	CAN_READ CH,TIME, DONE,ERR , FMT,DATA,LEN	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
CH	Input	INT	Constant (only 2 is allowed)
TIME	Input	INT	L, M, V, Constant
DONE	Output	BOOL	L, M, V
ERR	Output	BOOL	L, M, V
ID	Output	DWORD	L, M, V
FMT	Output	BYTE	L, M, V
DATA	Output	BYTE	M, V
LEN	Output	BYTE	L, M, V

Operands	Description
EN	Enable.

CH	The CAN port. 2 represents K541 module.
FMT	Report format. 0 represents standard frame and 1 represents extension frame.
DATA	Saved initial word address of sending data.
LEN	Length of data. Unit: word.
DONE	Execution result display. If executed then DONE is 0; if ends (response received or over time limit) DONE is 1.
ERR	Report status display. If sending fails (usually caused by a full sending-buffer area) then ERR will be reset to 1.
EN	Enable.
CH	The CAN port. 2 represents K541 module.

The rise change of EN input end will trigger this command: start receiving and receive any report from certain CAN port (CH).

When receiving starts, if PLC receive a CAN report within time limitation, it will put the report to output parameter ID, FMT, DATA and LEN respectively, it will then set DONE to 1 and quit receiving. If it does not receive any report within the time limitation, it will quit receiving and set DONE and ERR to 1.

After CAN_READ command starts, PLC will receive any report from any certain CAN port. Please pay attention when using along with other protocols (e.g. CANOpen).

➤ LD

The rise change of EN will trigger this command once and vice versa.

➤ IL

The rise change of CR will trigger this command once and vice versa.

The command will not affect the CR value.

6.12.4.3 CAN_RX

➤ Description

	Name	Usage	Group	
LD	CAN_RX	CAN_RX EN ENO CH DONE ID ERR FMT DATA MODE LEN TIME		<input type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	CAN_RX	CAN_RX CH, ID, FMT, MODE, TIME, DONE, ERR, FMT, DATA, LEN	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
CH	Input	INT	Constant (only 2 is allowed)
ID	Input	DWORD	L, M, V, Constant
FMT	Input	INT	L, M, V, Constant
MODE	Input	INT	L, M, V, Constant
TIME	Input	INT	L, M, V, Constant
DONE	Output	BOOL	L, M, V
ERR	Output	BOOL	L, M, V
DATA	Output	BYTE	M, V
LEN	Output	BYTE	L, M, V

Operands	Description
EN	Enable.
CH	Current CAN port. 2 represents K541 module.
ID	ID that expect to receive reports.
FMT	Format of report. 0 represents standard frame and 1 represents extension frame.
MODE	Receive mode. 0 represents permanent receiving and 1 represents single receiving
TIME	Receiving out of time limitation. Unit: ms
DONE	DONE represents successfully receive in single receiving mode.
ERR	Receive out of time.

DATA	The initial address of last receive report data.
LEN	The length of last receive report data. Unit: word.

The rise change of EN input end will trigger this command: start receiving and receive any report from certain CAN port (CH).

When receiving starts, if PLC receive a CAN report within time limitation, it will put the report to output parameter ID, FMT, DATA and LEN respectively, it will then set DONE to 1 and quit receiving. If it does not receive any report within the time limitation, it will quit receiving and set DONE and ERR to 1.

After CAN_READ command starts, PLC will receive any report from any certain CAN port. Please pay attention when using along with other protocols (e.g. CANOpen).

➤ LD

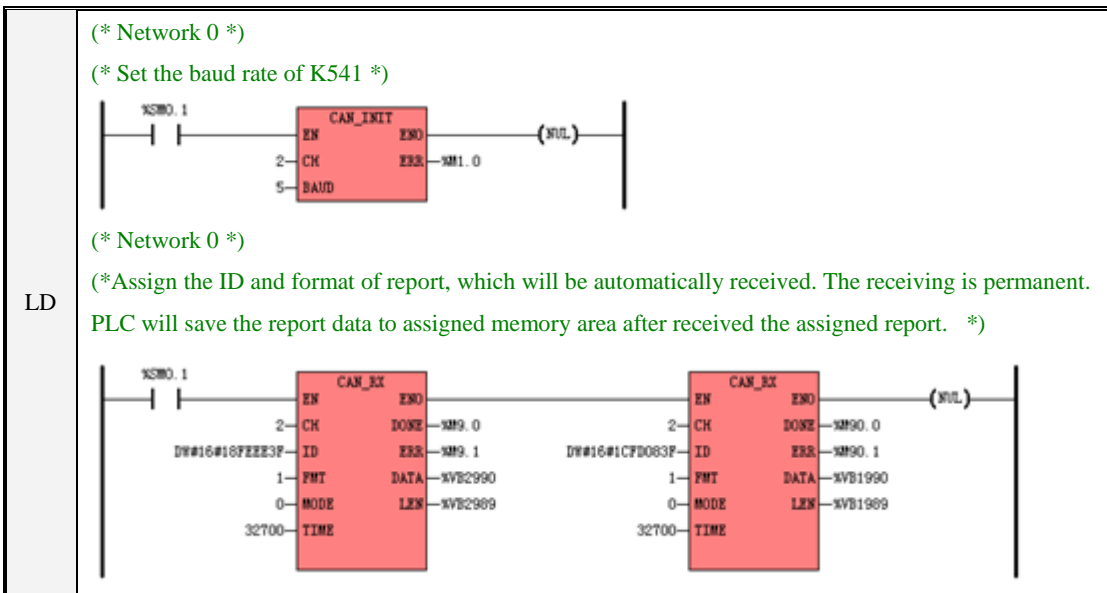
The rise change of EN will trigger this command once and vice versa.

➤ IL

The rise change of CR will trigger this command once and vice versa.

The command will not affect the CR value.

➤ Examples



6.13 Counters

6.13.1 CTU (Up Counter) and CTD (Down Counter)

Counter is one of the function blocks defined in the IEC61131-3 standard, totally in three types i.e. CTU, CTD and CTUD. Please refer to [3.6.5 Function Block and Function Block Instance](#) for more detailed information.

➤ Description

	Name	Usage	Group
LD	CTU		
	CTD		
IL	CTU	CTU Cx, R, PV	P
	CTD	CTD Cx, LD, PV	

- CPU504
- CPU504EX
- CPU506
- CPU506EA
- CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>Cx</i>	-	Counter instance	C
<i>CU</i>	Input	BOOL	Power flow
<i>R</i>	Input	BOOL	I, Q, M, V, L, SM, T, C
<i>CD</i>	Input	BOOL	Power flow
<i>LD</i>	Input	BOOL	I, Q, M, V, L, SM, T, C
<i>Q</i>	Output	BOOL	Power flow
<i>CV</i>	Output	INT	Q, M, V, L, SM, AQ

■ LD

The *CTU* counter counts up on the rising edge of the *CU* input. When the current value *CV* is equal to or greater

than the preset value *PV*, both the counter output *Q* and the status bit of *Cx* are set to be 1. *Cx* is reset when the reset input *R* is enabled. When the counter reaches *PV*, it continues counting until it reaches and keeps at the maximum INT value (i.e. 32767).

The *CTD* counter counts down on the rising edge of the *CD* input. When the current value *CV* is equal to or greater than the preset value *PV*, both the counter output *Q* and the status bit of *Cx* are set to be 1. *Cx* is reset and *PV* is loaded into *CV* when the load input *LD* is enabled. When the counter reaches *PV*, it continues counting until it reaches and keeps at 0.

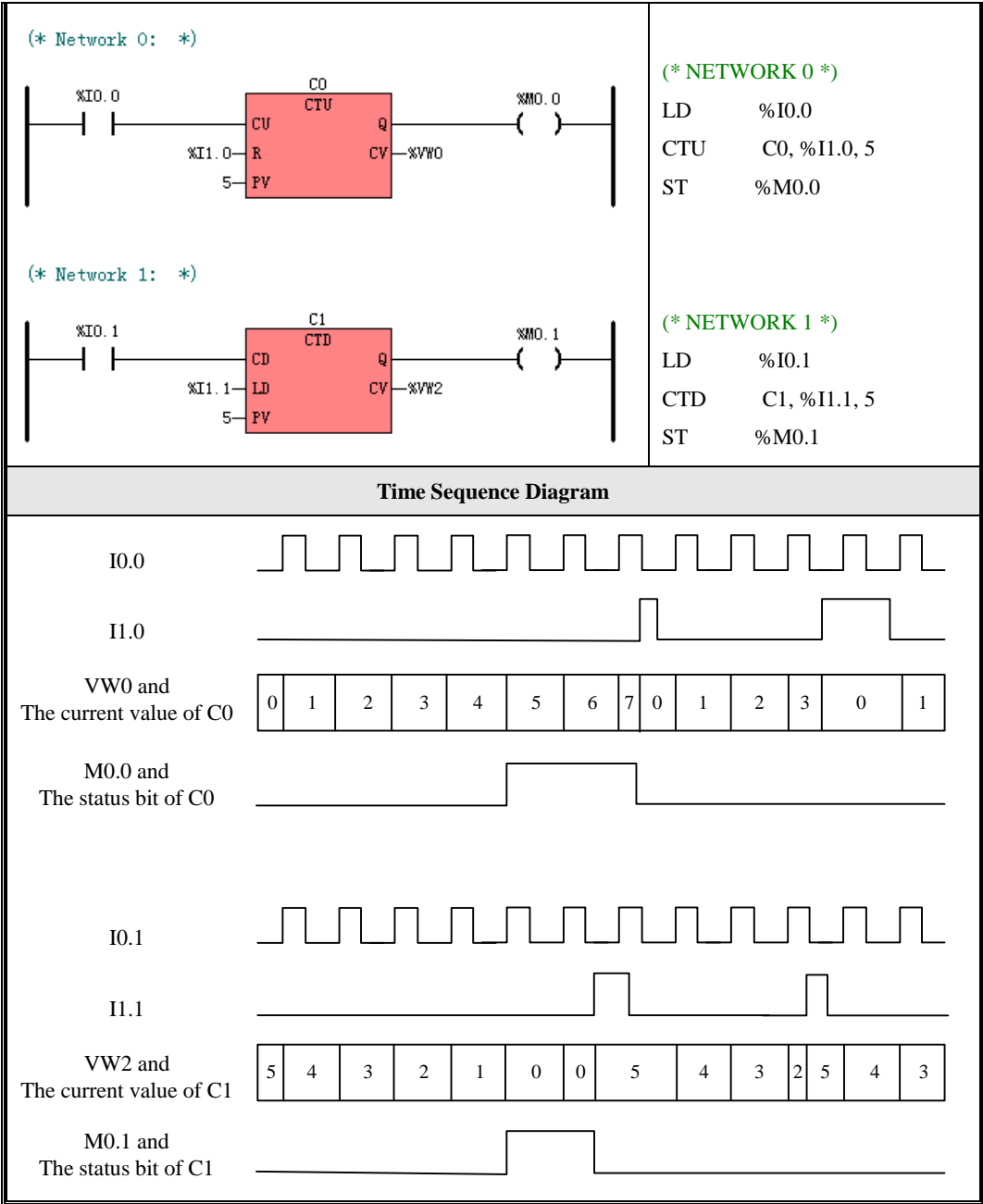
■ **IL**

The *CTU* counter counts up on the rising edge of *CR*. When the current value of *Cx* is equal to or greater than the preset value *PV*, the counter status bit are set to be 1. *Cx* is reset when the reset input *R* is enabled. When the counter reaches *PV*, it continues counting until it reaches and keeps at the maximum INT value (i.e. 32767). After each scan, *CR* is set to be the status bit value of *Cx*.

The *CTD* counter counts down on the rising edge of *CR*. When the current value of *Cx* is equal to or greater than the preset value *PV*, the counter status bit are set to be 1. *Cx* is reset and *PV* is loaded into the current value when the load input *LD* is enabled. When the counter reaches *PV*, it continues counting until it reaches and keeps at 0. After each scan, *CR* is set to be the status bit value of *Cx*.

➤ Examples

LD	IL
-----------	-----------



6.13.2 CTUD (Up-Down Counter)

➤ Description

	Name	Usage	Group	
LD	CTUD			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	CTUD	CTUD Cx, CD, R, LD, PV, QD	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>Cx</i>	-	Counter instance	C
<i>CU</i>	Input	BOOL	Power flow
<i>CD</i>	Input	BOOL	I, Q, M, V, L, SM, T, C, RS, SR
<i>R</i>	Input	BOOL	I, Q, M, V, L, SM, T, C, RS, SR
<i>LD</i>	Input	BOOL	I, Q, M, V, L, SM, T, C, RS, SR
<i>PV</i>	Input	INT	I, Q, M, V, L, SM, AI, AQ, constant
<i>QU</i>	Output	BOOL	Power flow
<i>QD</i>	Output	BOOL	Q, M, V, L, SM
<i>CV</i>	Output	INT	Q, M, V, L, SM, AQ

■ LD

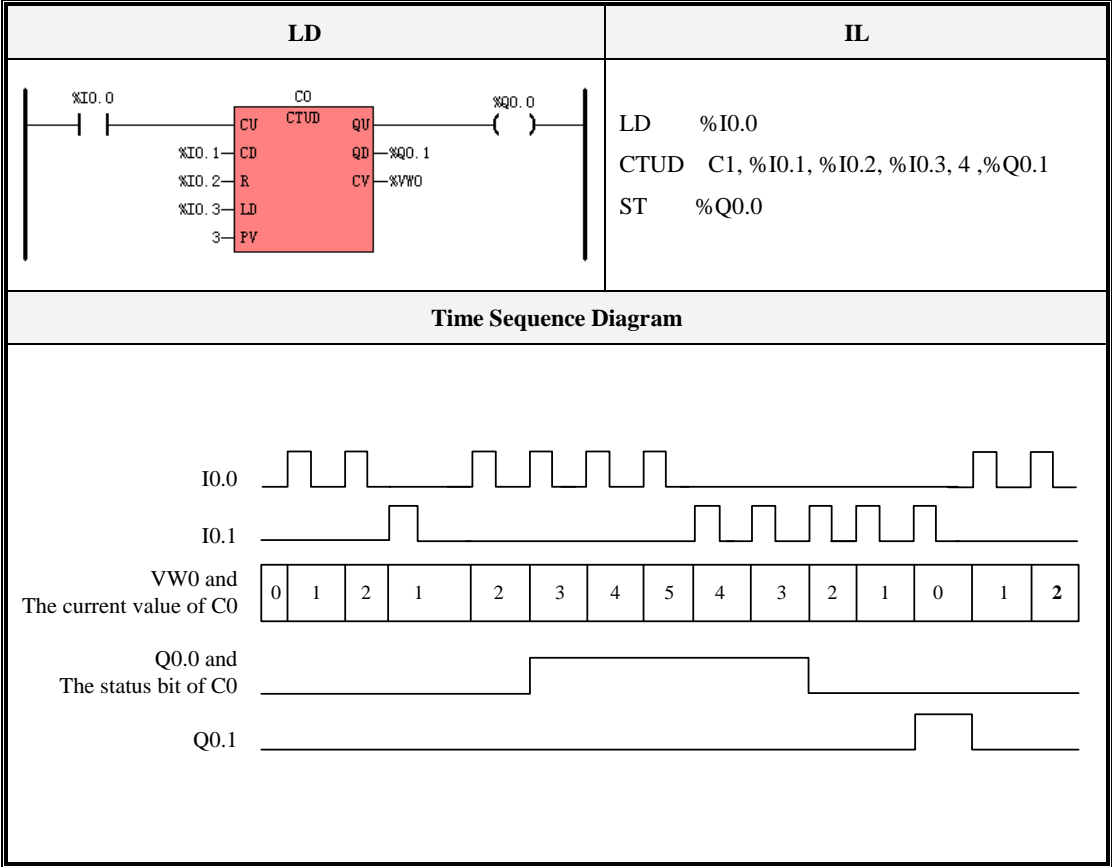
The *CTUD* counter counts up on the rising edge of the *CU* input and counts down on the rising edge of the *CD* input, and the current counter value *Cx* is assigned to *CV*. When *CV* is equal to or greater than the preset value *PV*, both *QU* and the status bit of *Cx* are set to 1, otherwise they are set to 0. When *CV* is equal to 0, *QD* is set to 1, otherwise it is set to 0. When the reset input *R* is enabled, *Cx* and *CV* is reset. When the load input *LD* is enabled, *PV* is loaded into *Cx* and *CV*. If *R* and *LD* are 1 at the same time, *R* takes the higher priority.

■ IL

The *CTUD* counter counts up on the rising edge of *CR* and counts down on the rising edge of the *CD* input, and the current counter value *Cx* is assigned to *CV*. When *CV* is equal to or greater than the preset value *PV*, both *QU* and the status bit of *Cx* are set to 1, otherwise they are set to 0. When *CV* is equal to 0, *QD* is set to 1, otherwise it is set to 0. When the reset input *R* is enabled, *Cx* and *CV* is reset. When the load input *LD* is enabled, *PV* is loaded into *Cx* and *CV*. If *R* and *LD* are 1 at the same time, *R* takes the higher priority.

After each scan, *CR* is set to be the status bit value of *Cx*.

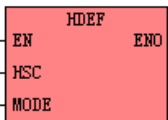
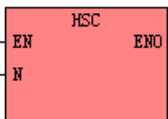
➤ Example



6.13.3 High-speed Counter Instructions

High-speed counters count high-speed pulse inputs that cannot be controlled at the CPU scan rate.

➤ Description

	Name	Usage	Group	
LD	HDEF			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
	HSC			
IL	HDEF	HDEF HSC, MODE	U	
	HSC	HSC N		

Operands	Input/Output	Data Type	Description
<i>HSC</i>	Input	INT constant (0~5)	HSC number
<i>MODE</i>	Input	INT constant (0~11)	Operations mode
<i>N</i>	Input	INT constant (0~5)	HSC number

The *HDEF* (High-speed Counter Definition) instruction is used to define the operation mode (*MODE*) of a high-speed counter (*HSC*). This instruction is suitable for each high-speed counter. A high-speed counter can be configured to be one of the 11 different operation modes. The mode decides the clock input, counting direction, start, and reset properties of the high-speed counter.

The *HSC* (High-Speed Counter) instruction configures and operates the high-speed counter whose number is specified by *N* according to the values of the corresponding *SM* registers.

In *IL*, *CR* decides whether to execute the *HDEF* and *HSC* instructions. They won't influence *CR*.

➤ **LD**

If the *EN* value is 1 then execute *HDEF* and *HSC* and vice versa.

➤ **IL**

If the *CR* value is 1 then execute *HDEF* and *HSC* and vice versa..

The execution of *HDEF* and *HSC* will not affect the *CR* value.

6.13.3.1 High-speed Counters Supported by the Kinco-K5

Feature	CPU504, CPU504EX, CPU506, CPU506EA, CPU508
High-speed counters	2 counters (HSC0 and HSC1)
Single phase	2 at 60KHz
Two phase	2 at 20KHz.

Before started, the high-speed counter should be assigned an operation mode by *HDEF* command. All the high-speed counters have the same function in the same operation mode.

Each input of a high-speed counter functions as follows:

6.13.3.2 Operation Modes and Inputs of the High-speed Counters

Input signals of high-speed counter include: clock (input impulse), direction, start and reset.

In different operation modes input signals is different. Please see below:

HSC 0				
Mode	Description	I0.1	I0.0	I0.5
0	Single-phase up/down counter with internal direction control: SM37.3	Clock		
1			Reset	
2			Reset	Start
3	Single-phase up/down counter with external direction control	Clock		Direction
4			Reset	Direction
6	Two-phase counter with up/down clock inputs	Clock Up	Clock Down	
9	A/B phase quadrature counter	Clock A	Clock B	

HSC 1					
Mode	Description	I0.3	I0.7	I1.2	I1.3
0	Single-phase up/down counter with internal direction control: SM47.3			Clock	
1		Reset			
2		Reset	Start		
3	Single-phase up/down counter with external direction control			Clock	Direction
4		Reset			Direction
5		Reset	Start		Direction
6	Two-phase counter with up/down clock inputs			Clock Down	Clock Up
7		Reset			
8		Reset	Start		
9	A/B phase quadrature counter			Clock A	Clock B
10		Reset			
11		Reset	Start		

6.13.3.3 Time Sequence of High-speed Counter

In order to help you well understand the high-speed counter, the following diagrams shows various time sequences.

➤ **Reset and Start**

The operations in the following figures are suitable for all modes that use the reset and start inputs.

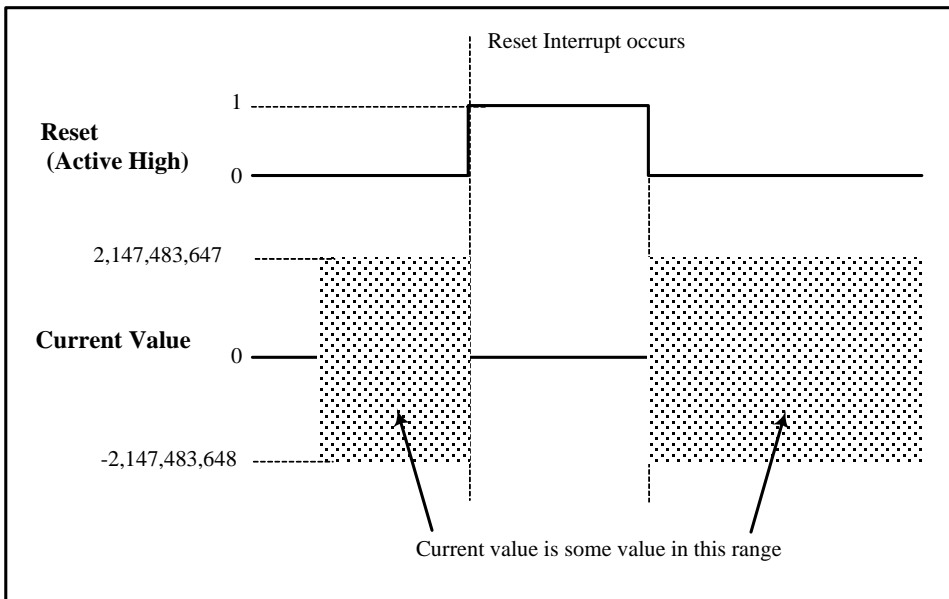


Figure 6-2 Time Sequence with Reset and without Start

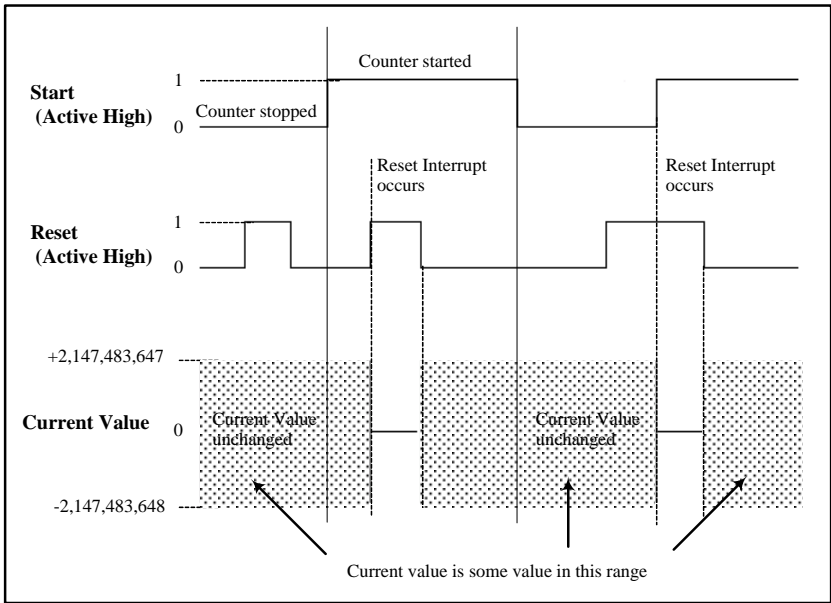


Figure 6-3 Time Sequence with Reset and Start

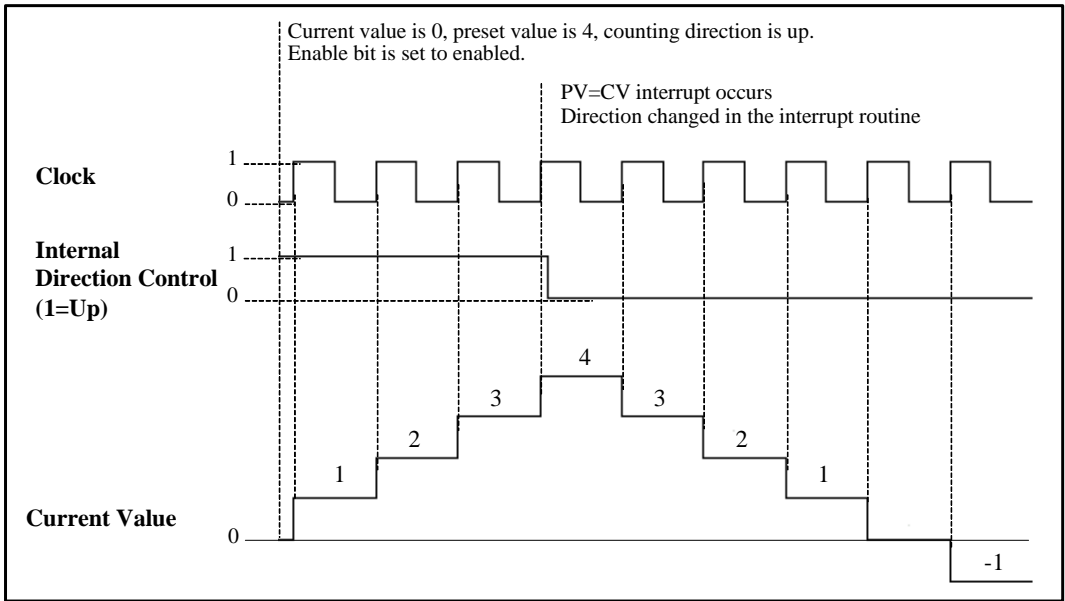


Figure 6-4 Time Sequence of Mode 0, 1 or 2

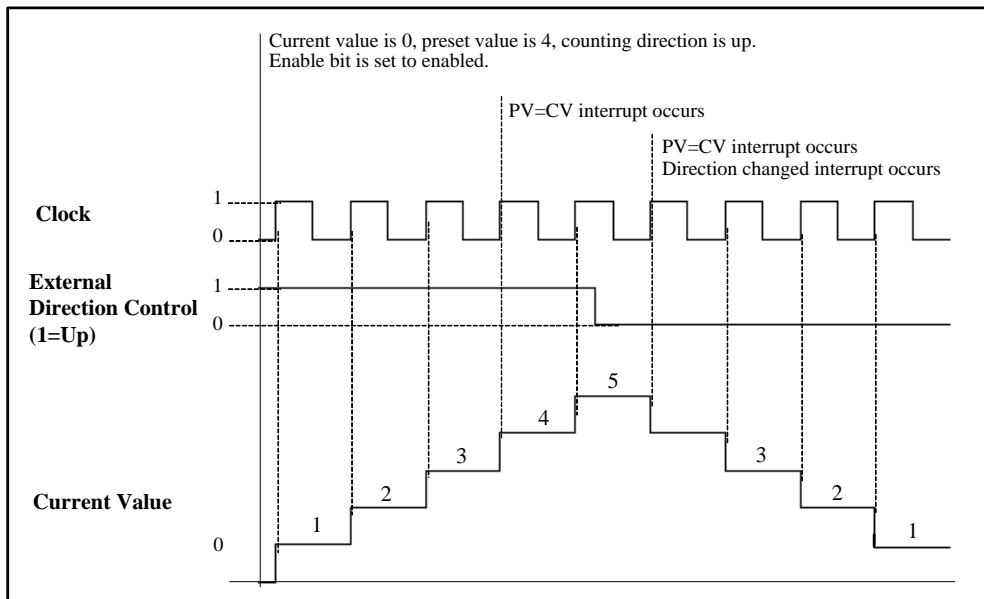


Figure 6-5 Time Sequence of Mode 3, 4 or 5

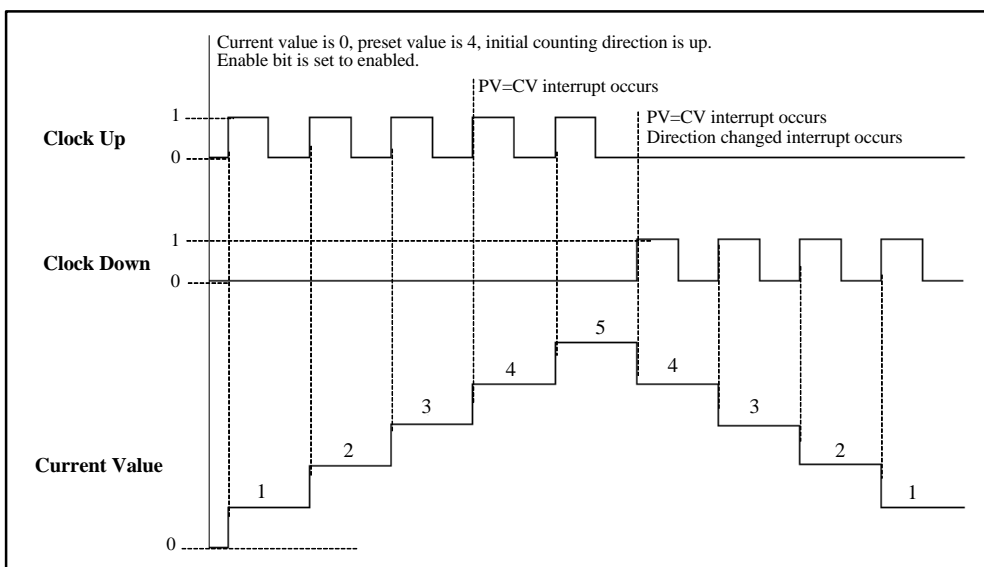


Figure 6-6 Time Sequence of Mode 6, 7 or 8

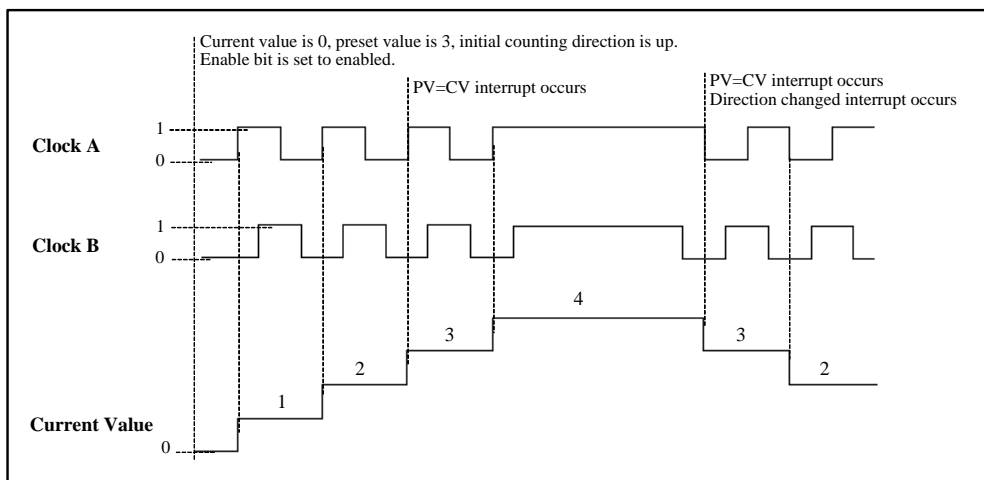


Figure 6-7 Time Sequence of Mode 9, 10 or 11 (Quadrature, 1x rate)

6.13.3.4 Control Byte and Status Byte

➤ Control Byte

In SM area each high-speed counter is assigned control byte to save its configuration data: one control word (8 bit), current value and pre-set (double-integer with 32 bit). Initial value of current assigned value. If the current value is written in the high-speed counter, it will start counting from that value. Please see below:

HSC0	HSC1	Description
SM37.0	SM47.0	Effective electrical level of reset signal: 0=high; 1=low
SM37.1	SM47.1	Effective electrical level to start signal: 0=high; 1=low
SM37.2	SM47.2	Orthogonal counter rate: 0=1x rate; 1=4x rate*
SM37.3	SM47.3	Counting direction 0=minus; 1=plus
SM37.4	SM47.4	Write counting direction in HSC? 0= NO; 1= Yes
SM37.5	SM47.5	Write new pre-set value in HSC? 0= NO; 1= Yes
SM37.6	SM47.6	Write new current value in HSC? 0= NO; 1= Yes
SM37.7	SM47.7	Allow this high-speed counter? 0=NO; 1= YES

HSC0	HSC1	Description
SMD38	SMD48	Current value
SMD42	SMD52	Pre-set value

Only after the high-speed counter and its mode are defined, can the dynamic parameters of the counter be programmed. A control byte is provided for each high-speed counter, and you can operate as follows:

- Enable or disable the HSC
- The counting direction control (limited to mode 0, 1 and 2), or the initial direction of all other modes
- Load the current value
- Load the preset value

The control byte, relevant current value and preset value shall be loaded before executing the *HSC* instruction.

The following table describes each of these control bits.

➤ **Status Byte**

In SM area, each high-speed counter has a status byte, in which some bits indicate the current counting direction and whether the current value is equal to or greater than the preset value. Definition of the status bits for each high-speed counter is shown in the following table.

HSC0	HSC1	Description
SM36.0	SM46.0	Reserved
SM36.1	SM46.1	Reserved
SM36.2	SM46.2	Reserved
SM36.3	SM46.3	Reserved
SM36.4	SM46.4	Reserved
SM36.5	SM46.5	Current counting direction: 0 = Down; 1 = Up
SM36.6	SM46.6	Current value equal to preset value: 0 = Not equal; 1 = Equal
SM36.7	SM46.7	Current value greater than preset value: 0 = Not greater than; 1 = Greater than

➤ **Configuring Current Value and Preset Value**

Each high-speed counter has a 32-bit current value (i.e. starting value) and 32-bit preset value. Either the current value or the preset value is signed double integer. In order to write the new current value and preset value into the high-speed counter, the control byte and the SM bytes that store the current value and/or the preset value must be configured firstly, and then the *HSC* instruction must be executed so that the new values can be written to the high-speed counter. The following table shows the SM bytes that store the new current value and preset value.

	HSC0	HSC1	HSC2	HSC3	HSC4	HSC5
New current value	SMD38	SMD48	SMD58	SMD138	SMD148	SMD158
New preset value	SMD42	SMD52	SMD62	SMD142	SMD152	SMD162

➤ **Accessing the Current Value of a High-Speed Counter**

The current counting value of a high-speed counter is read-only and can be represented only as a double integer (32-bit). The current counting value of a high-speed counter is accessed using the memory type (HC) and the counter number; for example, HC0 represents the current value of HSC0, as shown in the following diagram.

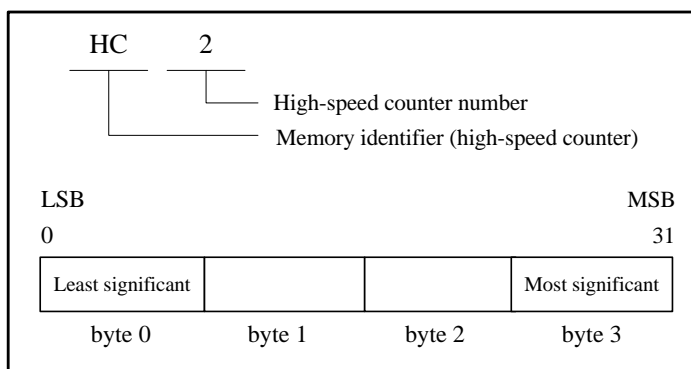


Figure 6-8 Accessing the Current Value of a High-Speed Counter

6.13.3.5 Assigning Interrupts

Each mode supports a PV=CV (the current value equal to the preset value) interrupt. The mode that use an

external reset input supports an External Reset interrupt. The mode that use an external direction control input supports a Direction Changed interrupt. Each of these interrupt conditions can be enabled or disabled separately. Please refer to [6.10.3 Types of Interrupt Events Supported by the Kinco-K5](#) for details.

6.13.3.6 Programming the High-speed Counter

You can program a high-speed counte as follows:

- Assign the control byte.
- Assign the current value (i.e. starting value) and the preset value.
- (Optional) Assign the interrupt routines using the *ATCH* instruction.
- Define the counter and its mode using the *HDEF* instruction.

Note: The *HDEF* instruction can only be executed once for each high-speed counter after the CPU enters RUN mode.

- Start the high-speed counter using the *HSC* instruction.

The following is the detailed introduction for the initialization and operation steps taking HSC0 as an example. You are recommended to make a subroutine that contains the *HDEF* instruction and other initialization instructions and call this subroutine in the main program using SM0.1 to reduce the CPU cycle time.

- **Using HSC**

The following example uses Mode 9. And the other modes take the similar steps.

- In the initialization subroutine, load the desired control status into SMB37.

For example (1x counting rate), SMB37 = b#16#F0 indicates:

- Enable HSC0
- Write a new current value to HSC0
- Write a new preset value to HSC0
- Set the start input and the reset input to be active high

- Load the desired current value (32-bit) into SMD38. If 0 is loaded, SMD38 is cleared.
- Load the desired preset value (32-bit) into SMD42.
- (Optional) Attach the CV = PV event (event 18) to an interrupt routine to respond in real time to a current-value-equal-to-preset-value event.
- (Optional) Attach the direction-changed event (event 17) to an interrupt routine to respond in real time to a direction-changed event.
- (Optional) Attach the external reset event (event 16) to an interrupt routine to respond in real time to an external reset event
- Execute the *HDEF* instruction with the *HSC* input set to be 0 and the *MODE* input set to 9.
- Execute the *HSC* instruction to cause the CPU to configure HSC0 and start it.

➤ **Change the Counting Direction in Mode 0, 1 and 2:**

The following introduces how to change the direction of HSC0 (Mode 0, 1 and 2).

- Load the desired control status into SMB37:
SMB37 = b#16#90: Enable the counter,
Set the new direction to be down-counter
- Execute the *HSC* instruction to cause the CPU to configure HSC0 and start it.

➤ **Load the new current value (in all the modes)**

The following introduces how to change the current value (i.e. starting value) of HSC0.

- Load the desired control status into SMB37:
SMB37 = b#16#C0 Enable the counter
Allow writing the new current value to HSC0.
- Load the desired current value into SMD38. If 0 is loaded, SMD38 is cleared.
- Execute the *HSC* instruction to cause the CPU to configure HSC0 and start it.

➤ **Load the new preset value (in all the modes)**

The following introduces how to change the preset value of HSC0.

- Load the desired control status into SMB37:

SMB37 = b#16#A0 Enable the counter

Allow writing the new preset value to HSC0.

- Load the desired preset value into SMD42.
➤ Execute the *HSC* instruction to cause the CPU to configure HSC0 and start it.

➤ **Disable the High-speed Counter (in all modes)**

The following introduces how to disable HSC0.

- Load the desired control status into SMB37:

SMB37 = b#16#00 Disable the counter;

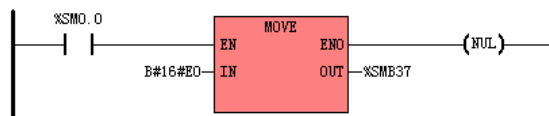
- Execute the *HSC* instruction to cause the CPU to disable the counter.

6.13.3.7 Examples

The following example also uses HSC0.

The initialization subroutine: Initialize

(* Network 0 *)
 (* 1x counting rate; Enable HSC0; Allow updating current value and preset value;
 Set the start input and the reset input to be active high *)

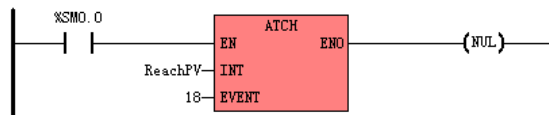


(* Network 1 *)
 (* Set the new current value and new preset value *)

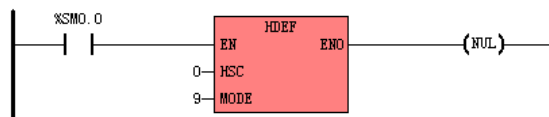


LD

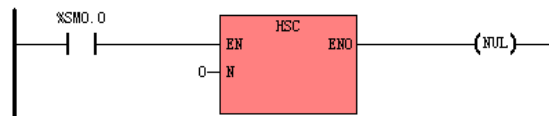
(* Network 2 *)
 (* Attach the CU = PU event (event 18) to ReachPV interrupt routine *)



(* Network 3 *)
 (* Define HSC0 to be in mode 9 *)

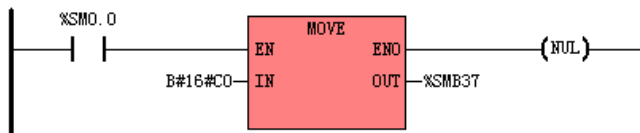


(* Network 4 *)
 (* Configure and start HSC0 *)

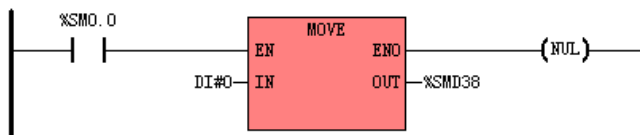


The interrupt routine: ReachPV

(* Network 0 *)
 (* Allow updating current value *)

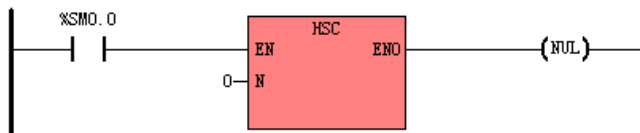


(* Network 1 *)
 (* Set the new current value to be 0 to re-count *)



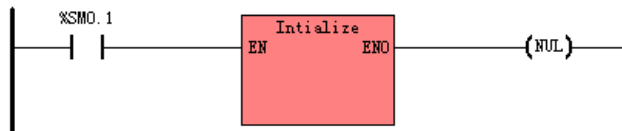
LD

(* Network 2 *)
 (* Configure and restart HSC0 *)



Main program:

(* Network 0 *)
 (* Call Initialize subroutine *)



The initialization subroutine: Initialize

(* Network 0 *)

(* 1x counting rate; Enable HSC0; Allow updating current value AND preset value; *)

(* Set the start input and the reset input to be active high *)

```
LD      %SM0.0
MOVE    B#16#E0, %SMB37
```

(* Network 1 *)

(*Set the new current value and new preset value*)

```
LD      %SM0.0
MOVE    DI#0, %SMD38
MOVE    DI#100, %SMD42
```

(* Network 2 *)

IL (*Attach the CV = PV event (event 18) to ReachPV interrupt routine*)

```
LD      %SM0.0
ATCH    ReachPV, 18
```

(* Network 3 *)

(*Define HSC0 to be in mode 9*)

```
LD      %SM0.0
HDEF    0, 9
```

(* Network 4 *)

(*Configure and start HSC0*)

```
LD      %SM0.0
HSC     0
```


The interrupt routine: ReachPV

(* Network 0 *)

(*Allow updating current value*)

LD %SM0.0

MOVE B#16#C0, %SMB37

(* Network 1 *)

(*Set the new current value to be 0 to re-count*)

LD %SM0.0

MOVE DI#0, %SMD38

(* Network 2 *)

(*Configure and restart HSC0*)

LD %SM0.0

IL HSC 0

Main program:

(* Network 0 *)

(*Call Initialize subroutine*)

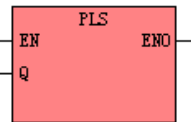
LD %SM0.1

CAL Intialize

6.13.4 High-speed Pulse Output Instructions

Here the high-speed pulse output means the Pulse Train Output (PTO) or the Pulse-Width Modulation (PWM). K5 provides two PWM impulse generator to output PWM, using tunnel of Q0.0 and Q0.1, called PWM0 and PWM1. The generator and DO image register share the same address Q0.0 and Q0.1. If PWM is started in Q0.0 or Q0.1, the generator will control the output tunnel and forbid the output of general function. When PWM function is forbidden, Q0.0 and Q0.1 will be controlled by DO image register. The highest output rate of K504 is 20kHz and other CPU modules is 200kHz.

➤ Description

	Name	Usage	Group	
LD	PLS			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	PLS	PLS <i>Q</i>	U	

Operands	Input/Output	Data Type	Description
Q	Input	INT	Constant (0 or 1)

The *PLS* instruction is used to load the corresponding configurations of the PTO/PWM specified by *Q* from the specified SM registers and then operate the PTO/PWM generator accordingly.

In LD, the *EN* input decides whether to execute the *PLS* instruction.

In IL, CR value decides whether to execute the *PLS* instructions. It won't influence CR.

6.13.4.1 Configuring and Controlling the PWM Operation

Each PWM generator is provided with some registers in SM area to store its configurations or indicate its status. The characteristics of a PWM waveform can be changed by modifying the corresponding SM registers and then executing the *PLS* instruction. The following table describes control registers in detail.

Q0.0	Q0.1	Description
SM67.0	SM77.0	Whether to update the cycle time: 0 = not update; 1 = update
SM67.1	SM77.1	Whether to update pulse width time: 0 = not update; 1 = update
SM67.2	SM77.2	Reserve
SM67.3	SM77.3	Time base: 0 = 1 μ s; 1 = 1ms
SM67.4	SM77.4	Reserve
SM67.5	SM77.5	Reserve
SM67.6	SM77.6	1 = PWM
SM67.7	SM77.7	Enable: 0 = disable; 1 = enable
SMW68	SMW78	Cycle time value, Range: 2 to 65535
SMW70	SMW80	Pulse width value, Range: 0 to 65535

The following table describes the status bits of the PTO/PWM generators.

Q0.0	Q0.1	Status Bits
SM66.4	SM76.4	PTO profile terminated due to increment calculation error: 0 = no error; 1 = terminated
SM66.5	SM76.5	PTO profile terminated due to user command: 0 = not terminated; 1 = terminated
SM66.6	SM76.6	PTO pipeline overflow/underflow 0 = no; 1 = overflow/underflow
SM66.7	SM76.7	PTO idle 0 = in progress; 1 = idle

The PTO Idle bit (SM66.7 or SM76.7) indicates the completion of the pulse train output. Besides, as soon as the pulse train is completed, the corresponding interrupt routine is invoked. If the multi-segment operation is being used, the interrupt routine is invoked as soon as the profile table is completed.

6.13.4.2 PWM Operations

The following takes PWM0 as an example to introduce how to configure and operate the PTO/PWM generator in the user programme.

➤ **Initializing the PWM Output**

Use SM0.1 (the first scan memory bit) to call a subroutine that contains the initialization instructions. Since SM0.1 is used, the subroutine shall be invoked only once, and this reduces scan time and provides a better program structure.

The following steps describes how to configure PWM0 in the initialization subroutine:

➤ Load the desired control status into SMB67:

For example, SMB67 = B#16#D3 indicates

- ◆ Enable the PWM function
 - ◆ Select PWM operation
 - ◆ Select 1µs as the time base
 - ◆ Allow updating the pulse width value and cycle time value
 - ◆ Setlect synchronous update method
- Load the cycle time value into SMW68.
- Load the pulse width value into SMW70.
- Execute the *PLS* struction to cause the CPU to configure PWM0 and start it.

■ **Changing the Pulse Width for the PWM Output**

The following steps describes how to change PWM output pulse width (assume that SMB67 has been preloaded with B#16#D2 or B#16#DA.):

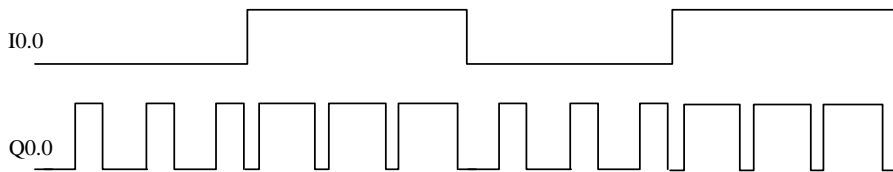
- Load the pulse width value (16-bit) into SMW70.
- Execute the *PLS* struction to cause the CPU to configure PWM and start it.

6.13.4.3 Example

➤ PWM

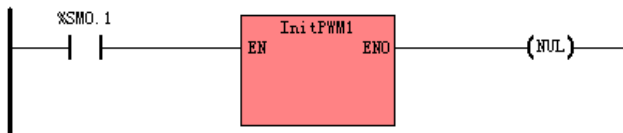
PWM1 (output through Q0.1) is used in the example.

If I0.0 is false, change the pulth width to 40% duty cycle; if I0.0 is true, change the pulth width to 40% duty cycle. The time sequence diagram is shown as follows:



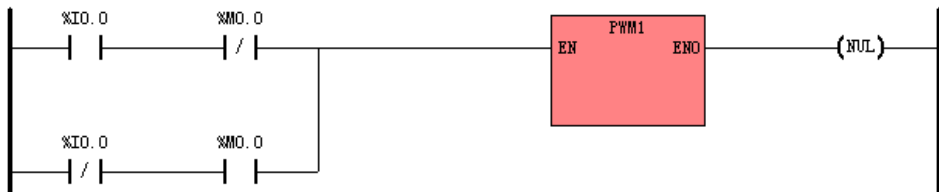
MAIN Program:

```
(* Network 0 *)
(* Use SM0.1 to call subroutine InitPWM1 to initialize PWM1 *)
```

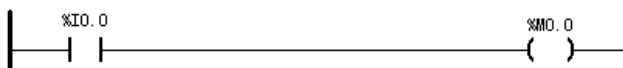


```
(* Network 1 *)
(* If the status of I0.0 changes,
subroutine PWM1 shall be called to change the pulse width. *)
```

LD

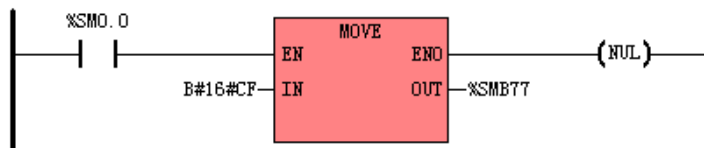


```
(* Network 2 *)
```

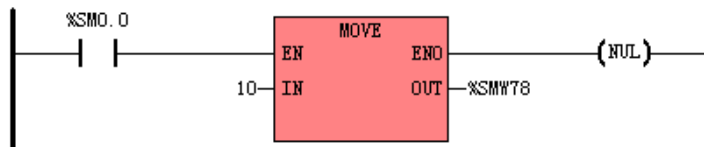


Subroutine *InitPWM1*:

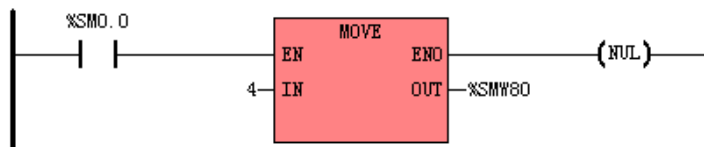
(* Network 0 *)
 (* Select PWM1; Select 1ms as the time base;
 Allow updating the cycle time value and the pulth width *)



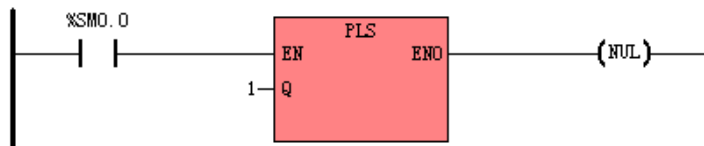
(* Network 1 *)
 (* Set the cycle time of PWM1 to be 10ms *)



(* Network 2 *)
 (* Set the pulse width of PWM1 to be 4ms *)



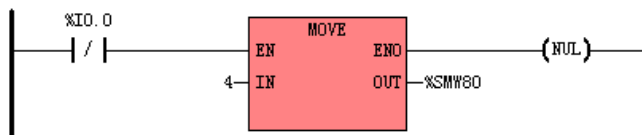
(* Network 3 *)
 (* Execute PWM1 *)



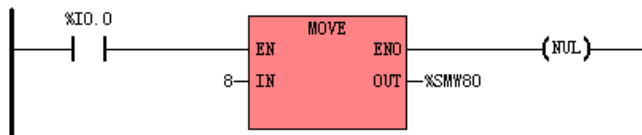
LD

Subroutine *PWM1*:

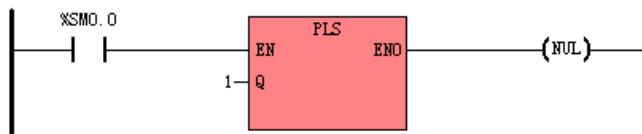
(* Network 0 *)
 (* IF I0.0 is false, the pulse width of PWM1 is set to be 4ms *)



(* Network 1 *)
 (* IF I0.0 is true, the pulse width of PWM1 is set to be 8ms *)



(* Network 2 *)
 (* Execute PWM1 *)



LD

IL	<p>MAIN Program:</p> <p>(* Network 0 *)</p> <p>(* Use SM0.1 to call subroutine InitPWM1 to initialize PWM1 *)</p> <p>LD %SM0.1</p> <p>CAL InitPWM1</p> <p>(* Network 1 *)</p> <p>(* If the status of IO.0 changes, subroutine PWM1 shall be called to change the pulse width. *)</p> <p>LD %IO.0</p> <p>ANDN %M0.0</p> <p>OR(</p> <p>LDN %IO.0</p> <p>AND %M0.0</p> <p>)</p> <p>CAL PWM1</p> <p>(* Network 2 *)</p> <p>LD %IO.0</p> <p>ST %M0.0</p>
-----------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

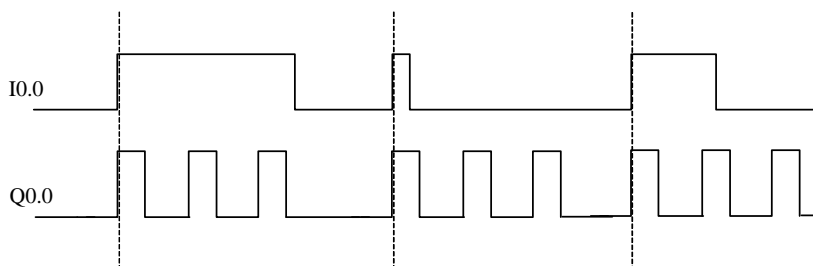
IL	<p>Subroutin <i>InitPWM1</i>:</p> <p>(* Network 0 *)</p> <p>(*Select PWM1; Select 1ms as the time base; Allow updating the cycle time value and the pulth width*)</p> <pre>LD %SM0.0 MOVE B#16#CF, %SMB77</pre> <p>(* Network 1 *)</p> <p>(*Set the cycle time of PWM1 to be 10ms*)</p> <pre>LD %SM0.0 MOVE 10, %SMW78</pre> <p>(* Network 2 *)</p> <p>(*Set the pulse width of PWM1 to be 4ms*)</p> <pre>LD %SM0.0 MOVE 4, %SMW80</pre> <p>(* Network 3 *)</p> <p>(*Execute PWM1*)</p> <pre>LD %SM0.0 PLS 1</pre>
	<p>Subroutin <i>PWM1</i>:</p> <p>(* Network 0 *)</p> <p>(*If I0.0 is false, the pulse width of PWM1 is set to be 4ms*)</p> <pre>LDN %I0.0 MOVE 4, %SMW80</pre> <p>(* Network 1 *)</p> <p>(*If I0.0 is true, the pulse width of PWM1 is set to be 8ms*)</p> <pre>LD %I0.0 MOVE 8, %SMW80</pre> <p>(* Network 2 *)</p> <p>(*Execute PWM1*)</p> <pre>LD %SM0.0 PLS 1</pre>

➤ **PTO operation (Single-Segment)**

PTO0 (output through Q0.0) is used in the example.

Start PTO0 and output 3 pulses every time on the rising edge of I0.0.

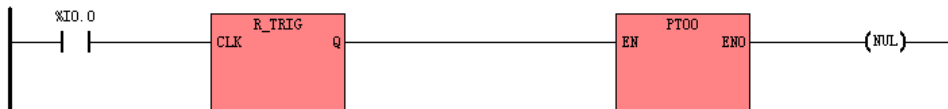
The time sequence diagram is shown as follows:



MAIN Program:

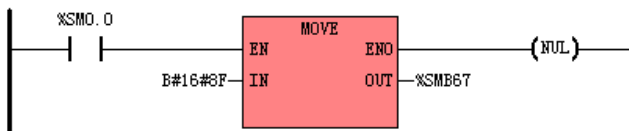
LD

(* Network 0 *)
(* Start PTO0 on the rising edge of I0.0 *)

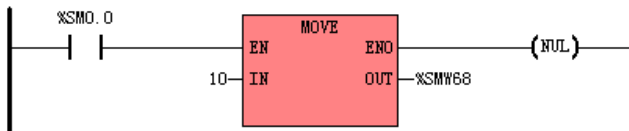


Subprogram *PT00*:

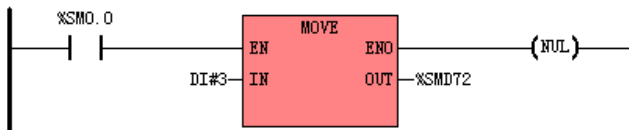
(* Network 0 *)
 (* Select a single-segment operation for PT00;
 Select 1ms as the time base; Allow updating the cycle time and the pulse count *)



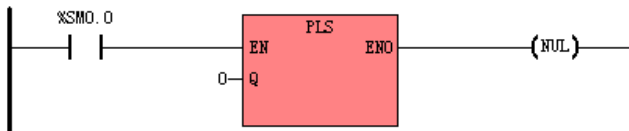
(* Network 1 *)
 (* Set the cycle time to be 10ms *)



(* Network 2 *)
 (* Set the pulse count to be 3 pulses *)



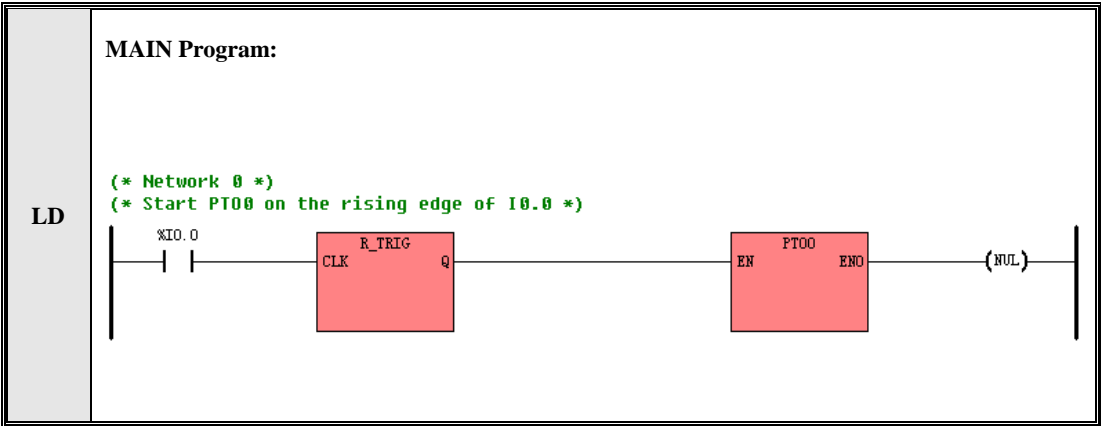
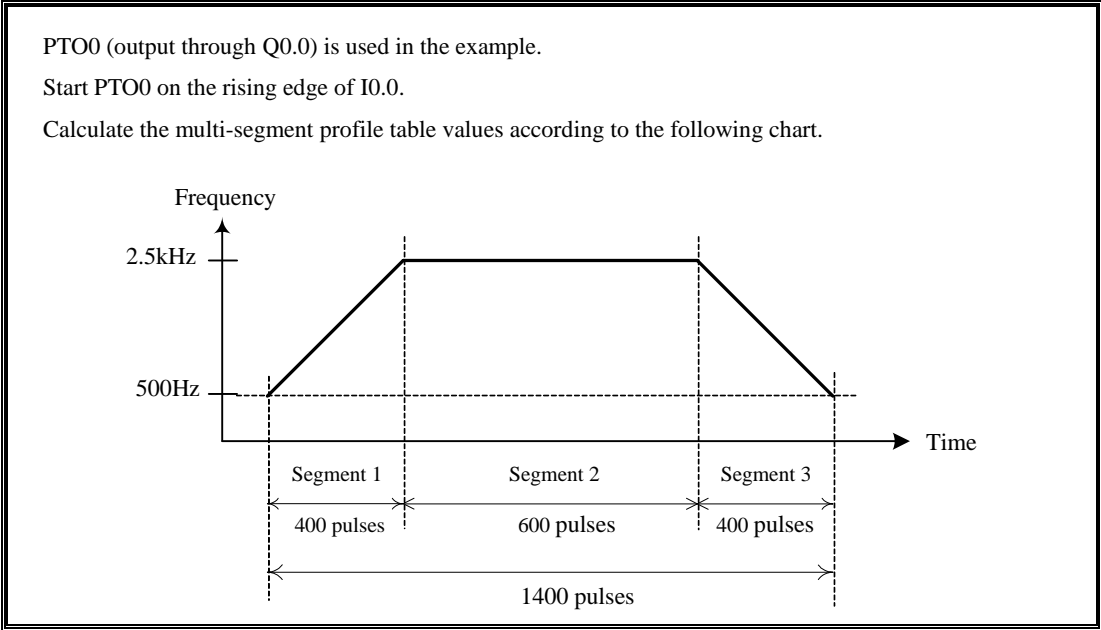
(* Network 3 *)
 (* Execute PT00 *)



LD

IL	<p>MAIN Pogram:</p> <p>(* Network 0 *)</p> <p>(* Start PTO0 on the rising edge of I0.0 *)</p> <pre>LD %I0.0 R_TRIG CAL PTO0</pre>
	<p>Subprogram PTO0:</p> <p>(* Network 0 *)</p> <p>(* Select a single-segment operation for PTO0; *)</p> <p>(* Select 1ms as the time base; Allow updating the cycle time and the pulse count *)</p> <pre>LD %SM0.0 MOVE B#16#8F, %SMB67</pre> <p>(* Network 1 *)</p> <p>(* Set the cycle time to be 10ms *)</p> <pre>LD %SM0.0 MOVE 10, %SMW68</pre> <p>(* Network 2 *)</p> <p>(* Set the pulse count to be 3 pulses *)</p> <pre>LD %SM0.0 MOVE DI#3, %SMD72</pre> <p>(* Network 3 *)</p> <p>(* Execute PTO0 *)</p> <pre>LD %SM0.0 PLS 0</pre>

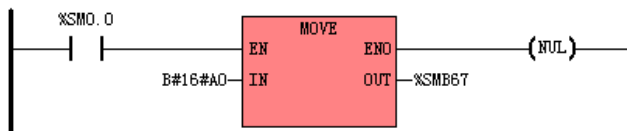
➤ **PTO operation (Multi-Segment)**



Subroutine *PT00*:

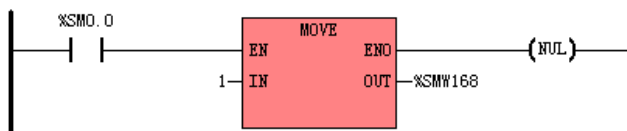
(* Network 0 *)

(* Enable PT00; Select multi-segment operation; Set the time base to be 1us *)



(* Network 1 *)

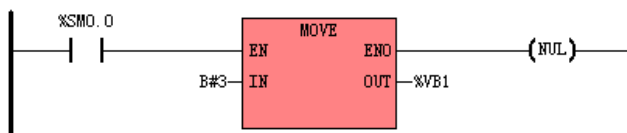
(* Use VB1 as the starting position of the profile table *)



LD

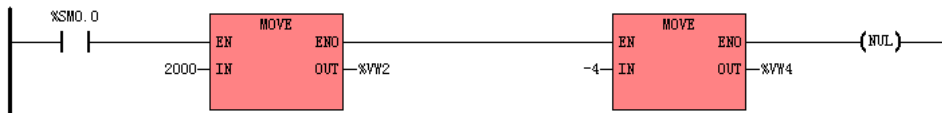
(* Network 2 *)

(* Set the number of segments to be 3 *)



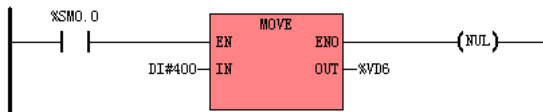
(* Network 3 *)

(* Segment 1: Set the initial cycle time to 2000us, set the cycle time increment to -4us *)



(* Network 4 *)

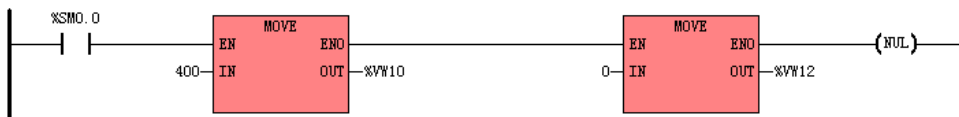
(* Segment 1: Set the number of pulses to 400 *)



Subroutine *PT00*: (Continued)

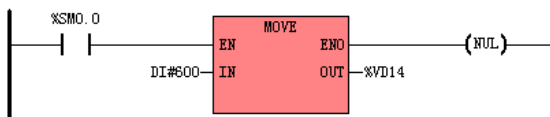
(* Network 5 *)

(* Segment 2: Set the initial cycle time to 400us, set the cycle time increment to 0 *)



(* Network 6 *)

(* Segment 2: Set the number of pulses to 600 *)



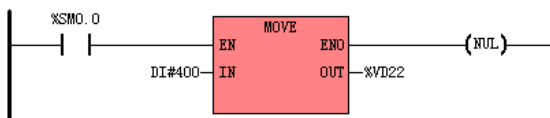
(* Network 7 *)

(* Segment 3: Set the initial cycle time to 400us, set the cycle time increment to 4us *)



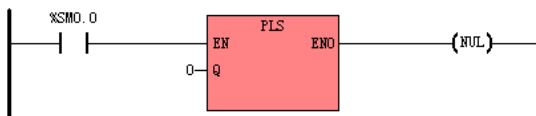
(* Network 8 *)

(* Segment 3: Set the number of pulses to 400 *)



(* Network 9 *)

(* Execut PT00 *)



LD

IL	<p>MAIN Program:</p> <p>(* Network 0 *)</p> <p>LD %I0.0</p> <p>R_TRIG</p> <p>CAL PTO0 (* Start PTO0 on the rising edge of I0.0 *)</p>
	<p>Subroutine PTO0:</p> <p>(* NETWORK 0 *)</p> <p>LD %SM0.0</p> <p>MOVE B#16#A0, %SMB67 (* Enable PTO0; Select multi-segment operation; Set the time base to be 1us *)</p> <p>MOVE 1, %SMW168 (* Use VB1 as the starting position of the profile table *)</p> <p>MOVE B#16#03, %VB1 (* Set the number of segments to be 3 *)</p> <p>(* Segment 1 *)</p> <p>MOVE 2000, %VW2 (* Set the initial cycle time to 2000us *)</p> <p>MOVE -4, %VW4 (* Set the cycle time increment to -4us *)</p> <p>MOVE DI#400, %VD6 (* Set the number of pulses to 400 *)</p> <p>(* Segment 2 *)</p> <p>MOVE 400, %VW10 (* Set the initial cycle time to 400us *)</p> <p>MOVE 0, %VW12 (* Set the cycle time increment to 0 *)</p> <p>MOVE DI#600, %VD14 (* Set the number of pulses to 600 *)</p> <p>(* Segment 3 *)</p> <p>MOVE 400, %VW18 (* Set the initial cycle time to 400us *)</p> <p>MOVE 4, %VW20 (* Set the cycle time increment to 4us *)</p> <p>MOVE DI#400, %VD22 (* Set the number of pulses to 400 *)</p> <p>PLS 0 (* Execute PTO0 *)</p>

6.14 Timers

Timer is one of the function blocks defined in the IEC61131-3 standard, totally in three types i.e. TON, TOF and TP. Please refer to [3.6.5 Function Block and Function Block Instance](#) for more detailed information.

6.14.1 The resolution of the timer

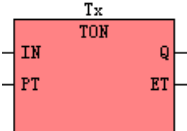
There are three resolutions for timers. The timer number determines the resolution as shown in the table.

The preset value and the current value of a timer are all multiples of this timer's resolution, for example, a value of 100 on a 10-ms timer represents 1000ms.

PLC will update the timing value of the timer only when execute the timer command. It will be influenced by the scan cycle.

6.14.2 TON (On-delay Timer)

➤ Description

	Name	Usage	Group	
LD	TON			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	TON	TON Tx, PT	P	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>Tx</i>	-	Timer instance	T
<i>IN</i>	Input	BOOL	Power flow
<i>PT</i>	Input	INT	I, AI, AQ, M, V, L, SM, constant
<i>Q</i>	Output	BOOL	Power flow
<i>ET</i>	Output	INT	Q, M, V, L, SM, AQ

T_x is an instance of TON function block.

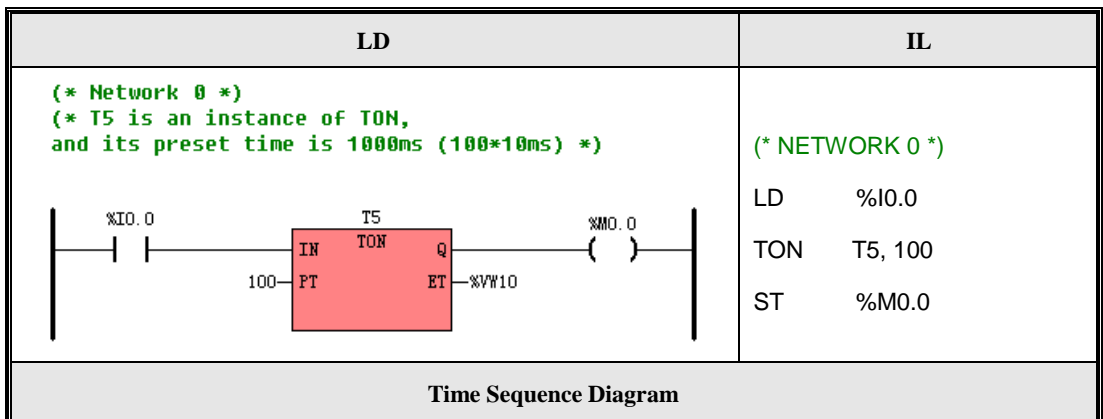
■ **LD**

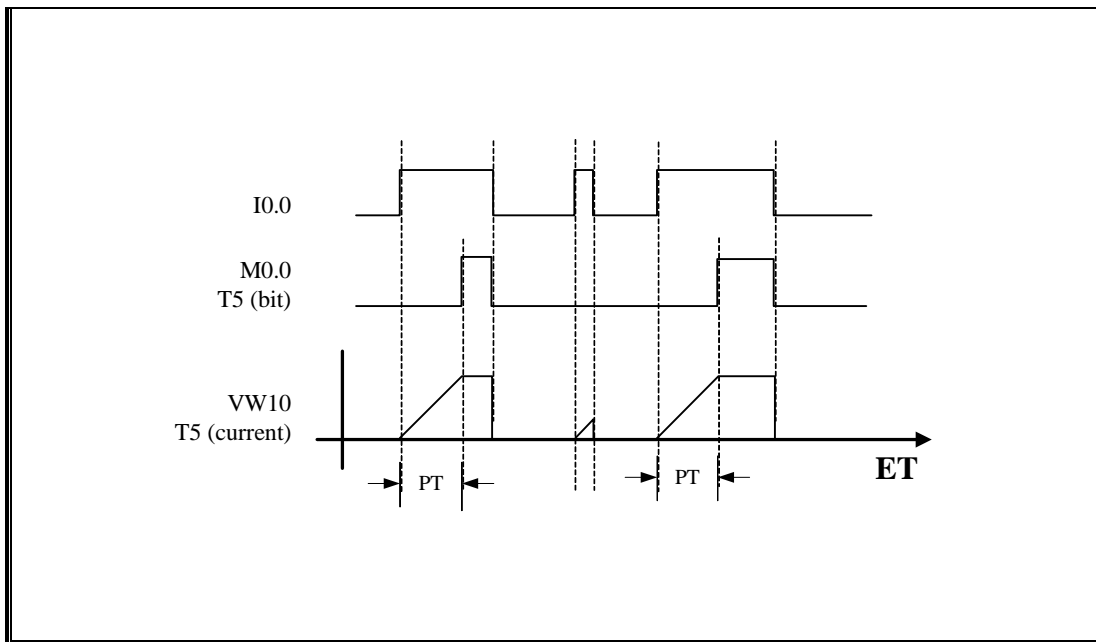
T_x starts to time on the rising edge of the IN input. When the elapsed time (i.e. the current value) ET is greater than or equal to the preset time PT , both the Q output and the status bit of T_x are set to be TRUE. If the IN input turns to FALSE, T_x is reset, and both the Q output and its status bit value are set to be FALSE, meanwhile its current value is cleared to 0.

■ **IL**

T_x starts to time on the rising edge of CR . When the current value is greater than or equal to the preset value PT , the status bit of T_x is set to be TRUE. If CR turns to FALSE, T_x is reset, and its status bit is set to be FALSE, meanwhile its current value is cleared to 0. After each scan, CR is set to be the status bit value of T_x .

➤ Examples





6.14.3 TOF (Off-delay Timer)

➤ Description

	Name	Usage	Group	
LD	TOF	<p>The diagram shows a red rectangular symbol for the TOF (Off-delay Timer). It has four terminals: IN on the left, PT on the bottom left, Q on the right, and ET on the bottom right. The symbol is labeled 'Tx' at the top and 'TOF' in the center.</p>		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	TOF	TOF Tx, PT	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>Tx</i>	-	Timer instance	T
<i>IN</i>	Input	BOOL	Power flow
<i>PT</i>	Input	INT	I, AI, AQ, M, V, L, SM, constant
<i>Q</i>	Output	BOOL	Power flow
<i>ET</i>	Output	INT	Q, M, V, L, SM, AQ

Tx is an instance of TOF function block.

■ **LD**

Tx starts to time on the falling edge of the *IN* input. When the elapsed time (i.e. the current value) *ET* is greater than or equal to the preset time *PT*, both the *Q* output and the status bit of *Tx* are set to be FALSE. If the *IN* input turns to TRUE, *Tx* is reset, and both the *Q* output and its status bit are set to be TRUE, meanwhile its current value is cleared to 0.

■ **IL**

Tx starts to time on the falling edge of CR. When the current value is greater than or equal to the preset value *PT*, the status bit of *Tx* is set to be FALSE. If CR turns to TRUE, *Tx* is reset, and its status bit is set to be TRUE, meanwhile its current value is cleared to 0. After each scan, CR is set to be the status bit value of *Tx*.

➤ Examples

LD	IL
-----------	-----------

(* Network 0 *)
 (* T5 is an instance of TOF,
 and its preset time is 1000ms (100*10ms) *)

(* NETWORK 0 *)

LD	%I0.0
TOF	T5, 100
ST	%M0.0

Time Sequence Diagram

6.14.4 TP (Pulse Timer)

➤ Description

	Name	Usage	Group	
				<input checked="" type="checkbox"/> CPU504

LD	TP		<input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	TP	TP Tx, PT	P

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>Tx</i>	-	Timer instance	T
<i>IN</i>	Input	BOOL	Power flow
<i>PT</i>	Input	INT	I, AI, AQ, M, V, L, SM, constant
<i>Q</i>	Output	BOOL	Power flow
<i>ET</i>	Output	INT	Q, M, V, L, SM, AQ

Tx is an instance of TP function block. The *TP* instruction is used to generate a pulse for the preset time.

■ LD

On the rising edge of the *IN* input, *Tx* starts to time, and both the *Q* output and the status bit of *Tx* are set to be TRUE. The *Q* output and the status bit remain TRUE within the preset time *PT*. As soon as the elapsed time (i.e. the current value) *ET* reaches the *PT*, both the *Q* output and the status bit become FALSE.

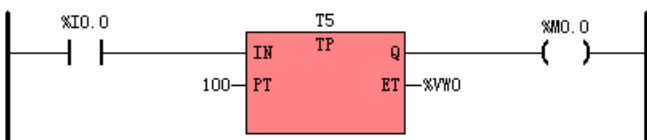
■ IL

On the rising edge of *CR*, *Tx* starts to time, and the status bit of *Tx* is set to be TRUE. The status bit remains TRUE within the preset time *PT*. As soon as the current value reaches the *PT*, the status bit becomes FALSE. After each scan, *CR* is set to be the status bit value of *Tx*.

➤ Examples

LD	IL
-----------	-----------

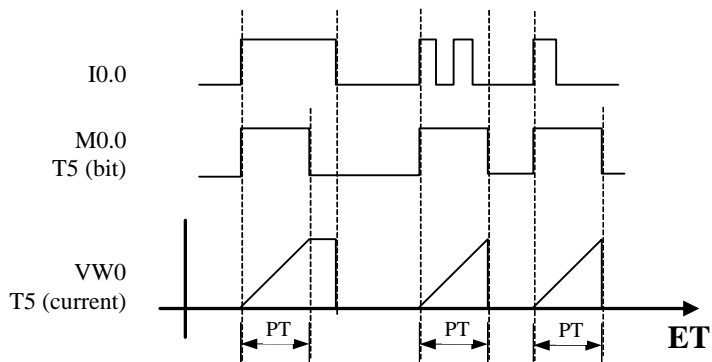
(* Network 0 *)
 (* T5 is an instance of TP,
 and its preset time is 1000ms (100*10ms) *)



(* NETWORK 0 *)

LD %IO.0
 TP T5, 100
 ST %MO.0

Time Sequence Diagram



6.15 PID

PID instruction is provided in KINCO-K5, and the position algorithm is adopted. You can use it as PID fixed set point controller with continuous input and output, and you can use up to 8 PID loops in one CPU.

➤ Description

	Name	Usage	Group	
LD	PID	<div style="border: 1px solid black; background-color: #f0f0f0; padding: 5px; width: fit-content; margin: 0 auto;"> <p style="text-align: center;">PID</p> <p>EN ENO</p> <p>AUTO XOUT</p> <p>PV XOUTP</p> <p>SP</p> <p>XO</p> <p>KP</p> <p>TR</p> <p>TD</p> <p>PV_H</p> <p>PV_L</p> <p>XOUTP_H</p> <p>XOUTP_L</p> <p>CYCLE</p> </div>		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	PID	PID AUTO, PV, SP, XO, KP, TR, TD, PV_H, PV_L, XOUTP_H, XOUTP_L, CYCLE, XOUT, XOUTP	U	

Operands	IN/OUT	Data Type	Memory Areas	Comment
<i>AUTO</i>	INPUT	BOOL	I, Q, V, M, SM, L, T, C	Manual/Auto. 0=Manual, 1=Auto.
<i>PV</i>	INPUT	INT	AI, V, M, L	Process Variable
<i>SP</i>	INPUT	INT	V	Setpoint
<i>XO</i>	INPUT	REAL	V	Manual value, range [0.0, 1.0]
<i>KP</i>	INPUT	REAL	V	Proportionality constant
<i>TR</i>	INPUT	REAL	V	Reset time, which determines the time response of the integrator. (Unit: s)
<i>TD</i>	INPUT	REAL	V	Derivative time, which determines the time response of the derivative unit. (Unit: s)
<i>PV_H</i>	INPUT	INT	V	The upper limit value of <i>PV</i>
<i>PV_L</i>	INPUT	INT	V	The lower limit value of <i>PV</i>
<i>XOUTP_H</i>	INPUT	INT	V	The upper limit value of <i>XOUTP</i>
<i>XOUTP_L</i>	INPUT	INT	V	The lower limit value of <i>XOUTP</i>
<i>CYCLE</i>	INPUT	DINT	V	Sampling period. (Unit: ms)
<i>XOUT</i>	OUTPUT	REAL:	V	Manipulated Value, range [0.0, 1.0].
<i>XOUTP</i>	OUTPUT	INT	AQ, V	Manipulated Value Peripheral. This value is the normalizing result of <i>XOUT</i> .

➤ **LD**

If *EN* is 1, this instruction is executed.

➤ **IL**

If *EN* is 1, this instruction is executed, and it does not influence CR.

➤ Other information

➤ **Manual/Auto**

It is possible to switch between a manual and an automatic mode with the help of *Auto* input.

If *Auto* is 0, then the PID is in the manual mode, and now the value of *XO* input shall be directly set as the manipulated value (*XOUT*).

If *Auto* is 1, then the PID is in the automatic mode, and now it shall execute the PID calculations according to the inputs and set the final result as the manipulated value (*XOUT*).

➤ **Normalizing the *PV* and *SP***

The *PV* and *SP* can be input in the peripheral format (an integer). But PID algorithm needs a floating-point value of 0.0 to 1.0, so normalization is needed.

The Kinco-K5 automatically finishes the normalization according to the *PV*, *SP*, *PV_H* and *PV_L* input. You may assign any linear correlation values of them, but the inputs must be the same dimension.

For example, you want to control the pressure to the expected value 25MPa. A pressure transmitter is used to measure the pressure, and the transformer's measuring range is 0-40MPa and its output range is 4-20mA. The transformer's output is connected to a channel of an AI module, and this channel is configured as the following: the address is AIW0, and the measured type is '4-20mA' whose the measured value is '4000-20000'. Now, you can assign the following values to the PID inputs:

	Actual Parameter	Comment
<i>PV</i>	AIW0	AIW0 can be set as <i>PV</i> because of their linear relation.
<i>SP</i>	14000	14mA. Because 14mA means the real pressure value 25MPa.
<i>PV_L</i>	4000	The lower limit value of the transformer's output
<i>PV_H</i>	20000	The upper limit value of the transformer's output

➤ **Manipulated Values**

This PID has two manipulated values: *XOUT* and *XOUTP*.

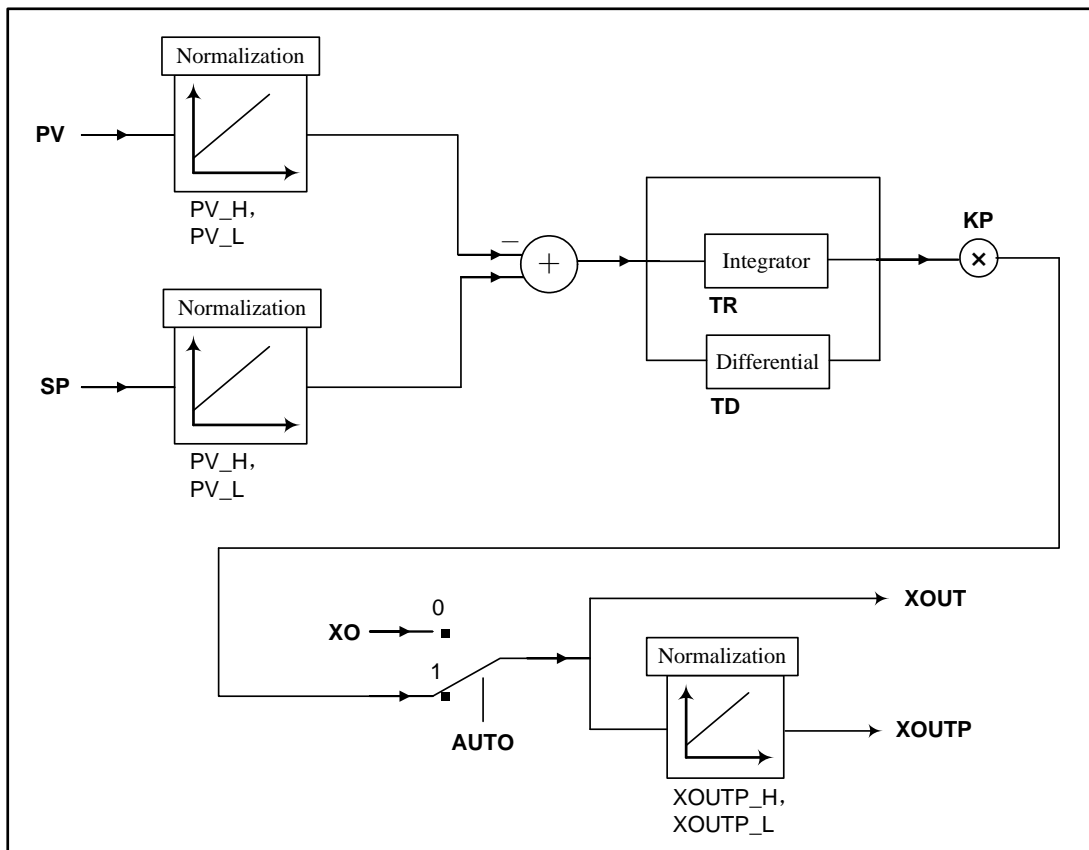
XOUT is a value between 0.0 and 1.0 (that is between 0.0 and 100.0%).

XOUTP is an integer value with the user-defined peripheral format, and it is the result of normalizing *XOUT* according to the *XOUTP_H* and *XOUTP_L* input:

$$XOUTP = (XOUTP_H - XOUTP_L) * XOUT + XOUTP_L$$

It is convenient for the user to transfer *XOUT_P* to an AO channel.

➤ **PID Diagram**



➤ **Example**

In this example, we build a control system, and the system will maintain a constant temperature inside a box.

IL	(* Network 0 *)		
	(* At first, enter the actual parameters *)		
	LD	%SM0.0	
	MOVE	7200, %VW0	(* SP *)
	MOVE	4000, %VW2	(* PV_L *)
	MOVE	20000, %VW4	(* PV_H *)
	MOVE	4000, %VW6	(* XOUTP_L *)
	MOVE	20000, %VW8	(* XOUTP_H *)
	(* Network 1 *)		
	(* Execute PID *)		
	LD	%SM0.0	
	PID	%M0.0, %AIW0, %VW0, %VR100, %VR104, %VR108, %VR112, %VW2, %VW4, %VW6, %VW8, %VD10, %VR116, %AQW0	

6.16 Position Control

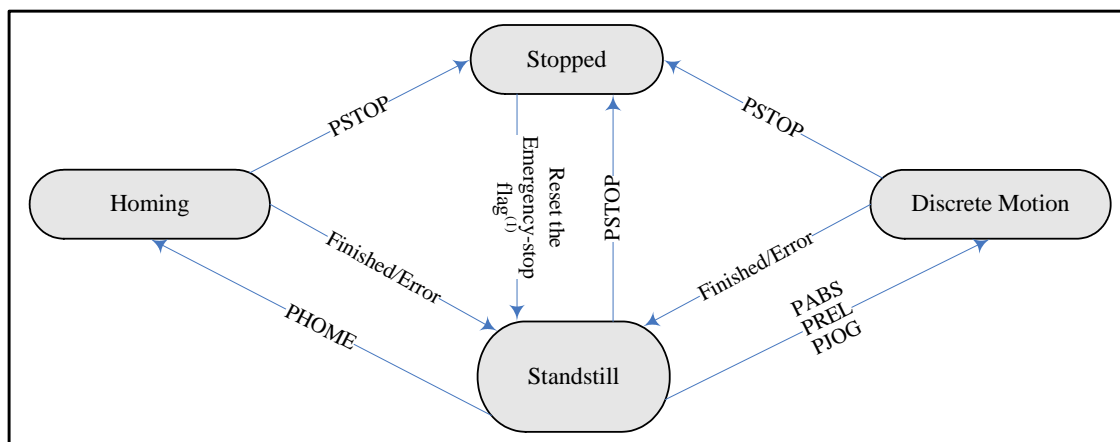
The Kinco-K5 provides 2 high-speed pulse output channels: Q0.0 and Q0.1, and can be used for position control for 2 axes. In [6.13.4 High-speed Pulse Output Instructions](#), the usage of PWM and the PLS instruction is described detailedly.

The Position Control instructions described in this chapter is another usage of the high-speed pulse output function. Comparing with the PLS instruction, the Position Control instructions are more convenient for the position control applications. Similarly, the frequency of the pulse output is between 20kHz-200kHz.

6.16.1 Model

The following diagram is focused on a single axis, and it normatively defines the behavior of the axis at a high level when the position control instructions are activated. The basic rule is that position commands are always taken sequentially.

The axis is always in one of the defined state (see diagram below). Any position command is a transition that changes the state of the axis and, as a consequence, modifies the way the current position is computed.



- The Emergency-Stop flag is SM201.7/ SM231.7. It will be set to 1 automatically while executing the PSTOP instruction. Please refer to the detailed description in the following section.

6.16.2 The correlative variables

6.16.2.1 The direction output channel

For the Position Control instructions, the Kinco-K5 specifies a direction output channel for each high-speed pulse output channel, and a control bit in the SM area to enable the direction output. Please see the following table.

High-speed Pulse Output Channel	Q0.0	Q0.1
Direction output channel	Q0.2	Q0.3
Direction control bit	SM201.3	SM231.3

The direction output channel is used for providing a direction signal which controls the direction of the electric motors: 0 means rotating forwards, and 1 means rotating backwards.

The direction control bit is used to disable or enable the corresponding direction output channel. The direction control bit has the highest priority. If disabled, no direction signal will be provided while executing a position control instruction, and the corresponding direction output channel can be used as a normal DO point.

6.16.2.2 The Status and Control Registers

For the Position Control instructions, the Kinco-K5 specifies a control byte for each high-speed output channel to store its configurations.

A status register is also specified for storing the current value (the number of pulses output, DINT). The current value increases when rotating forwards, and decreases when rotating backwards. The following table describes these registers detailedly. Note: After a position control instruction has finished, the current value will not be cleared automatically, and you can clear it in your program.

The following table describes the control byte and the current value.

Q0.0	Q0.1	Description
SM201.7	SM231.7	Emergency-Stop flag. If this bit is 1, no position control instructions can be executed. When executing the PSTOP instruction, this bit is set to 1 automatically, and it must be reset by your program.
SM201.0~SM201.2	SM201.0~SM201.2	Reserved
SM201.3	SM231.3	Direction control bit. 1 --- Disable the direction output channel. 0 --- Enable the direction output channel.
SM201.0~SM201.2	SM201.0~SM201.2	Reserved
Q0.0	Q0.1	Description
SMD212	SMD242	The current value

6.16.2.3 The error identification

During the execution of the position control instructions, non-fatal errors may occur, then the CPU will generate error identification, and write it to the *ERRID* parameter of the instruction. The following table describes these error codes and their descriptions.

Error Code	Description
0	No error
1	The value of <i>AXIS</i> is not 0 or 1.
2	The value of <i>MINF</i> is larger than the value of <i>MAXF</i> (200KHz).
3	The value of <i>MINF</i> is less than the allowed lowest frequency (20Hz).
4	The value of <i>TIME</i> (accelerating / decelerating time) doesn't match the value of <i>MINF</i> and <i>MAXF</i> .
5	The value of <i>MINF</i> is larger than that of <i>MAXF</i>

6.16.3 PHOME (Homing)

➤ Description

	Name	Usage	Group	
LD	PHOME	<div style="border: 1px solid black; background-color: #f08080; padding: 5px; display: inline-block;"> PHOME EN ENO AXIS DONE EXEC ERR HOME ERRID NHOME MODE DIRC MINF MAXF TIME </div>		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	PHOME	PHOME <i>AXIS, EXEC, HOME, NHOME, MODE, DIRC, MINF, MAXF, TIME, DONE, ERR, ERRID</i>	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>AXIS</i>	Input	INT	Constant (0 or 1)
<i>EXEC</i>	Input	BOOL	I, Q, V, M, L, SM, RS, SR
<i>HOME</i>	Input	BOOL	I, Q, V, M, L, SM, RS, SR
<i>NHOME</i>	Input	BOOL	I, Q, V, M, L, SM, RS, SR
<i>MODE</i>	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constant
<i>DIRC</i>	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constant
<i>MINF</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>MAXF</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>TIME</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>DONE</i>	Output	BOOL	Q, M, V, L, SM
<i>ERR</i>	Output	BOOL	Q, M, V, L, SM

<i>ERRID</i>	Output	BYTE	Q, M, V, L, SM
--------------	--------	------	----------------

The following table describes all the operands detailedly.

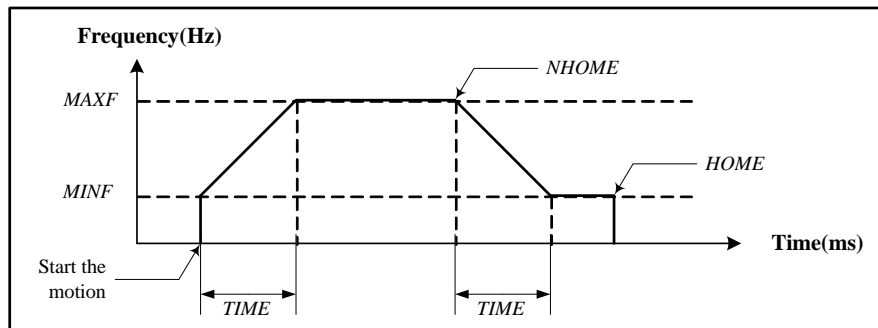
Operands	Description
<i>AXIS</i>	The high-speed output channel, 0 means Q0.0, 1 means Q0.1.
<i>EXEC</i>	If <i>EN</i> is 1, the <i>EXEC</i> starts the ‘search home’ motion on the rising edge.
<i>HOME</i>	The home signal from the home sensor
<i>NHOME</i>	The near home signal from the near home sensor
<i>MODE</i>	Specifies the homing mode: 0 means that the home signal and the near home signal are all used; 1 means that only the home signal is used.
<i>DIRC</i>	Specifies the rotating direction of the electric motor: 0 means rotating forwards; 1 means rotating backwards. Please refer to 6.16.2.1 The direction output channel for more information.
<i>MINF</i>	Specifies the initial speed (i.e., the initial frequency) of the pulse train output. Unit: Hz. Note: the value of <i>MINF</i> must be equal to or less than 2KHz.
<i>MAXF</i>	Specifies the highest speed (i.e., the highest frequency) of the pulse train output. Unit: Hz. The available range of <i>MAXF</i> is 20Hz ~ 20KHz. <i>MAXF</i> must be larger than or equal to <i>MINF</i> .
<i>TIME</i>	Specifies the acceleration/deceleration time. Unit: ms. In the position control instructions, the acceleration time is the same as the deceleration time. The acceleration time is the time for the speed accelerating from <i>MINF</i> to <i>MAXF</i> . The deceleration time is the time for the speed decelerating from <i>MAXF</i> to <i>MINF</i> .
<i>DONE</i>	Indicates that the instruction has finished successfully. 0 = not finished; 1 = finished.
<i>ERR</i>	Indicates that error has occurred during the execution. 0 = no error; 1 = an error has occurred.
<i>ERRID</i>	Error identification.

This instruction controls the *AXIS* to execute the ‘search home’ sequence using the *HOME* and *NHOME* signals. The *MODE* specifies the homing mode. While executing the ‘search home’ motion, if the *DIRC* is set to be 0 (rotating forwards), the current value (SMD212/SMD242) increases; if the *DIRC* is set to be 1 (rotating

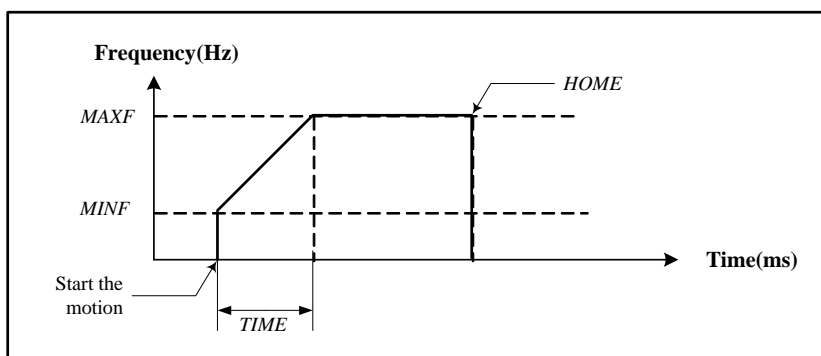
backwards), the current value (SMD212/SMD242) decreases.

- If the *MODE* is 0 (using both the *HOME* and the *NHOME* signals), the *PHOME* instruction will control the *AXIS* to decelerate as soon as the *NHOME* becomes 1, and to stop as soon as the *HOME* becomes 1.

The timing diagram is as followings:



- If the *MODE* is 1 (using the *HOME* signal only), the *PHOME* instruction will control the *AXIS* to stop as soon as the *HOME* becomes 1. The timing diagram is as followings:



■ **LD**

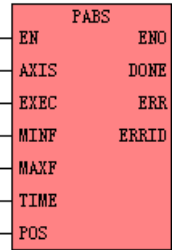
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.16.4 PABS (Moving Absolutely)

➤ Description

	Name	Usage	Group	
LD	PABS			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	PABS	PABS AXIS, EXEC, MINF, MAXF, TIME, POS, DONE, ERR, ERRID	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>AXIS</i>	Input	INT	Constant (0 or 1)
<i>EXEC</i>	Input	BOOL	I, Q, V, M, L, SM, RS, SR
<i>MINF</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>MAXF</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>TIME</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>POS</i>	Input	DINT	I, Q, M, V, L, SM, HC, Constant
<i>DONE</i>	Output	BOOL	Q, M, V, L, SM
<i>ERR</i>	Output	BOOL	Q, M, V, L, SM
<i>ERRID</i>	Output	BYTE	Q, M, V, L, SM

The following table describes all the operands detailedly.

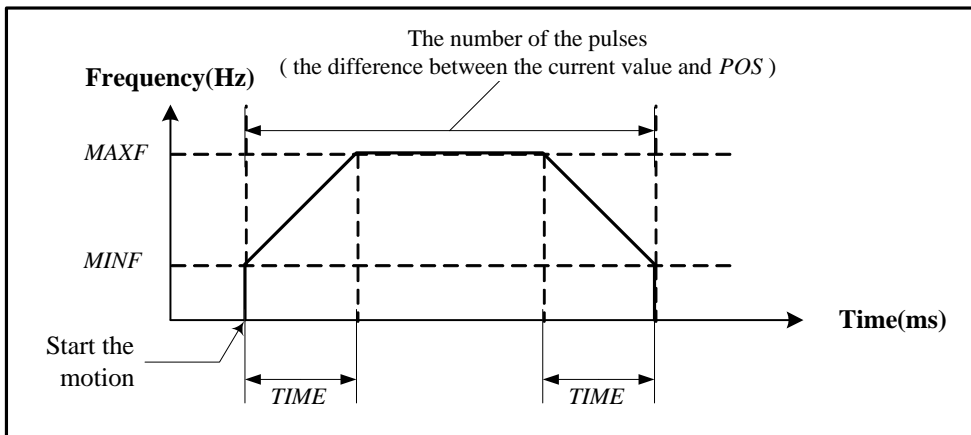
Operands	Description
<i>AXIS</i>	The high-speed output channel, 0 means Q0.0, 1 means Q0.1.
<i>EXEC</i>	If <i>EN</i> is 1, the <i>EXEC</i> starts the absolute motion on the rising edge.
<i>MINF</i>	Specifies the initial speed (i.e., the initial frequency) of the pulse train output. Unit: Hz. Note: the value of <i>MINF</i> must be equal to or less than 2KHz.

<i>MAXF</i>	<p>Specifies the highest speed (i.e., the highest frequency) of the pulse train output. Unit: Hz.</p> <p>The available range of <i>MAXF</i> is 20Hz ~ 20KHz.</p> <p><i>MAXF</i> must be larger than or equal to <i>MINF</i>.</p>
<i>TIME</i>	<p>Specifies the acceleration/deceleration time. Unit: ms.</p> <p>In the position control instructions, the acceleration time is the same as the deceleration time.</p> <p>The acceleration time is the time for the speed accelerating from <i>MINF</i> to <i>MAXF</i>.</p> <p>The deceleration time is the time for the speed decelerating from <i>MAXF</i> to <i>MINF</i>.</p>
<i>POS</i>	<p>Specifies the target value. It is represented with the number of pulses between the home position, where the current value is 0, and the target position.</p> <p>As shown in the following figure, if the object is moved from A to B, the <i>POS</i> should be set as '100'; If it is moved from B to C, the <i>POS</i> should be set as '300'; If it is moved from C to B, the <i>POS</i> should be set as '100'.</p> <div style="text-align: center;"> <p>The diagram shows a horizontal axis with an arrow pointing to the right. Three vertical dashed lines mark positions A, B, and C. Below the axis, 'Home (where the current value is 0)' is aligned with A, '100' is aligned with B, and '300' is aligned with C. An arrow points from A to B, another from B to C, and a third from C back to B.</p> </div>
<i>DONE</i>	<p>Indicates that the instruction has finished successfully.</p> <p>0 = not finished; 1 = finished.</p>
<i>ERR</i>	<p>Indicates that error has occurred during the execution.</p> <p>0 = no error; 1 = an error has occurred.</p>
<i>ERRID</i>	<p>Error identification.</p>

This instruction controls the *AXIS* to motion to the specified absolute position (*POS*), and it provides pulse train output until the current value is equal to the target value.

If the Direction Control Bit (SM201.3/SM231.3) is set to 0, the PABS instruction will generate the direction output signal at the corresponding direction output channel (Q0.2/Q0.3): If the target value is greater than the current value, it generates a direction output of rotating forwards, then the current value (SMD212/SMD242) increases; If the target value is less than the current value, it generates a direction output of rotating backwards, and then the current value (SMD212/SMD242) decreases.

The timing diagram is as following:



■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.16.5 PREL (Moving Relatively)

➤ Description

	Name	Usage	Group	
LD	PREL	<div style="border: 1px solid black; background-color: #f0f0f0; padding: 5px; display: inline-block;"> <pre> PREL - EN ENO - - AXIS DONE - - EXEC ERR - - MINF ERRID - - MAXF - TIME - DIST </pre> </div>		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	PREL	PREL AXIS, EXEC, MINF, MAXF, TIME, DIST, DONE, ERR, ERRID	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>AXIS</i>	Input	INT	Constant (0 or 1)
<i>EXEC</i>	Input	BOOL	I, Q, V, M, L, SM, RS, SR
<i>MINF</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>MAXF</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>TIME</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>DIST</i>	Input	DINT	I, Q, M, V, L, SM, HC, Constant
<i>DONE</i>	Output	BOOL	Q, M, V, L, SM
<i>ERR</i>	Output	BOOL	Q, M, V, L, SM
<i>ERRID</i>	Output	BYTE	Q, M, V, L, SM

The following table describes all the operands detailedly.

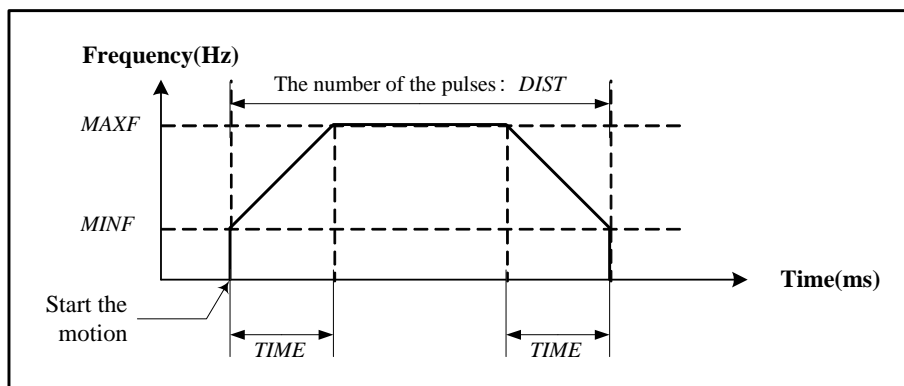
Operands	Description
<i>AXIS</i>	The high-speed output channel, 0 means Q0.0, 1 means Q0.1.
<i>EXEC</i>	If <i>EN</i> is 1, the <i>EXEC</i> starts the relative motion on the rising edge.
<i>MINF</i>	Specifies the initial speed (i.e., the initial frequency) of the pulse train output. Unit: Hz. Note: the value of <i>MINF</i> must be equal to or less than 2KHz.
<i>MAXF</i>	Specifies the highest speed (i.e., the highest frequency) of the pulse train output. Unit: Hz. The available range of <i>MAXF</i> is 20Hz ~ 20KHz. <i>MAXF</i> must be larger than or equal to <i>MINF</i> .
<i>TIME</i>	Specifies the acceleration/deceleration time. Unit: ms. In the position control instructions, the acceleration time is the same as the deceleration time. The acceleration time is the time for the speed accelerating from <i>MINF</i> to <i>MAXF</i> . The deceleration time is the time for the speed decelerating from <i>MAXF</i> to <i>MINF</i> .

<i>DIST</i>	<p>Specifies the target distance. It is represented with the number of pulses between the current position and the target position.</p> <p>As shown in the following figure, if the object is moved from A to B, the <i>DIST</i> should be set as '100'; If it is moved from B to C, the <i>DIST</i> should be set as '200'; If it is moved from C to B, the <i>DIST</i> should be set as '-200'.</p> <div style="text-align: center;"> <p>The diagram shows a horizontal axis with three points: A, B, and C. A is labeled 'Home (where the current value is 0)'. B is labeled '100' and C is labeled '300'. Vertical dashed lines mark these positions. A horizontal line with an arrow pointing right is drawn from A to B. Another horizontal line with an arrow pointing right is drawn from B to C. A third horizontal line with an arrow pointing left is drawn from C to B.</p> </div>
<i>DONE</i>	<p>Indicates that the instruction has finished successfully. 0 = not finished; 1 = finished.</p>
<i>ERR</i>	<p>Indicates that error has occurred during the execution. 0 = no error; 1 = an error has occurred.</p>
<i>ERRID</i>	<p>Error identification.</p>

This instruction controls the *AXIS* to execute a motion of a specified distance (*DIST*) relative to the current value at the time of the execution.

If the Direction Control Bit (SM201.3/SM231.3) is set to 0, the PREL instruction will generate the direction output signal at the corresponding direction output channel (Q0.2/Q0.3): If the *DIST* is positive, it generates a direction output of rotating forwards, then the current value (SMD212/SMD242) increases; If the *DIST* is negative, it generates a direction output of rotating backwards, and then the current value (SMD212/SMD242) decreases.

The timing diagram is as following:



■ **LD**

If EN is 1, this instruction is executed.

■ **IL**

If CR is 1, this instruction is executed, and it does not influence CR .

6.16.6 PJOG (Jog)

➤ Description

	Name	Usage	Group	
LD	PJOG	<div style="border: 1px solid black; background-color: #f0f0f0; padding: 5px; display: inline-block;"> PJOG EN ENO AXIS DONE EXEC ERR MAXF ERRID DIRC </div>		<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	PJOG	PJOG AXIS, EXEC, MINF, DIRC, DONE, ERR, ERRID	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>AXIS</i>	Input	INT	Constant (0 or 1)
<i>EXEC</i>	Input	BOOL	I, Q, V, M, L, SM, RS, SR
<i>MINF</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>DIRC</i>	Input	INT	I, Q, M, V, L, SM, AI, AQ, T, C, Constant
<i>DONE</i>	Output	BOOL	Q, M, V, L, SM
<i>ERR</i>	Output	BOOL	Q, M, V, L, SM
<i>ERRID</i>	Output	BYTE	Q, M, V, L, SM

The following table describes all the operands detailedly.

Operands	Description
<i>AXIS</i>	The high-speed output channel, 0 means Q0.0, 1 means Q0.1.
<i>EXEC</i>	If <i>EN</i> is 1, the <i>EXEC</i> starts the jog motion on the rising edge.
<i>MINF</i>	Specifies the speed (i.e., the initial frequency) of the pulse train output. Unit: Hz.
<i>DIRC</i>	Specifies the the direction of the electric motors: 0 means rotating forwards, and 1 means rotating backwards.
<i>DONE</i>	Indicates that the instruction has finished successfully. 0 = not finished; 1 = finished.
<i>ERR</i>	Indicates that error has occurred during the execution.

	0 = no error; 1 = an error has occurred.
<i>ERRID</i>	Error identification.

This instruction controls the *AXIS* to execute a jog motion: generating a durative pulse train output, whose frequency is *MINF*.

If the Direction Control Bit (SM201.3/SM231.3) is set to 0, the *PJOG* instruction will generate the direction output signal at the corresponding direction output channel (Q0.2/Q0.3): if the *DIRC* is 0 (rotating forwards), the current value (SMD212/SMD242) increases; if the *DIRC* is 1 (rotating backwards), the current value (SMD212/SMD242) decreases.

■ **LD**

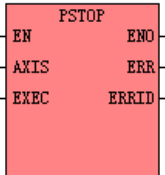
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

6.16.7 PSTOP (Stop)

➤ Description

	Name	Usage	Group	
LD	PSTOP			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	PSTOP	PSTOP AXIS, EXEC, ERR, ERRID	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>AXIS</i>	Input	INT	Constant (0 or 1)
<i>EXEC</i>	Input	BOOL	I, Q, V, M, L, SM, RS, SR
<i>ERR</i>	Output	BOOL	Q, M, V, L, SM
<i>ERRID</i>	Output	BYTE	Q, M, V, L, SM

The following table describes all the operands detailedly.

Operands	Description
<i>AXIS</i>	The high-speed output channel, 0 means Q0.0, 1 means Q0.1.
<i>EXEC</i>	If <i>EN</i> is 1, the <i>EXEC</i> stops the current motion on the rising edge.
<i>ERR</i>	Indicates that error has occurred during the execution. 0 = no error; 1 = an error has occurred.
<i>ERRID</i>	Error identification.

This instruction stops the current motion of the *AXIS*. At the same time, the Emergency-Stop flag (SM201.7/SM231.7) is set to 1, and no position control instruction can be executed until this flag is reset by your program.

■ **LD**


If *EN* is 1, this instruction is executed.

■ **IL**

If CR is 1, this instruction is executed, and it does not influence CR.

6.16.8 PFLO_F

➤ Description

	Name	Usage	Group	
LD	PFLO_F			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	PFLO_F	PFLO_F AXIS, F, NUME, DENOM, COUNT, DONE	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
AXIS	Input	INT	Constant (0 or 1)
F	Input	DINT	L, M, V, Constant
NUME	Input	INT	L, M, V, Constant
DENOM	Input	INT	L, M, V, Constant
DONE	Output	BOOL	Q, M, V, L
COUNT	Input	DWORD	L, M, V, Constant

The following table describes all the operands detailedly.

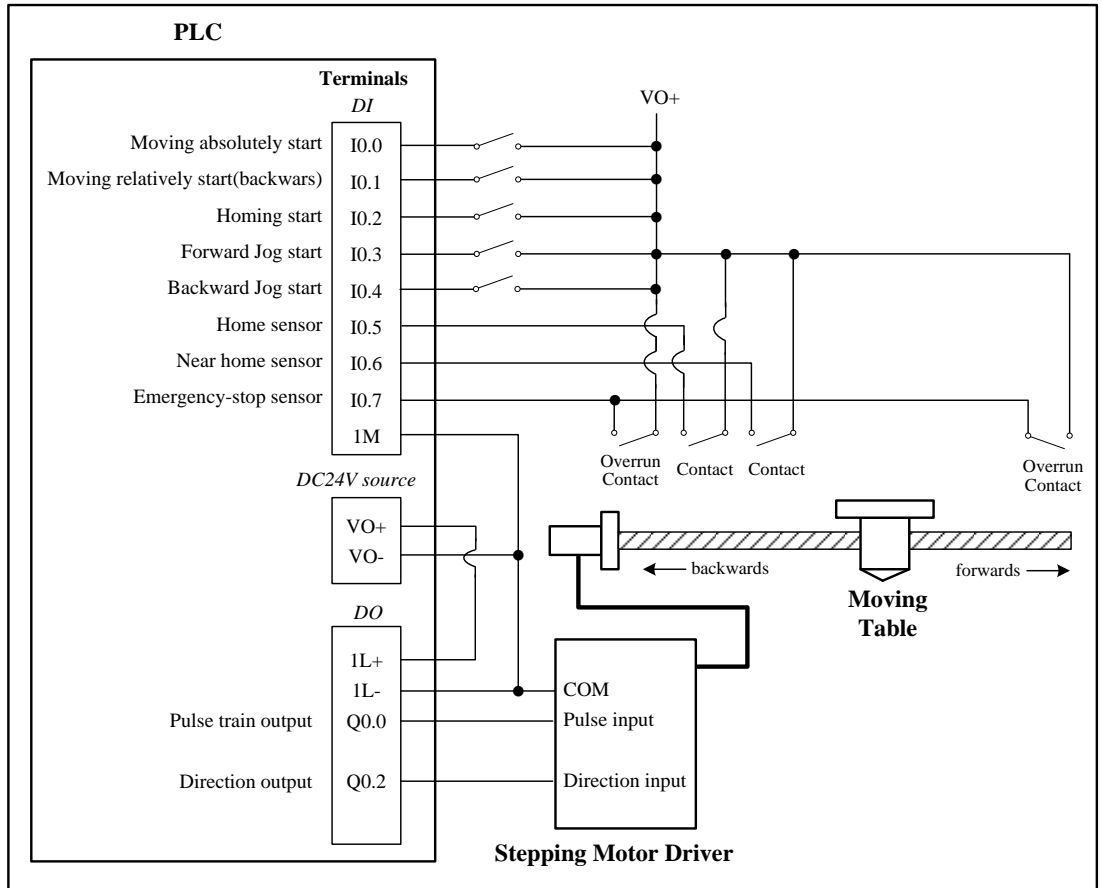
Operands	Description
EN	Enable. If En is 1 then will execute follow impulse Output otherwise will stop.

AXIS	High-speed tunnel. 0 represents using Q0.0, 1 represents Q0.1.
F	Input rate. Unit:Hz
NUME	Electronic gear numerator of output impulse rate.
DENOM	Electronic gear denominator of output impulse rate.
DONE	Done. If impulse Output exists then it will be 0, otherwise it will be 1.

6.16.9 Examples

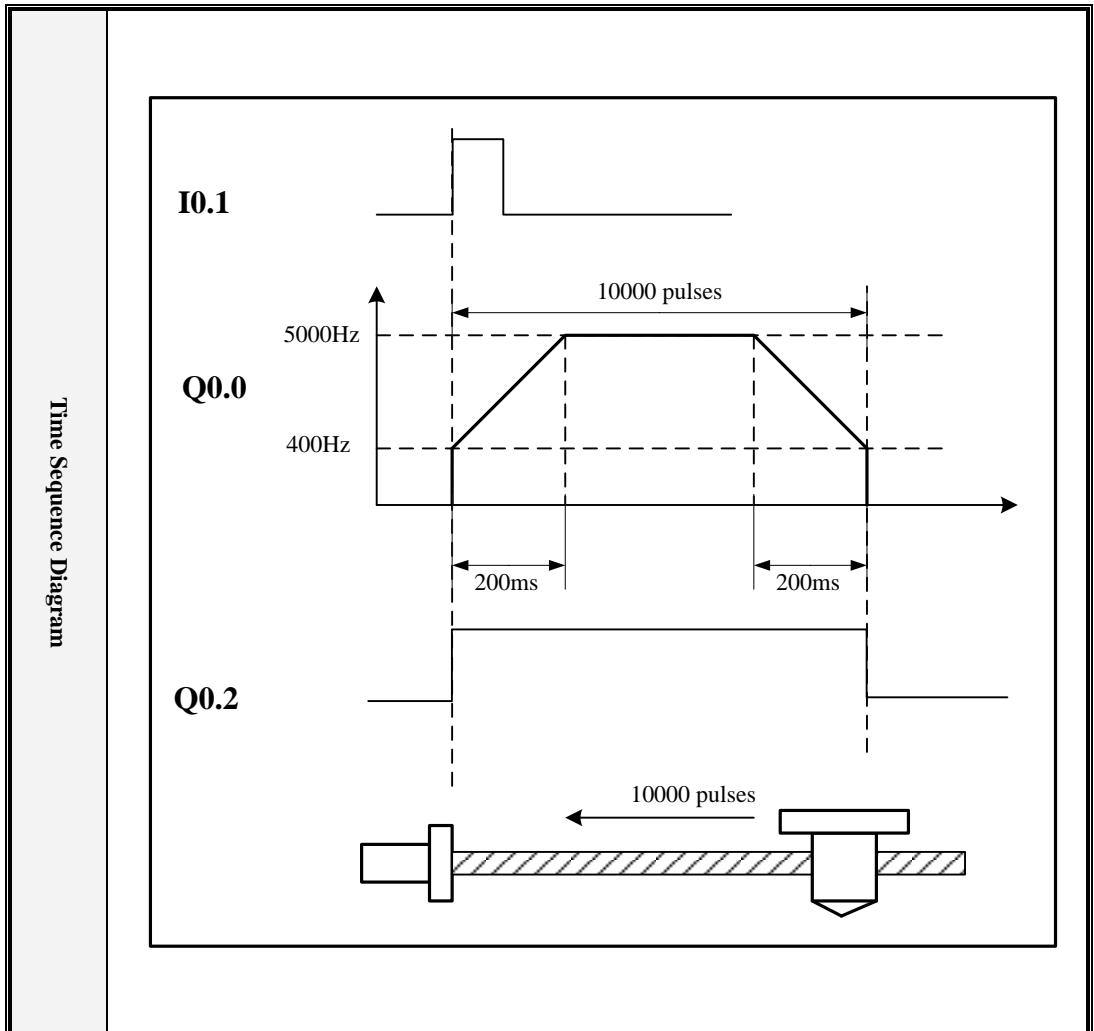
➤ Wiring

The following system is taken as the example to describe how to use the instructions PREL, PABS, PHOME, PJOG and PSTOP.



➤ Moving relatively

I0.1 is used for starting to move relatively (backwards).



LD

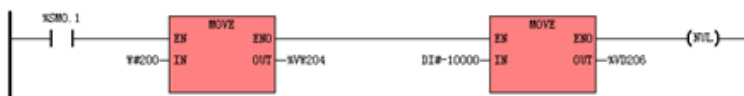
(* Network 0 *)

(*Set the initial frequency and the maximum frequency*)



(* Network 1 *)

(*Set the acceleration/deceleration time and the distance*)



(* Network 2 *)

(*Reset the emergency-stop flag*)



(* Network 3 *)

(*Call the PREL instruction*)

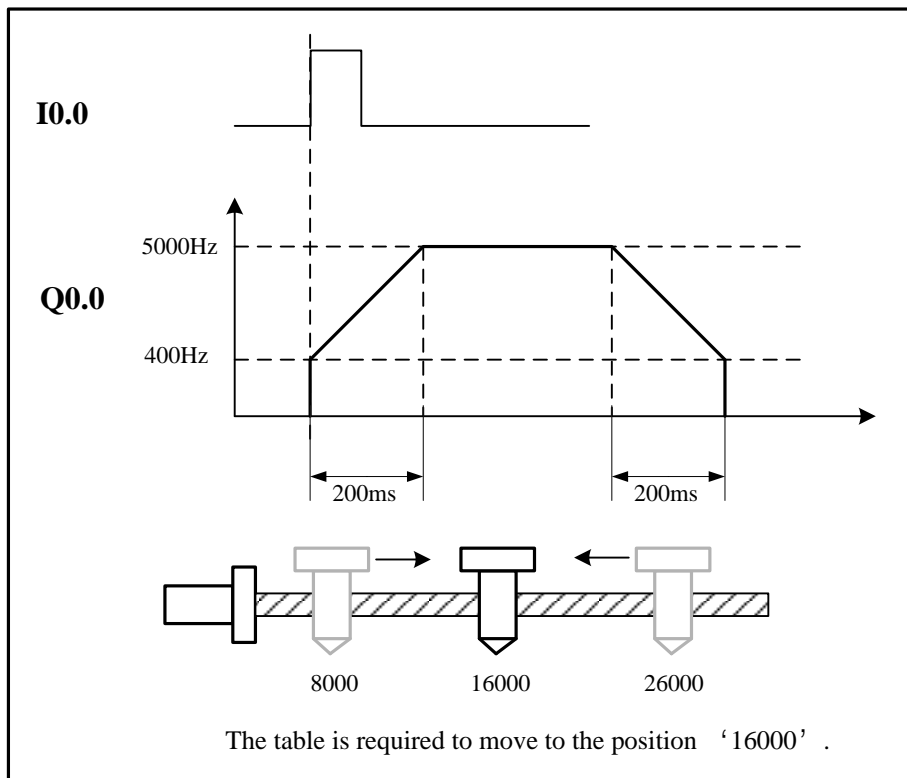


IL	(* Network 0 *)
	(*Set the initial frequency and the maximum frequency*)
	LD %SM0.1
	MOVE W#400, %VW200
	MOVE W#5000, %VW202
	(* Network 1 *)
	(*Set the acceleration/deceleration time and the distance*)
	LD %SM0.1
	MOVE W#200, %VW204
	MOVE DI#-10000, %VD206
	(* Network 2 *)
	(*Reset the emergency-stop flag*)
	LD %I0.1
	R %SM201.7
(* Network 3 *)	
(*Call the PREL instruction*)	
LD %SM0.0	
PREL 0, %I0.1, %VW200, %VW202, %VW204, %VD206, %M1.0, %M1.1, %VB1	

- Moving absolutely

I0.0 is used for starting to move absolutely.

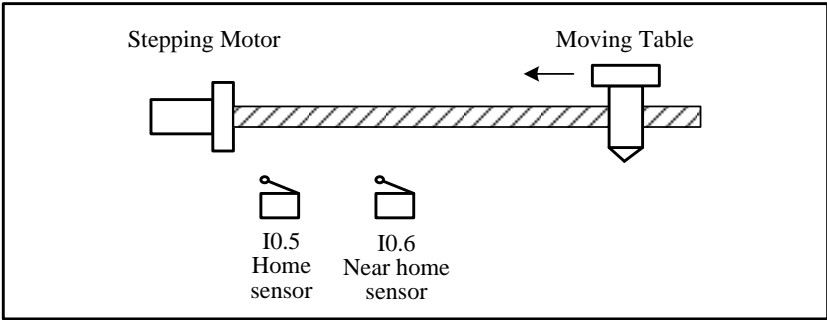
Time Sequence Diagram



➤ Home

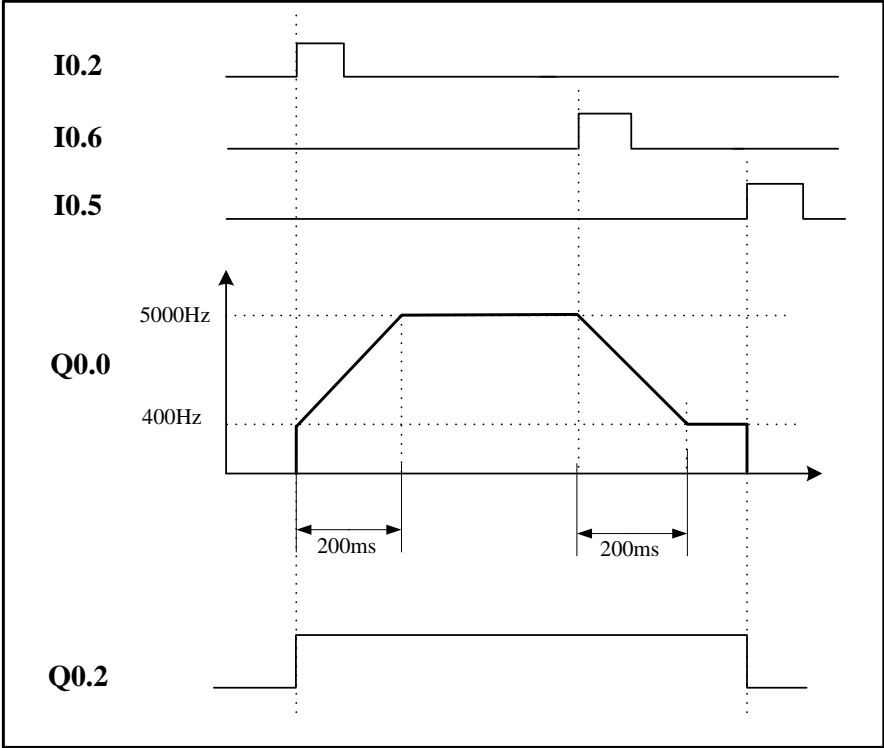
I0.2 is used for starting to return to the home position,

Supposing that the moving is in the following initial status:



During the motion, Q0.2 is 1 because of moving backwards.

Time Sequence Diagram

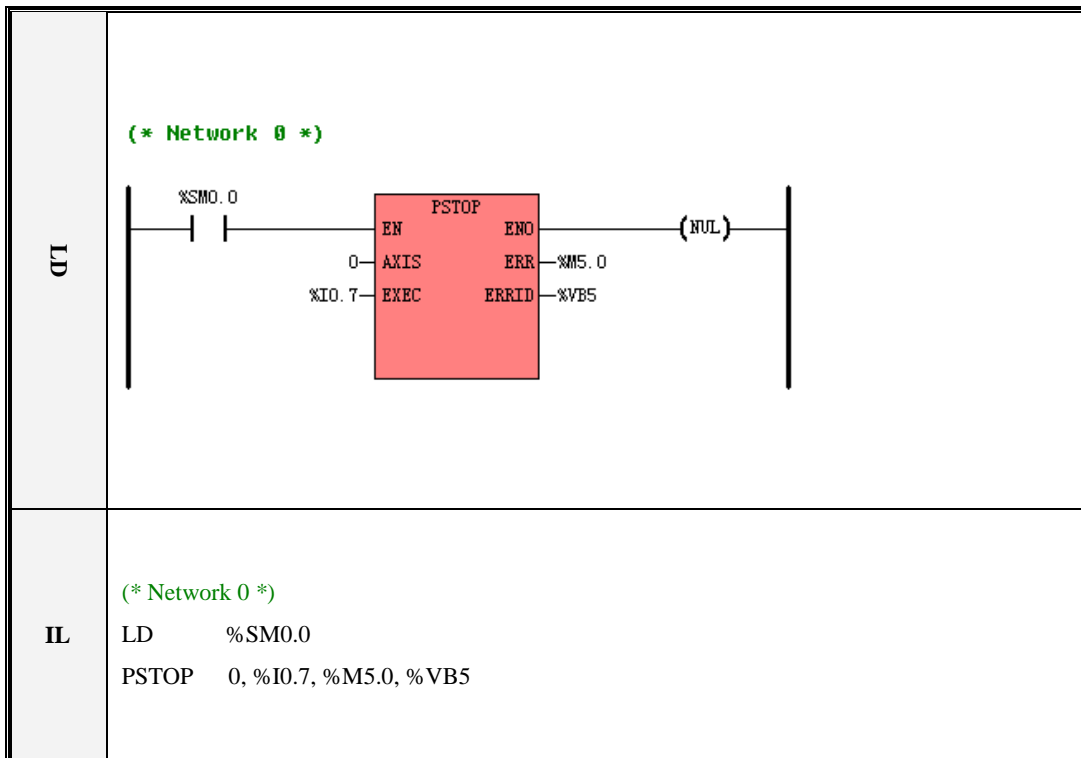


IL	<p>(* Network 0 *)</p> <p>(*use both the home and the near home input; move backwards*)</p> <pre>LD %SM0.1 MOVE 0, %VW396 MOVE 1, %VW398</pre> <p>(* Network 1 *)</p> <p>(*set the initial frequency, maximum frequency and acceleration/deceleration time*)</p> <pre>LD %SM0.1 MOVE W#400, %VW400 MOVE W#5000, %VW402 MOVE W#200, %VW404</pre> <p>(* Network 2 *)</p> <p>(*Reset the emergency-stop flag*)</p> <pre>LD %I0.2 R %SM201.7</pre> <p>(* Network 3 *)</p> <pre>LD %SM0.0 PHOME 0, %I0.2, %I0.5, %I0.6, %VW396, %VW398, %VW400, %VW402, %VW404, %M3.0, %M3.1, %VB3</pre>
-----------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

IL	(* Network 0 *)
	(*Set the frequency of PTO*)
	LD %SM0.1
	MOVE W#1000, %VW500
	(* Network 1 *)
	(*Set the direction*)
	LD %I0.3
	ANDN %I0.4
	MOVE 0, %VW502
	(* Network 2 *)
	LD %I0.4
	ANDN %I0.3
	MOVE 1, %VW502
	(* Network 3 *)
	(*Jog*)
	LD %I0.3
	OR %I0.4
	ST %M10.0
	R %SM201.7
	(* Network 4 *)
LD %SM0.0	
PJOG 0, %M10.0, %VW500, %VW502, %M4.0, %M4.1, %VB4	

➤ Stop

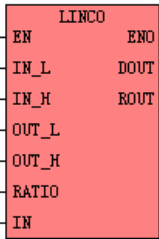
There are 2 overrun contacts at the 2 ends of the feed screw, and they are connected in parallel to I0.7 as the emergency-stop signal



6.17 Additional Instructions

6.17.1 LINCO (Linear Calculation)

➤ Description

	Name	Usage	Group	
LD	LINCO			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	LINCO	LINCO IN_L, IN_H, OUT_L, OUT_H, RATIO, IN, DOUT, ROUT	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN_L</i>	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constants
<i>IN_H</i>	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constants
<i>OUT_L</i>	Input	REAL	V, L, Constants
<i>OUT_H</i>	Input	REAL	V, L, Constants
<i>RATIO</i>	Input	REAL	Constants
<i>IN</i>	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ
<i>DOUT</i>	Output	DINT	Q, M, V, L, SM
<i>ROUT</i>	Input	REAL	V, L

Note: *IN_L*, *IN_H*, *OUT_L* and *OUT_H* must be all constants or all variables.

This instruction calculates the input *IN* according to the specified linear relation, and multiplies the result with the coefficient *RATIO*, and then assigns the new result to *ROUT*. Also, the truncated DINT value of *ROUT* (by

discarding the decimal part) to *DOUT*. The linear relation is specified according to the method '2 points decide a line', and the 2 points are (*IN_L*, *OUT_L*) and (*IN_H*, *OUT_H*).

The function of LINCO instruction can be described with the following formula:

$$ROUT = RATIO * (k * IN + b)$$

$$DOUT = TRUNC(ROUT)$$

Therein, $k = \frac{OUT_H - OUT_L}{IN_H - IN_L}$, $b = OUT_L - k \times IN_L$.

■ **LD**

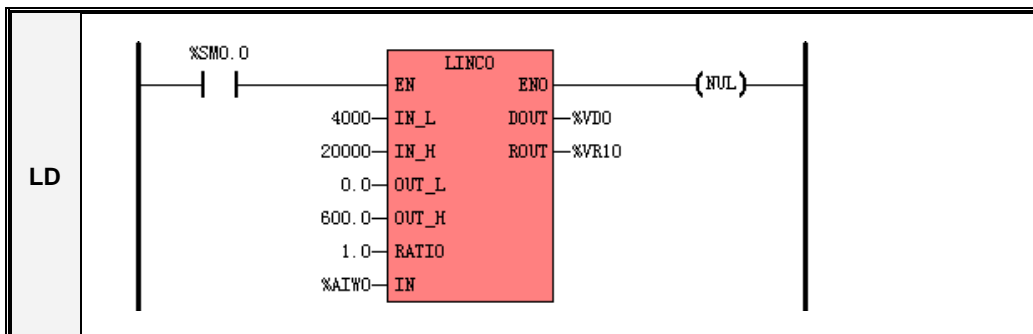
If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

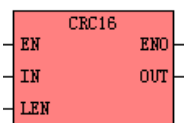
Assume that the measurement range of a temperature transducer is 0~600 °C, and its output range is 4~20mA. The output signal of the transducer is connected to the channel AIW0 of the KINCO-K5. Now the KINCO-K5 needs to calculate the actual temperature value.



IL	LD %SM0.0 LINCO 4000, 20000, 0.0, 600.0, 1.0, %AIW0, %VDO, %VR10
-----------	----------------------------------------------------------------------------

6.17.2 CRC16 (16-Bit CRC)

➤ Description

	Name	Usage	Group	
LD	CRC16			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA
IL	CRC16	CRC16 IN, OUT, LEN	U	<input checked="" type="checkbox"/> CPU508

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>IN</i>	Input	BYTE	I, Q, M, V, L, SM
<i>LEN</i>	Input	BYTE	I, Q, M, V, L, SM, Constant
<i>OUT</i>	Output	BYTE	Q, M, V, L, SM

This instruction calculates the 16-bit CRC (Cyclical Redundancy Check) for the number *LEN* of successive variables beginning with *IN*, and puts the result into 2 continuous byte variables beginning with *OUT*. Therein, *OUT* is the high byte of the CRC, and the succeeding byte variable after *OUT* is the low byte of the CRC.

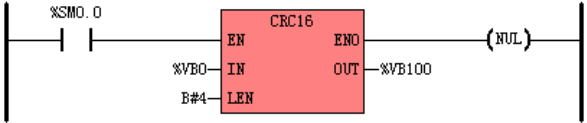
■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

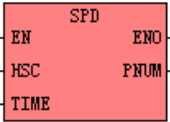
If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

LD		<p>SM0.0 is always 1, so CRC16 is always executed: calculates the CRC for the 4 continuous bytes beginning with VB0, then puts the high byte of the result into VB100, and the low byte into VB101.</p>																		
IL	<pre>LD %SM0.0 CRC16 %VB0, %VB100, B#4</pre>																			
Result	<p>The result is as the following:</p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="4">The data to be checked</th> <th colspan="2">16-bit CRC</th> </tr> <tr> <th>VB0</th> <th>VB1</th> <th>VB2</th> <th>VB3</th> <th>VB100</th> <th>VB101</th> </tr> </thead> <tbody> <tr> <td>B#16#1A</td> <td>B#16#2B</td> <td>B#16#3C</td> <td>B#16#4D</td> <td>B#16#A6</td> <td>B#16#1</td> </tr> </tbody> </table>		The data to be checked				16-bit CRC		VB0	VB1	VB2	VB3	VB100	VB101	B#16#1A	B#16#2B	B#16#3C	B#16#4D	B#16#A6	B#16#1
The data to be checked				16-bit CRC																
VB0	VB1	VB2	VB3	VB100	VB101															
B#16#1A	B#16#2B	B#16#3C	B#16#4D	B#16#A6	B#16#1															

6.17.3 SPD (Speed detection)

➤ Description

	Name	Usage	Group	
LD	SPD			<input checked="" type="checkbox"/> CPU504 <input checked="" type="checkbox"/> CPU504EX <input checked="" type="checkbox"/> CPU506 <input checked="" type="checkbox"/> CPU506EA <input checked="" type="checkbox"/> CPU508
IL	SPD	SPD HSC, TIME, PNUM	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
<i>HSC</i>	Input	INT	Constant (the number of a HSC)
<i>TIME</i>	Input	WORD	I, Q, M, V, L, SM, Constant
<i>PNUM</i>	Output	DINT	Q, M, V, L, SM

This instruction counts the number of the pulses received at the specified High-speed counter, whose number is *HSC*, in the specified time frame (*TIME*, in ms), and writes the result to the output *PNUM*.

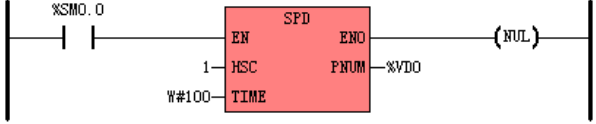
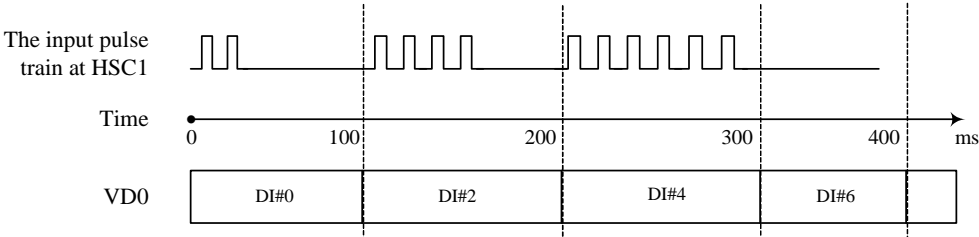
■ **LD**

If *EN* is 1, this instruction is executed.

■ **IL**

If *CR* is 1, this instruction is executed, and it does not influence *CR*.

➤ Examples

LD		<p>SM0.0 is always 1, so SPD is always executed: count the number of the pulses received at HSC1 every 100ms, and write the result to VD0.</p>
IL	<pre>LD %SM0.0 SPD 1, W#100, %VD0</pre>	
Result	<p>The result is as the following:</p> 	

Appendix A Communicate Using Modbus RTU Protocol

Default, the Kinco-K5 serves as a slave using Modbus RTU Protocol, and can communicate with a Modbus RTU master directly.

1. PLC Memory Area

1.1 Accessible Memory Areas

The memory areas that can be accessed by a Modbus RTU master are classified as follows:

Type	Available Function Code	Corresponding Memoery Area of PLC
DO (Digital Output, 0XXXX)	01, 05, 15	Q, M
DI (Digital Input, 1XXXX)	02	I, M
AO (Analog Output, 4XXXX)	03, 06, 16	AQ, V
AI (Analog Input, 3XXXX)	04	AI, V
Error record (16-bit whole number without sign)	03, 04	PLC error recording area

1.2 Modbus Register Number

Due to the memory differences between CPU504/CPU 504EX and other CPUs, the ranges that allow to visit have limitation as well.

(1) In some equipment, modbus registers begin with 1, so 1 should be added to each data in this column.

➤ For CPU504

Area	Range	Type	Corresponding Modbus Registers
------	-------	------	--------------------------------

I	I0.0 --- I0.7	DI	0 --- 7
Q	Q0.0 --- Q0.5	DO	0 --- 5
M	M0.0 --- M1023.7	DI/DO	320 -- 8511
AI	---	AI	---
AQ	---	AO	---
V	VW0 ---VW4094	AI/AO	100 -- 2147

➤ **For CPU504EX**

Area	Range	Type	Corresponding Modbus Registers
I	I0.0 --- I4.7	DI	0 --- 39
Q	Q0.0 --- Q4.7	DO	0 --- 39
M	M0.0 --- M1023.7	DI/DO	320 -- 8511
AI	AIW0 --- AIW14	AI	0 --- 15
AQ	AQW0 --- AQW14	AO	0 --- 15
V	VW0 ---VW4094	AI/AO	100 -- 2147

➤ **For CPU506, CPU506EA and CPU508**

Area	Range	Type	Corresponding Modbus Registers
I	I0.0 --- I31.7	DI	0 --- 255
Q	Q0.0 --- Q31.7	DO	0 --- 255
M	M0.0 --- M31.7	DI/DO	320 -- 8511
AI	AIW0 --- AIW62	AI	0 --- 31
AQ	AQW0 --- AQW62	AO	0 --- 31
V	VW0 ---VW4094	AI/AO	100 -- 2147

➤ **Error Record**

Modbus Register No.	Description
9000-9127	Latest 128 common error codes after PLC starts Among which 9000 is the latest error and 9001 is the second latest

9128-9255	Latest 128 common error codes after PLC starts Among which 9128 is the latest error and 9129 is the second latest.
9256-9383	Latest 128 common error codes after PLC starts Among which 9256 is the latest error and 9383 is the second latest
9384-9511	Latest 128 common error codes after PLC starts Among which 9384 is the latest error and 9511 is the second latest

2. Basic Report Format of Modbus RTU

In CRC identification codes, usually high bytes will be in prior to low bytes

Report interval time for report that are no longer than 3.5 characters	Target Station	Function Code	Data	CRC Identification Code
	1 byte	1 byte	N bytes	2 bytes

2.1 Modbus RTU

2.1.1 Function Code 01: Read Coil (On/Off Output)

Request format:

Target Station Number	Function Code	Initial Address		Read Amount		CRC
1 byte	01	High byte	Low byte	High byte	Low byte	2 bytes

Correct response format

Station Number	Function Code	Return data byte amount	Return data byte 1	Return data byte 2	...	CRC
1 byte	01	1 byte	1 byte	1 byte	...	2 byte

2.1.2 Function Code 02: Write/read Status (On/Off Input)

Request Format: Request Format

Target Station Number	Function Code	Initial Address		Read Amount		CRC
1byte	02	High byte	Low byte	High byte	Low byte	2byte

Correct Response Format:

Station Number	Function Code	Return data byte amount	Return data byte 1	Return data byte 2	...	CRC
1byte	02	1byte	1byte	1byte	...	2byte

2.1.3 Function Code03: Read Control Register (Analog Qty Output)

Request Format:

Target Station Number	Function Code	Initial Address		Read Amount		CRC
1byte	03	High byte	Low byte	High byte	Low byte	2byte

Correct Response Format:

Station Number	Function Code	Return data byte amount	Register1High byte	Register1Low byte	...	CRC
1byte	03	1byte	1byte	1byte	...	2byte

2.1.4 Function Code04: Read Input Register(Analog Qty Input)

Request Format:

Target Station Number	Function Code	Initial Address		Read Amount		CRC
1byte	04	High byte	Low byte	High byte	Low byte	2byte

Correct Response Format:

Station Number	Function Code	Return data byte amount	Register1High byte	Register1Low byte	...	CRC
1byte	04	1byte	1byte	1byte	...	2byte

2.1.5 Function Code05: Write Single Coil (On/Off Output)

Request Format: If set successfully then return the original report

Target Station Number	Function Code	Coil Address		Force Value		CRC
1byte	05	High byte	Low byte	High byte	Low byte	2byte

NOTE: Force Value = 0xFF00 then the Coil is connected; Force Value = 0x0000 then the Coil is disconnected

Response Format: If set successfully then return the original report

2.1.6 Function Code06: Write Single Control Register (Analog Qty Output)

Request Format:

Target Station Number	Function Code	Register Address		Force Value		CRC
1byte	06	High byte	Low byte	High byte	Low byte	2byte

Response Format: If set successfully then return the original report

2.1.7 Function Code15: Write Multiple Coil (On/Off Output)

Request Format:

Target Station Number	Function Code	Initial Address		Write Amount		Force Value Byte Amount	Force Value 1st byte	...	CRC
1byte	15	High byte	Low byte	High byte	Low byte	1byte	1byte	...	2byte

Correct Response Format:

Target Station Number	Function Code	Initial Address		Write Amount		CRC	
1byte	15	High byte	Low byte	High byte	Low byte	High byte	Low byte

2.1.8 Function Code16: Write Hold Register (On/Off Output)

Request Format:

Target Station Number	Function Code	Initial Address		Write Qty Amount		Force Value byte Amount	Force Value1 High byte	Force Value1 Low byte	...	CRC
1byte	16	High byte	Low byte	High byte	Low byte	1byte	1byte	1byte	...	2byte

Correct Response Format:

Target Station Number	Function Code	Initial Address		Write Amount		CRC
1byte	16	High byte	Low byte	High byte	Low byte	2byte

2.2 CRC Verification Algorithm in Modbus Protocol

In Modbus RTU Protocol, frames is verified by CRC. The algorithms are as follows:

2.2.1 Direct CRC Calculation

/* Parameter: chData —— const BYTE*, direct to the initial address of the memory storage of the data save area to be verified

uNO —— Byte qty. of the data to be verified

Return Value: WORD, to calculate the CRC value */

WORD CalcCrc(const BYTE* chData, WORD uNo)

```
{
    WORD crc=0xFFFF;
    WORD wCrc;
    UCHAR i,j;
    for (i=0; i<uNo; i++)
    {
        crc ^= chData[i];
        for (j=0; j<8; j++)
        {
            if (crc & 1)
            {
                crc >>= 1;
                crc ^= 0xA001;
            }
            else
                crc >>= 1;
        }
    }
}
```

```
}  
  
wCrc=( (WORD)LOBYTE(crc) )<<8;  
wCrc=wCrc|((WORD)HIBYTE(crc) );  
return (wCrc);  
}
```

2.2.2 Fast CRC Calculation by Referring to Table

/ High Byte CRC Table */*

```
const UCHAR auchCRCHi[] =  
{  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,  
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,  
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,  
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,  
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,  
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,  
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,  
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,  
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,  
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,  
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,  
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
```

```
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,  
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,  
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40  
};
```

```
/* Low Byte CRC Table */
```

```
const UCHAR auchCRCLo[] =  
{  
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,  
0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,  
0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,  
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,  
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,  
0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,  
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,  
0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,  
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,  
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,  
0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,  
0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,  
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,  
0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,  
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,  
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,  
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,  
0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,  
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,  
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,  
0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,  
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,  
0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,  
0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
```



```
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,  
0x43, 0x83, 0x41, 0x81, 0x80, 0x40  
};
```

/ Parameter: puchMsg — const BYTE*, direct to the initial address of the memory storage of the data save area to be verified*

usDataLen — Byte qty. of the data to be verified

*Return Value: WORD, to calculate CRC */*

```
WORD CKINCOSerialCom::CalCrcFast(const BYTE* puchMsg , WORD usDataLen)
```

```
{  
    BYTE uchCRCHi = 0xFF ; /* CRC High Byte Initialization */  
    BYTE uchCRCLo = 0xFF ; /* CRC Low Byte Initialization */  
    WORD uIndex ; /* CRC Table Index*/  
  
    while (usDataLen--)  
    {  
        uIndex = uchCRCHi ^ *puchMsg++ ; /* Calculate CRC */  
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex];  
        uchCRCLo = auchCRCLo[uIndex] ;  
    }  
    return (uchCRCHi << 8 | uchCRCLo) ;  
}
```

Appendix B Dynamic Modification Parameter of RS485

Communication Port

It is default that they can only take effect when the parameters of each communication port to be edited in **[PLC Hardware Configuration]** and downloaded in PLC.

In the meantime, K5 provides the function for user to use certain SM register to modify the RS485 communication port (PORT1 and PORT2). PORT 0 is the programming port that may be frequently used therefore is not allowed to be modified.

1. General Description

- Allow to dynamically modify the [PLC Station Number], [Baud Rate] and [Odd-Even Check]
- Dynamically modified parameter value is stored in the permanent storage.
- The priority of communication parameter dynamically modified is higher than that in **[PLC Hardware Configuration]**. If the user re-download a new project K5 will have the priorities to adopt communication parameter dynamically modified. User may use **[PLC]** → **[Clear...]** to clear all parameters.
- After the communication parameters be modified, [PLC Station Number] will take effect immediately but **[Baud Rate]** and **[Odd-Even Check]** is not for certain: if the communication port is free then the two parameter will take effect immediately; otherwise not.

All modified parameters will take effect immediately when PLC reboot next time.

2. Register Instruction

K5 will use SMB20--SMB25 to modify the communication parameters. Please find below:

➤ **Parameters: SMB23, SMB24 and SMB25**

SMB	Description
SMB23	PLC station number. Valid range: 1-31 If execute write, SMB23 will write new PLC station number; if execute read, SMB23 can only read current PLC station number; if execute clear SMB23 will be ignored.
SMB24	Baud rate. Valid range 0-5: 0 represents 2400, 1 represents 4800, 2 represents 9600 and 3 represents 19200 If execute write, SMB24 will write new baud rate; if execute read, SMB24 can only read current baud rate; if execute clear SMB24 will be ignored.
SMB25	Odd-Even Check. Valid range: 0-2. 0 represents no check, 1 represents odd check and 2 represents even check. If execute write, SMB25 will check the odd-even value that to be wrote; if execute read SMB25 will read current odd-even value; if execute clear SMB25 will be ignored.

➤ **Control byte: SMB20 and SMB21**

Bit	Description
SMB20: Assign the port and operation	
SM20.7	Value = 1 represents execution of write. PLC will clear the bit to 0 after new parameters wrote in.
SM20.6	Value = 1 represents execution of read. PLC will clear the bit to 0 after new parameters wrote in.
SM20.5	Value = 1 represents execution of clear. PLC will clear the bit to 0 after new parameters wrote in.
SM20.4	Remain, must be valued to 0.
SM20.3 ~ SM20.0	These four items are the port numbers to be operated 1 represents PORT1 and 2 represents PORT2. If the bits are set to other null value, errors will occur and PLC will quit operating
SMB21: Assign the communication parameters to be operated	

SM21.7 ∼ SM21.3	Substitute. Must be valued to 0.
SM21.2	1 represents editing or clearing the odd-even value of certain communication port.
SM21.1	1 represents editing or clearing the baud rate of certain communication port.
SM21.0	1 represents editing or clearing the PLC signal of certain communication port.

At the same moment, only one value of SM20.5, SM20.6 and SM20.7 is allowed to be 1 or errors will occur and PLC will quit operation.

If execute read, the value of SMB21 will be ignored and PLC will read all communication parameters in one-time

When one parameter is cleared PLC will adopt corresponding parameters in the hardware configuration.

➤ **Status bit: SMB22**

In SMB22 the operation result of this dynamic modification of communication parameters is saved

Bit (read-only)	Description
SM22.7	1 represents completion of operation. If the operation is completed, regardless of success or failure, SM22.7 will be set to 1 automatically. Only when SM22.7 set to 1 other bits will be valid in SMB22.
SM22.6	When SM22.7 is set to 1, if SM22.6 set to 1 represents success operation and if set to 0 represents failure.
SM20.5 ∼ SM20.0	If operation fails these bits will show error codes, please see below.

Error Code	Error Description
1	Wrong command, e.g. SM20.7 and SM20.6 be set to 1 at meantime.
2	Wrong port.
3	Wrong SMB21 value.
4	Wrong SMB23 value.
5	Wrong SMB24 value.
6	Wrong SMB25 value.
10	Fail to read PORT1 dynamic PLC station number saved in permanent register.
11	PORT1 dynamic PLC station number not set yet.

12	Fail to read PORT1 dynamic PLC baud rate saved in permanent register.
13	PORT1 dynamic PLC dynamic baud rate not set yet.
14	Fail to read PORT1 dynamic PLC odd-even value saved in permanent register.
15	PORT1 dynamic PLC dynamic odd-even value not set yet.
20	Fail to read PORT2 dynamic PLC station number saved in permanent register.
21	PORT2 dynamic PLC station number not set yet.
22	Fail to read PORT2 dynamic PLC baud rate saved in permanent register.
23	PORT1 dynamic PLC dynamic baud rate not set yet.
24	Fail to read PORT2 dynamic PLC odd-even value saved in permanent register.
25	PORT2 dynamic PLC dynamic odd-even value not set yet.
61	Fail to write dynamic communication parameter into permanent register.

3. Instructions

- Edit the communication parameters
- Set low four of SMB20 to the port number to be operated
e.g. SMB=B#1 represents parameters of PORT1 to be operated
- Give corresponding values to SMB21 in accordance with the type of parameter
e.g. SMB21=B#16#03 represents PLC station number and baud value to be edited.
- Give expecting new parameters to corresponding register: SMB23 is the station number of new PLC, SMB24 is the new baud value and SMB25 is the new odd-even check value.
E.g. SMB23=B#03 represents editing PLC station number to 3, SMB24=B#3 represents editing baud rate to 19200.
- (Optional) If the bit had been through a parameter operation (read, write or clear), SM22.7 should be checked in prior. Only when SM22.7 be set to 1 can the operation starts.
- Value SM20.7 to 1 and start write operation. PLC will clear SM20.7 after new parameters wrote in.
- (Optional) Check bit SM22.7 and SM22.6. Both setting to 1 represents successfully editing the parameters.
- Read PLC Communication Parameter
- Set low-four of SMB20 to the port number to be operated.
e.g. SMB21=B#1 represents PORT1 parameter to be read.
- (Optional) If the bit had been through a parameter operation (read, write or clear), SM22.7 should be checked in prior. Only when SM22.7 be set to 1 can the operation starts.
- Value SM20.6 to 1 and start write operation. PLC will clear SM20.6 after new parameters wrote in.
- Check Sm22.7 and SM22.6. Both setting to 1 represents successfully editing the parameters. After read parameters will be saved in registers as follows: in SMB23 is the PLC station number, in SMB24 is the baud rate and SMB25 is the odd-even check value.
- Clearing the communication parameters of PLC
- Set low-four of SMB20 to the port number to be operated.

e.g. SMB21=B#1 represents PORT1 parameter to be read.

- Value the corresponding value in accordance with parameter types to be cleared.
e.g. SMB21=B#1 represents dynamic PLC station number and baud rate to be cleared in permanent registers.
- (Optional) If the bit had been through a parameter operation (read, write or clear), SM22.7 should be checked in prior. Only when SM22.7 be set to 1 can the operation starts.
- Value SM20.5 to 1 and start write operation. PLC will clear SM20.5 after new parameters wrote in.
- (Optional) Check Sm22.7 and SM22.6. Both setting to 1 represents successfully editing the parameters.

4. Example

Following example is the edition of dynamic station number of PORT1 and PORT2 on HMI. Its programming language is IL, you may copy them in the KincoBuilder editor and execute [Project]→[LD Language].

VW47 is the new station number which can be edited on HMI. VW48 can also be saved in VW3690. PLC will check the real-time value of VW48 and compare it to the that saved in VW3690. If the value changes and VW48 given valid value, then VW48 will be regarded as the new station number of PORT1 and PORT2 and the edition starts.

(* Network 0 *)

(*When powered on use values that saved permanently to initialize VW48*)

```
LD      %SM0.1
MOVE    %VW3690, %VW48
```

(* Network 1 *)

(*Judge if VB48 value changes and the its validity. Then save BW48 and starts edition.*)

```
LD      %SM0.0
GE      %VB48, B#1
LE      %VB48, B#31
NE      %VW48, %VW3690
MOVE    %VW48, %VW3690
ST      %M999.7
```

(* Network 2 *)

(*Start editing PORT1 station number*)

```
LD      %M999.7
R_TRIG
MOVE    B#1, %SMB20
MOVE    B#1, %SMB21
MOVE    %VB48, %SMB23
S       %SM20.7
S       %M999.6
```

(* Network 3 *)

(*PORT2 should be edited after PORT1 successfully been edited *)

LD %M999.6

AND %SM22.7

R_TRIG

AND %SM22.6

MOVE B#2, %SMB20

S %SM20.7

R %M999.6

Appendix C Data Backup

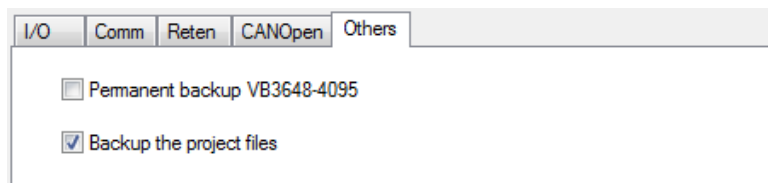
Data backup means write data in to permanent register to allow PLC retain the data even when the electricity is cut.

Permanent register of K5 adopts FARM, which allows write 10 billion times. You should be noted that: **you may only backup data when necessary. If the register loses its efficacy it will cause CPU errors.**

K5 provides data backup area in V area, in which data will be automatically wrote in the permanent register. Please see below table:

Length	448 bytes
Range	VB3648 - VB4095

To be compatible with Kinco-K3, K5 will take VB2684-VB3902 in the data backup area as effective when creating new project, which means data after VB3902 cannot be backed up. If you would like to have areas after VB3903 to be the backup area, you may set in [PLC Hardware Configuration]→[Others]. Please see below figure:



- [Permanently Save VB3648-4095]

If this item is clicked, VB3648-4095 will become the data backup area.

- [Save the Whole Project to PLC Permanent Register]

It is default in K5 to only save the information of hardware configuration and project info. If this item is clicked all information will be saved in PLC.

Appendix D Error Diagnose

K5 has three levels of errors: fatal error, serious error and normal error. When an error occurs PLC will take measures according to the level and record the error code by time sequence for future analysis. PLC will record a same error maximum 4 times.

Regardless of the level, we strongly suggest you to analyze and check after error occurs at your own risk.

1. Error Level

➤ Fatal Error

Fatal error occurs when PLC detects the chips are encountering unexpected stoppage. A fatal error may cause breaking down of PLC and further errors. The solution to fatal errors is to make PLC into safety status.

When a fatal error occurs PLC will automatically quit normal scanning and reset or enter into independent safe sub-OS according to SM2.0. SM2.0 decides the actions when fatal error occurs: if the value is 0, the PLC will enter into safe sub-OS; if 1 PLC will reset and reboot.

Below are descriptions of safety status:

- All outputs (DO and AO) will output the value defined in [PLC Hardware Configuration].
- ERR and STOP indicators flash, indicating a fatal error occur.
- Record error point and code and allow you to record the information with certain software. NOTE: Fatal errors will cause PLC unable to run normally, which can be recorded.

➤ Serious Error

A serious error will cause PLC unable to execute some important functions but the results are within expectation. If serious errors occur PLC will take measures automatically:

- Set PLC to STOP status, all outputs (DO and AO) output the “Stop and Output” value accordingly.
- ERR and STOP indicator long flash.

➤ Record the error code and allow you to read the records through KincoBuilder and Modbus RTU protocol.

➤ **Normal Error**

Normal error occur when PLC execute some functions but PLC is able to run other program. The results are within expectation. PLC will take measures as follows:

➤ PLC continues running.

➤ ERR and STOP indicator long flash.

➤ Record the error code and allow you to read the records through KincoBuilder and Modbus RTU protocol.

2. Error codes

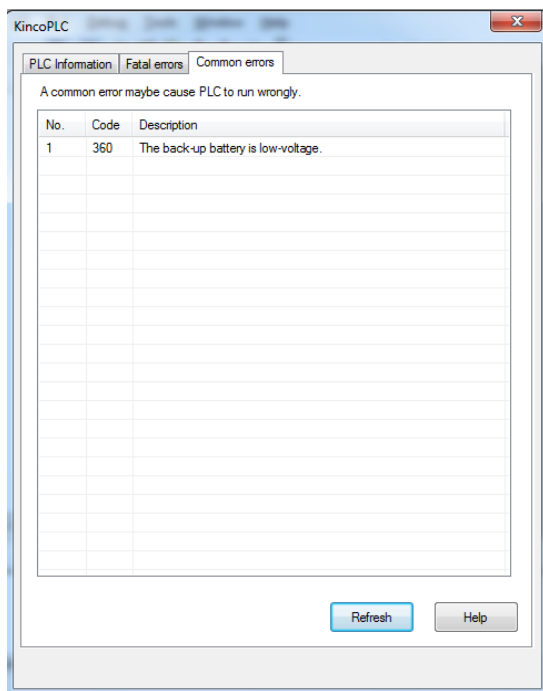
Code	Description
Fatal Error	
20	CPU type of [PLC Hardware Configuration] not the same with that actual type connects
21	Wrong extension module in [PLC Hardware Configuration]
25	When powered on, fail to read PLC protection type.
26	When powered on, fail to read clear file.
27	When powered on, fail to read cipher file.
28	When powered on, fail to check CRC of target file.
29	When powered on, PLC detects unknown command.
30	When powered on, number of parameters out of limitation.
35	When powered on, fail to read data from permanent.
40	Fail to execute JMP command.
41	Fail to call sub-programme.
42	Fail to call interruption sub-programme.
60	When powered on, no response from the 1 st extension module due to out of limitation.
61	When powered on, the 1 st extension responds error.
62	The 1 st extension module sype the same with that of hardware configuration.
65	When powered on, no response from the 2 st extension module due to out of limitation.
66	When powered on, the 2 st extension responds error.
67	The 2 nd extension module sype the same with that of hardware configuration.
70	When powered on, no response from the 3 rd extension module due to out of limitation.
71	When powered on, the 3 rd extension responds error.
72	The 3 rd extension module sype the same with that of hardware configuration.
75	When powered on, no response from the 4 th extension module due to out of limitation.
76	When powered on, the 4 th extension responds error.
77	The 4 th extension module sype the same with that of hardware configuration.
80	When powered on, no response from the 5 th extension module due to out of limitation.
81	When powered on, the 5 th extension responds error.
22	The 5 th extension module sype the same with that of hardware configuration.

85	When powered on, no response from the 6 th extension module due to out of limitation.
86	When powered on, the 6 th extension responds error.
87	The 6 th extension module sypte the same with that of hardware configuration.
95	When powered on, CPU fails to send extension report.
96	When powered on, CPU extension bus enters error passive status.
97	When powered on, CPU extension bus enters bus closure status.
Normal Error	
136	When powered on, adjustment of AI tunnel fails to be read.
137	When powered on, adjustment of AO tunnel fails to be read.
138	When adjusted, adjustment of AI tunnel fails to be read.
139	When adjusted, adjustment of AO tunnel fails to read.
300	When running, body AI tunnel has DMA error.
301	When running, body AI tunnel stopped when conducting sample conversion.
320	When running, extension bus communication has frame format error.
321	When running, extension bus communication has entered into error active status.
322	When running, extension bus communication has entered into error passive status.
323	When running, extension bus has been closed.
324	When running, extension communication error: receiving buffer area full.
325	When running, extension communication error: sending buffer area full.
326	When running, extension communication error: sending report fail.
327	When running, detect CANOpen salve station error (heartbeat or node protection out of time, SDO no response, etc.)
329	When running, error occurs: divided by zero
330	When running, error occurs: type conversion commands (I_TO_B, DI_TO_I) overflow
331	When running, error occurs: LN command set to 0 or negative numbers.
332	When running, error occurs: LOG command set to 0 or negative numbers.
333	When running, error occurs: SQRT command set to 0 or negative numbers.
334	When running, error occurs: I_TO_BCD command has invalid input value.
335	When running, error occurs: A_TO_H command has invalid input value.
336	When running, error occurs: R_TO_A command has invalid input value.
341	When running, error occurs:FOR command has invalid input value.
350	When running, error occurs: fail to save permanent data.

351	When powered on, power-failure in RAM data missing
360	Low backup battery.

3. How to Read Errors Occur Before

PLC will automatically record the error code. You may read the error with ways as follows:



➤ Through KincoBuilder

You may execute [PLC]→[PLC Serious Error] or [PLC Normal Error] in KincoBuilder PORT0 and check as per above figure shows. You may click [Refresh] button to refresh the information.

➤ Through Modbus RTU

User may use Modbus RTU communication command (function code 03 and 04) to record error information through PORT0, PORT1 and PORT2._

Register addresses are as follow:

Modbus Register	Description

9000-9127	After this time of powered on, the latest 128 normal error codes. Among which 9000 is the latest error and 9001 the second.
9128-9255	After this time of powered on, the latest 128 normal error codes. Among which 9128 is the latest error and 9129 the second.
9256-9383	Last time of powered on, the latest 128 normal error codes. Among which 9256 is the latest error and 9257 the second.
9384-9511	Last time of powered on, the latest 128 normal error codes. Among which 9384 is the latest error and 9385 the second.

4. Error Register

K5 provides a error register in SM and will record error when error occurs. You may directly read the register.

➤ SMB2: control bytes

Bit (can be read and wrote)	Description
SM2.0	Its value decides the actions when fatal error occurs. Initial value is 0. If it is 0: PLC enters into independent sub-OS to run safely. If it is 1: PLC reset when fatal error occur.

■ SMB0 and SMB1: memory and command error

SM	Description
SMB0 (read only)	
SM0.2	If PLC detects power failure data in RAM missing, this bit will be set to 1 otherwise to 0.
SMB1 (read only)	
SM1.0	1 represents errors occur: DIV and MOD divided by zero.
SM1.1	1 represents errors occur: LN, LOG and SQRT command is invalid (0 or negative number)
SM1.2	1 represents errors occur: I_TO_B, DI_TO_I overflow
SM1.3	1 represents errors occur: I_TO_BCD invalid BCD code input
SM1.4	1 represents errors occur: A_TO_H input alphabet string has undefined bytes
SM1.5	1 represents errors occur: R_TO_A represents conversion result overflow.
SM1.6	1 represents errors occur: FOR input parameter invalid

■ SMB3 和 SMB96-SMB110: extension module error

If PLC detects any communication error or sending error reports in extension bus, it will set corresponding extension bus and place the error code in the error register for future checking. If no errors are detected then the bit and register will be set to 0.

NOTE: If PLC detects error in extension bus or extension module, it will enters STOP status and lit up ERR indicator. It will not set corresponding register due to no executions by CPU.

SM	Description	
SMB3 (read only): extension bus error signal		
SMB3.0		If the 1 st extension module has errors it will be set to 1.
SMB3.1		If the 2 nd extension module has errors it will be set to 1.
SMB3.2		If the 3 rd extension module has errors it will be set to 1.
SMB3.3		If the 4 th extension module has errors it will be set to 1.
SMB3.4		If the 5 th extension module has errors it will be set to 1.
SMB3.5		If the 6 th extension module has errors it will be set to 1.
SMB3.7		If CPU detects error in extension bus communication it will be set to 1.
SMB96 - SMB103 (read only): extension bus error code		
SMW96	SMB96	If the 1 st extension module has error then the code will be saved here.
	SMB97	Error code of report is saved here
SMW98	SMB98	If the 1 st extension module has error then the code will be saved here.
	SMB99	Error code of report is saved here
SMW100	SMB100	If the 1 st extension module has error then the code will be saved here.
	SMB101	Error code of report is saved here
SMW102	SMB102	If the 4 th extension module has error then the code will be saved here.
	SMB103	Error code of report is saved here
SMW104	SMB104	If the 5 th extension module has error then the code will be saved here.
	SMB105	Error code of report is saved here
SMW106	SMB106	If the 6 th extension module has error then the code will be saved here.
	SMB107	Error code of report is saved here
SMB110	SMB110	Error code of CUP extension bus error is saved here.

Error Code	Description
0	No error.

6	Heartbeat report out of time.
7	CPU receives emergency report in extension module Error code please refer to following figure 2

Figure 1 Extension Error

Error Code	Description
0	No error.
2	CAN controller enters into Error Passive
3	CAN controller enters into Bus Off
5	PDO length error
6	Other reports except PDO length error
10	ADC of AI tunnel conversion error
11	Adjustment value saving error
12	Adjustment value reading error
14	1st tunnel input signal of stimulation module out of measurement
15	2nd tunnel input signal of stimulation module out of measurement
16	3rd tunnel input signal of stimulation module out of measurement
17	4th tunnel input signal of stimulation module out of measurement

Figure 2 Error code in emergency report

Code	Description
0	No error
1	Communication frame format error
2	Extension bus enters into error warning status
3	Extension bus enters into error passive status
4	Extension bus enters into bus shutdown status and just recovered
5	Extension bus receiving buffer area full
6	Extension bus sending buffer full
7	CPU fails to send the report.

Figure 3 Error code of CPU extension bus communication error

Appendix E Definition of SM Area

This Appendix describe the definitions of system register area, SM area. This area is used to assist Kinco-K5 to realize certain functions. You may also use it to read PLC status.

1. SMB0: system status byte

SM0.0-SM0.7 is valued by CPU software and cannot be controlled by user. You can only call some functions (read only):

Bit (read only)	Description
SM0.0	Always be "1"
SM0.1	First scanning bit. It is "1" when first scanning and be clear afterwards. Used when initialization.
SM0.2	If the power failure data in RAM missing then set this bit to 1 and the clear.
SM0.3	Pulse string with a cycle of 1s, 50%
SM0.4	Pulse string with a cycle of 2s, 50%
SM0.5	Pulse string with a cycle of 4s, 50%
SM0.6	Pulse string with a cycle of 60s, 50%

2. SMB2: system control byte

Bit (read only)	Description
SM2.0	Its value decides the actions taken when fatal errors occur. Initial value is 0. If it is 0: PLC enters into sub-OS to run safely If it is 1: PLC resets
SM2.1	Its value decides the status of AI/AO tunnels. Initial value is 0. If it is 0: body AI/AO tunnel run normally If it is 1: body AI/AO tunnel enters into adjustment

3. Communication Port Reset

K5 provides function to reset communication port (PORT0, PORT1 and PORT2). After reset K5 will clear the buffer area of the communication ports and start initialization. After reset the parameters and functions of the port will remain the same.

➤ **Control Register and Status Register**

Bit			Value	Description
PORT 0	PORT 1	PORT 2		
SM87.0	SM187.0	SM287.0	1	Set the two bits to certain value and use RCV command.
SM87.7	SM187.7	SM287.7	0	
SM4.0	SM4.1	SM4.2	-	After success reset K5 will value the bit to 1. It requires manual reset

- Reset (PORT0 as example)
- (Optional) Set SM4.0 to 0.
- Set SM87.7 to 0 and SM87.0 to 1.
- Call RCV and set its PORT parameters as the communication number. Need to be valued to 0.
- (Optional) Check SM4.0. If 1 means success reset.

➤ Example

(* Network 0 *)

(*Use rise chang of I0.0 to trigger PORT1 resetting*)

(*RCV command will not effect the resetting.*)

LD %I0.0

R_TRIG

MOVE B#0, %SMB4

AND B#16#7F, %SMB187

OR B#16#1, %SMB187

RCV %VB222, 1

(* Network 1 *)

(*After resetting you may wait until PORT1 stays stable and continue operation. *)

(*PORT1 will enter into receiving status after resetting*)

LD %SM4.1

TON T4, 3

OR B#16#80, %SMB187

RCV %VB222, 1

R %SM4.1

4. Other functional variables

SM	Description
SMB6	Read-only. Save the last PLC scanning time. Unit: ms.
SMW10	Read-only. Save voltage of back-up battery. Unit: 0.01V. If the power supply of back-up battery be lower than 2.6V constantly, PLC will warn “Low Back-up Battery”
SMB274-SMB285	CPU’s sole ID

5. SMD12 and SMD16: Timer Interruption Cycle

K5 can provide two Timer Interruptions based on 0.1 ms: Timer Interruption 0 with numbered 3; Time Interruption 1 with numbered 4.

SMD12 is used to define the cycle of Timer Interruption 0, with a unit of 0.1 ms. If SMD12 is set to 0 then Timer Interruption 0 will be forbidden. The default value of SMD12 is 0;

SMD16 is used to define the cycle of Timer Interruption 1, with a unit of 0.1 ms. If SMD12 is set to 1 then Timer Interruption 0 will be forbidden. The default value of SMD16 is 0;

Timer Interruption will generate periodically and you may use it to complete periodical tasks. Timer Interruption will not affected by PLC scanning period and can be used for precise timing.

Kinco[®] Automation

<http://www.kinco.cn> Email:sales@kinco.cn

(All trademarks and logos in this brochure are property of and registered by their respective owners.)