Contents

1 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-K3 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 expend 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-K3 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 ong 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 E^2PROM or FRAM through relevant instruction for permanent storage. *Notice: Every type of permanent memory has its own expected life, for example, E^2PROM allows 100 thousand of times of writing operations and FRAM allows 10 billions of times.*

2 Chapter II Concepts for Programming

This chapter will detailedly introduce the fundamentals for programming KINCO-K3 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 mannual.

2.1 POU (Programme Orgnization 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 mannual.

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 POUs that are either provided by the manufacturer or created by the user. POUs 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 POUs cannot call themselves either directly or indirectly

2.2 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-K3 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} ~ $(2^{15}-1)$	0
DINT	Signed Double integer	32	-2^{31} ~ $(2^{31}-1)$	0
REAL	Floating-point number, ANSI/IEEE 754--1985 standard format	32	$1.18*10^{-38}$ ~ $3.40*10^{38}$, $-3.40*10^{38}$ ~ $-1.18*10^{-38}$	0.0

Table 2-1 Elementary Data Types that the KINCO-K3 supports

2.3 Identifiers

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

2.3.1 How to define an identifier

You must comply with the following principles while difining 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.

2.3.2 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.

2.4 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-K3 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	$1.18*10^{-38}$ ~ $3.40*10^{38}$, $-3.40*10^{38}$ ~ $-1.18*10^{-38}$	1.0, -243.456
	xEy		-2.3E-23

Table 2-2 Constants

Notice:

(1) The descriptor is not case-sensitive, e.g. the constants W#234 and w#234 shall be the same.

(2) 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.

2.5 Variables

In contrast to *constants*, *variables* provide a means of identifying data objects whose contents may change, 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.

2.5.1 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-K3 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 decalred, 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 POUs.

Table 2-3 Variable Types

2.5.2 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.

2.5.3 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.

2.6 How to Access PLC Memory

The KINCO-K3 stores information in different memory units. To be convenient for the users, the KINCO-K3 provides two addressing methods to access the memory units:

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

2.6.1 Memory Types and Characteristics

The memory of the KINCO-K3 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-K3 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-K3 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-K3 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-K3 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 2-4 Memory Types and Characteristics

2.6.2 Direct Addressing

Direct addressing meas 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 (.). 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.

2.6.2.1 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	**%I***x.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	**%IB***x*
	Description	*x*: byte address of the variable
	Data type	BYTE
	Example	%IB0 %IB1 %IB5
Word Addressing	Format	**%IW***x*
	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	**%ID***x*
	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	**%Q***x.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	**%QB***x*
	Description	*x*: byte address of the variable
	Data type	BYTE
	Example	%QB0 %QB1 %QB4
Word Addressing	Format	**%QW***x*
	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	**%QD**x
	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	**%AIW***x*
	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	**%AQW***x*
	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	*x*: byte address of the variable *y*: 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	*x*: byte address of the variable
	Data type	BYTE
	Example	%MB0 %MB1 %MB10
Word Addressing	Format	**%MW***x*
	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	%MW0 %MW2 %MW12
Double word Addressing	Format	**%MD***x*
	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	%MD0 %MD4 %MD12

- **V Area**

Bit Addressing	Format	**%V***x.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	%V0.0 %V0.7 %V5.6
Byte Addressing	Format	**%VB***x*
	Description	*x*: byte address of the variable
	Data type	BYTE
	Example	%VB0 %VB1 %VB10

Word Addressing	Format	**%VW***x*
	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	%VW0 %VW2 %VW12
Double word Addressing	Format	**%VD***x*
	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	%VD0 %VD4 %VD12
REAL Addressing	Format	**%VR***x*
	Description	*x*: starting byte address of the variable. Since the size of REAL is 4 bytes, *x* must be an even number.
	Data type	REAL
	Example	%VR0 %VR4 %VR1200

- **SM Area**

Bit Addressing	Format	**%SM***x.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	%SM0.0 %SM0.7 %SM5.6
Byte Addressing	Format	**%SMB***x*
	Description	*x*: byte address of the variable
	Data type	BYTE
	Example	%SMB0 %SMB1 %SMB10
Word Addressing	Format	**%SMW***x*
	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	%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	**%L***x.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	**%LB***x*
	Description	*x*: byte address of the variable
	Data type	BYTE
	Example	%LB0 %LB1 %LB10
Word Addressing	Format	**%LW***x*
	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	**%LD***x*
	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

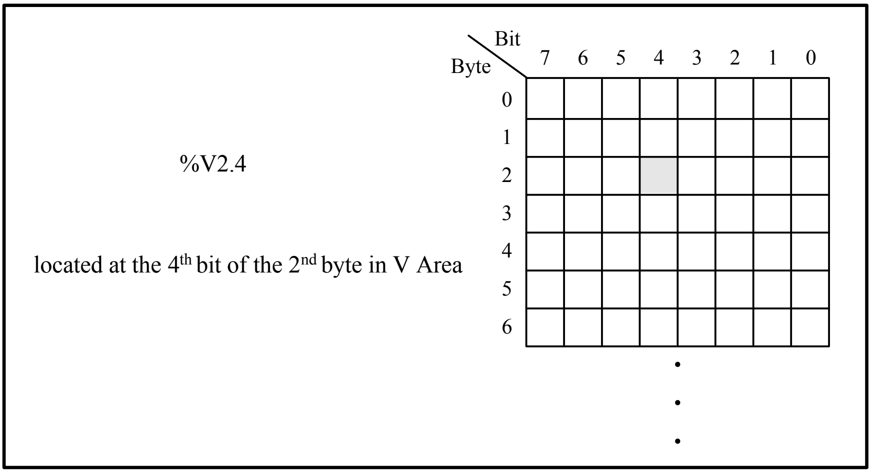
- **HC Area**

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

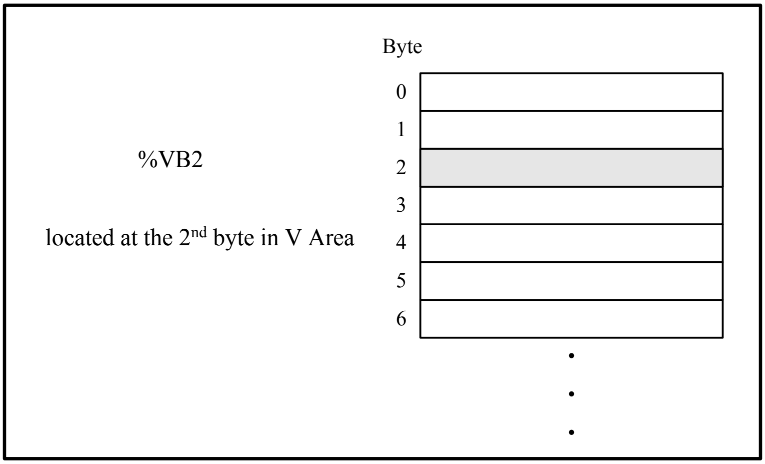
2.6.2.2 Mapping between Direct Address and PLC Memory Location

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.

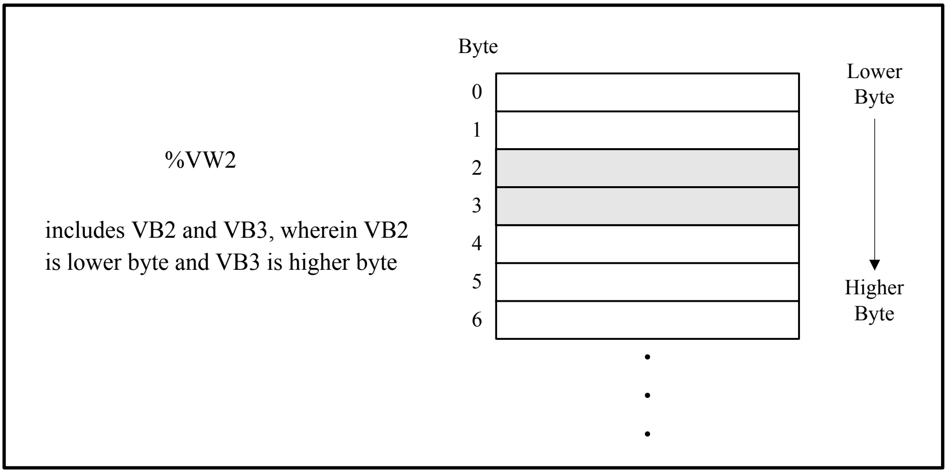
- **Bit Addressing**



- **Byte Addressing**



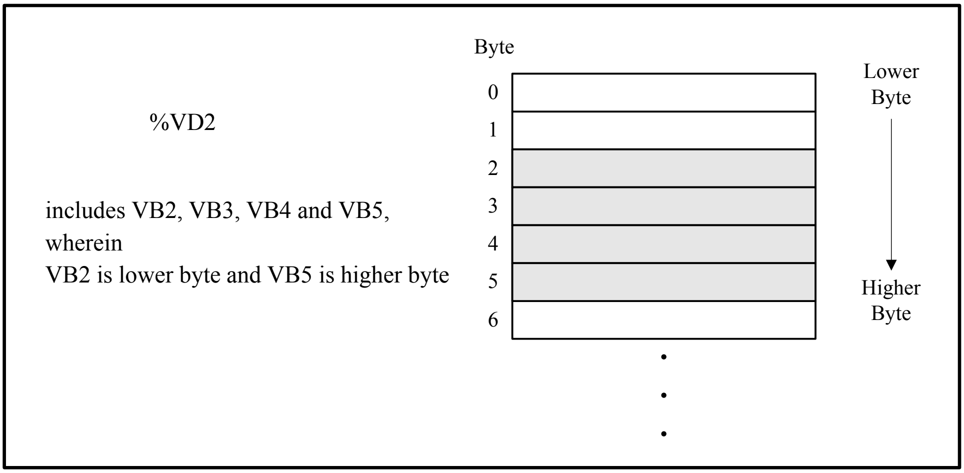
- **Word Addressing**



%VW2

includes VB2 and VB3, wherein VB2 is lower byte and VB3 is higher byte

- **Double word Addressing**



%VD2

includes VB2, VB3, VB4 and VB5, wherein
VB2 is lower byte and VB5 is higher byte

2.6.3 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-K3 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.

Note: Only the CPU306Ex and the CPU308 support the indirect ddressing method.

2.6.3.1 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
```

2.6.3.2 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, pleae pay attention to the data types of the instructin’s operands.

For example:

```
LD          %SM0.0
MOVE     &VB0, %VD200       (*Create a pointer (VD200) which points to VW2. *)
MOVE     *VD200, %VB10      (* Assign the value of VB0 to VB10. The pointer VD200 points to VB0, *)
                            (* so *VD200 represents VB0. *)
```

2.6.3.3 Modifying the value of a pointer

A pointer is a 32-bit variable, and so it's 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.

2.6.3.4 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-K3 only supports single-level pointer and address, multiple-level pointers and addresses are illegal. For example, the following instruction is illegal:

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

2.6.3.5 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 *)
```

2.6.4 Memory Address Ranges

The KINCO-K3 provides several types of CPU module. The memory address ranges of different types of CPU may be different from each other, and the addresses byond 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.

		CPU304	CPU304EX, CPU306	CPU306EX, CPU308
I	Size	2 bytes	8 bytes	32 bytes
	Bit address	%I0.0 --- %I1.7	%I0.0 --- %I7.7	%I0.0 --- %I31.7
	Byte address	%IB0、IB1	%IB0 --- %IB7	%IB0 --- %IB31
	Word address	%IW0	%IW0 ---% IW6	%IW0 ---% IW30
	Double-word address	———	%ID0 --- %ID4	%ID0 --- %ID28
Q	Size	2 bytes	8 bytes	32 bytes
	Bit address	%Q0.0 --- %Q0.7	%Q0.0 --- %Q7.7	%Q31.0 --- %Q31.7
	Byte address	%QB0	%QB0 --- %QB7	%QB0 --- %QB31
	Word address	———	%QW0 --- %QW6	%QW0 --- %QW30
	Double-word address	———	%QD0 --- %QD4	%QD0 --- %QD28
AI	Size	0	32 bytes	64 bytes
	Word address	———	%AIW0 --- %AIW30	%AIW0 --- %AIW62
AQ	Size	0	32 bytes	64 bytes
	Word address	———	%AQW0 -- %AQW30	%AQW0 -- %AQW62
HC	Size	8 bytes	24 bytes	
	Word address	%HC0，%HC1	%HC0 --- %HC5	
V	Size	2048 bytes	4096 bytes	
	Bit address	%V0.0 --- %V2047.7	%V0.0 ---%V4095.7	
	Byte address	%VB0 --- %VB2047	%VB0 --- %VB4095	
	Word address	%VW0 --- %VW2046	%VW0 --- %VW4094	
	Double-word address	%VD0 --- %VD2044	%VD0 --- %VD4092	
	REAL address	%VR0 --- %VR2044	%VR0 --- %VR4092	
M	Size	32 bytes		
	Bit address	%M0.0 --- %M31.7		
	Byte address	%MB0 --- %MB31		
	Word address	%MW0 --- %MW30		

	Double-word address	%MD0 --- %MD28
SM	Size	300 bytes
	Bit address	%SM0.0 --- %SM299.7
	Byte address	%SMB0 --- %SMB299
	Word address	%SMW0 --- %SMW298
	Double-word address	%SMD0 --- %SMD296
L	Size	272 bytes
	Bit address	%L0.0 --- %L271.7
	Byte address	%LB0 --- %LB271
	Word address	%LW0 --- %LW270
	Double-word address	%LD0 --- %LD268

Table 2-5 CPU Memory Ranges

2.6.5 Function Block and Function Block Instance

2.6.5.1 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

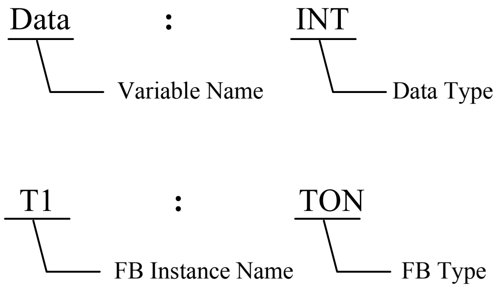
2.6.5.2 Instances of Function Blocks

"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.



Data is an instance of *INT* type, and *T1* is an instance of *TON*.

2.6.5.3 FB Instance Memory Areas

A fixed memory area is allocated for each type of FB to store its instances in the KINCO-K3 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 SRcan 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 2-6 FB Instance Memory Areas

2.6.6 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	*x*: a decimal digit, indicating the timer number.
Data type	BOOL --- status bit of the timer INT --- current value of the timer **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.
Example	T0 T5 T20

- **C**

Format	**C***x*

Description	*x*: a decimal digit, indicating the counter number.
Data type	BOOL --- status bit of the counter INT --- current counting value of the counter **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.
Example	C0 C5 C20

- **RS**

Format	**RS***x*
Description	*x*: 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	*x*: a decimal digit, indicating the SR Bistable number.
Data Types	BOOL --- the status of the SR Bistable
Example	SR0, SR5, SR10

2.6.7 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-K3 CPU allocates a different memory range for the FB instances. The detailed descriptions are given in the following table.

		CPU304	**CPU304EX, CPU306**	**CPU306EX, CPU308**
T	Amount	64	128	256

	Range	T0 --- T63	T0 --- T127	T0 --- T255
	Resolution	T0 --- T3: 1ms T4 --- T19: 10ms T20 --- T63: 100ms	T0 --- T3: 1ms T4 --- T19: 10ms T20 --- T127: 100ms	T0 --- T3: 1ms T4 --- T19: 10ms T20 --- T255: 100ms
	Max timing	32767* Resolution	32767* Resolution	32767* Resolution
C	Amount	64	128	256
	Range	C0 --- C63	C0 --- C127	C0 --- C255
	Max counting value	32767	32767	32767
RS	Amount	———		32
	Range	———		RS0 --- RS31
SR	Amount	———		32
	Range	———		SR0 --- SR31

Table 2-7 FB Instances Memory Ranges

3 Chapter III How to Use KincoBuider … 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-K3 PLC. The purpose of this chapter is to give you a quick guide, and further details will be presented in the following chapter.

3.1 Computer Requirements

KincoBuilder runs on a personal computer. The following is the minimum requirements for your computer:

- CPU: 133 MHz or higher
- Hard disk: at least 10M bytes of free space
- RAM: 32M or more
- Keyboard, mouse, a serial communication port
- 256-color VGA or higher, 1024*768,
- Operating system: English version Windows NT 4.0 (or later version)/Windows 2000/Windows XP

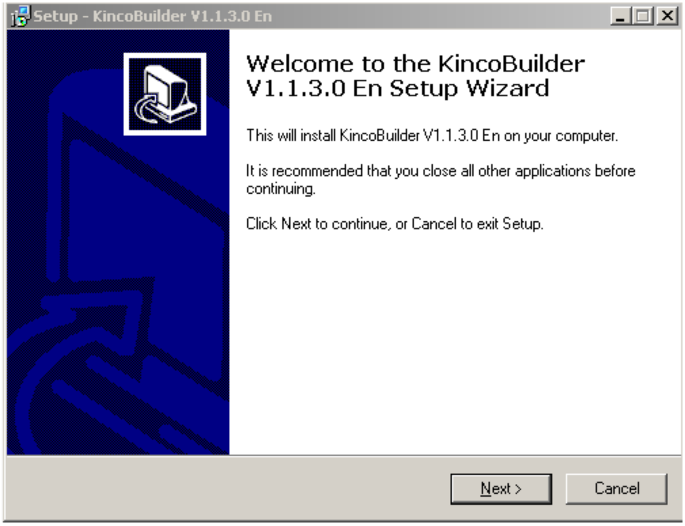
3.2 Install/Uninstall

3.2.1 Installing KincoBuilder

If you have an earlier version of KincoBuilder installed in your system, uninstall it before installing new version.

You can click *Cancel* at any step to exit setup.

① Run **KincoBuilderV*xxxx*_setup.exe** (*xxxx* represents the version number, e.g. 1930) to launch the setup wizard as shown in Figure 3-1:



Setup - KincoBuilder V1.1.3.0 En

Welcome to the KincoBuilder V1.1.3.0 En Setup Wizard

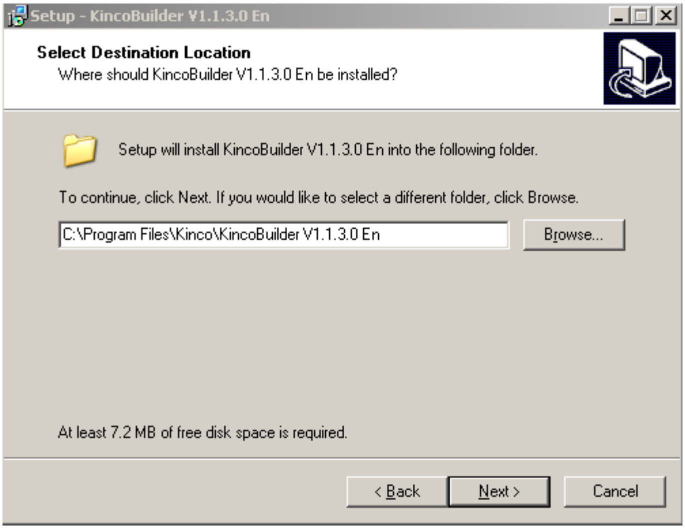
This will install KincoBuilder V1.1.3.0 En on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Figure 3-1

② Click *Next*, continue to select the path. You can either choose the default path or modify it, as shown in Figure 3-2.



Setup - KincoBuilder V1.1.3.0 En

Select Destination Location

Where should KincoBuilder V1.1.3.0 En be installed?

Setup will install KincoBuilder V1.1.3.0 En into the following folder.

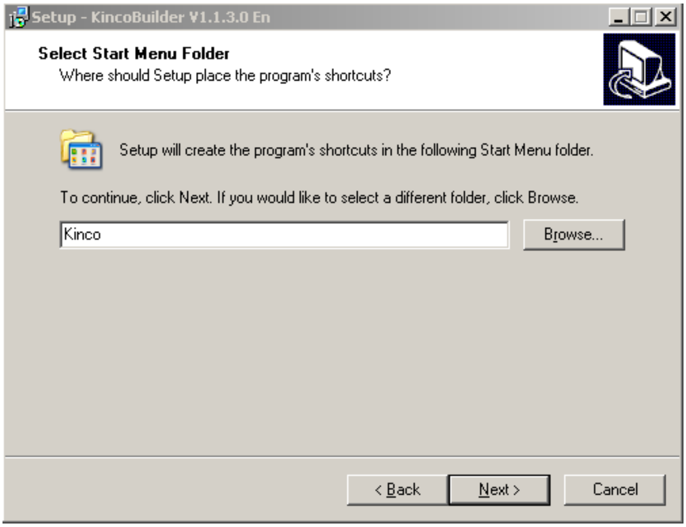
To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\Kinco\KincoBuilder V1.1.3.0 En

At least 7.2 MB of free disk space is required.

Figure 3-2

③ Click *Next*, continue to select a Start Menu folder to save the shortcut, the default folder is “KINCO”, as shown in Figure 3-3:



Setup - KincoBuilder V1.1.3.0 En

Select Start Menu Folder

Where should Setup place the program's shortcuts?

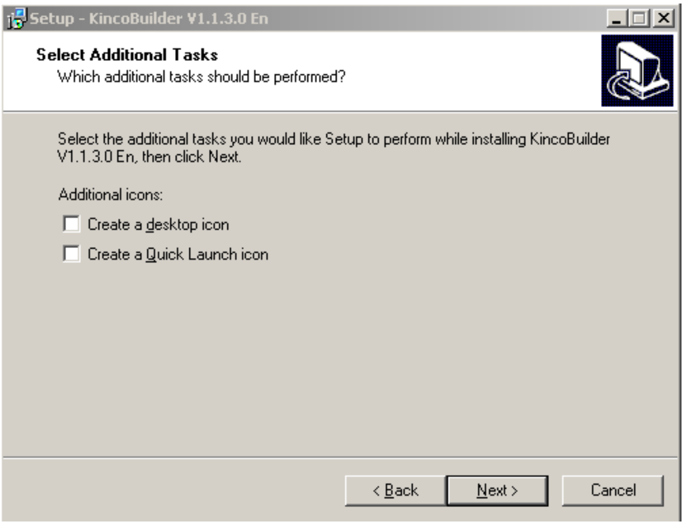
Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

Kinco

Figure 3-3

④ Click *Next*, continue to confirm whether to create a desktop icon or a quick launch icon, as shown in Figure 3-4:



Setup - KincoBuilder V1.1.3.0 En

Select Additional Tasks

Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing KincoBuilder V1.1.3.0 En, then click Next.

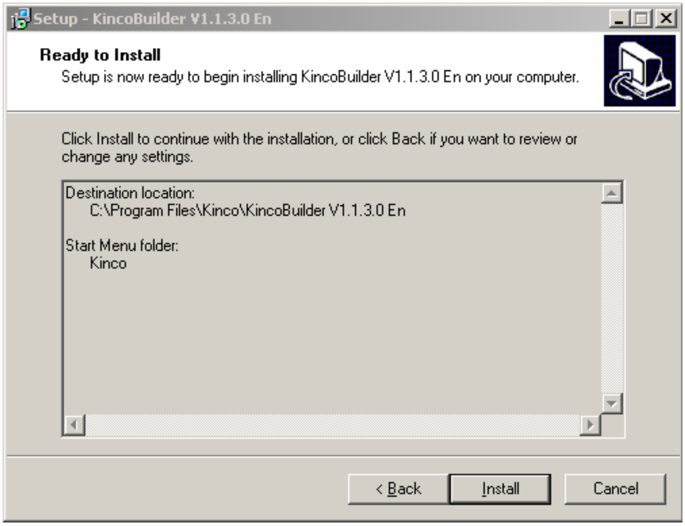
Additional icons:

□ Create a desktop icon

□ Create a Quick Launch icon

Figure 3-4

⑤ Click *Next*, the wizard will prompt Ready to Install, as shown in Figure 3-5:



Ready to Install

Setup is now ready to begin installing KincoBuilder V1.1.3.0 En on your computer.

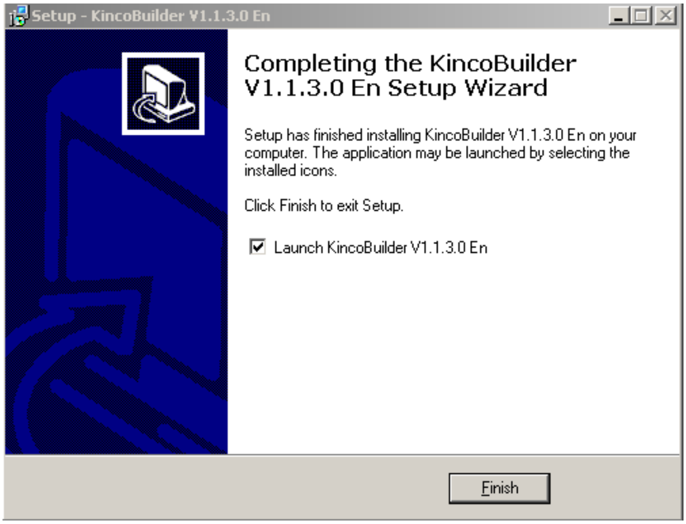
Click Install to continue with the installation, or click Back if you want to review or change any settings.

Destination location:
C:\Program Files\Kinco\KincoBuilder V1.1.3.0 En

Start Menu folder:
Kinco

Figure 3-5

⑥ Click *Install*, KincoBuilder shall be installed on your computer, and there will be a prompt after the installation is finished, as shown in Figure 3-6:



Completing the KincoBuilder V1.1.3.0 En Setup Wizard

Setup has finished installing KincoBuilder V1.1.3.0 En on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

☑ Launch KincoBuilder V1.1.3.0 En

Figure 3-6

⑦ Click *Finish* to finish the installation.

If you check *Launch KincoBuilder* simultaneously, KincoBuilder will be launched immediately.

3.2.2 Uninstalling KincoBuilder

Please exit KincoBuilder before uninstalling it.

There are two ways to uninstall KincoBuilder:

➢ Click the [**Start**] button and choose [**Programs**] > [**KINCO**] > [**Uninstall KincoBuilder**].

KincoBuilder files will be removed automatically.

➢ Select [**Start**] > [**Settings**] > [**Control Panel**];

Open the [**Control Panel**] and double-click [**Add/Remove Programs**];

Select [KincoBuilder V*x.x.x.x*] (*x.x.x.x* presents version number) and click the [**Add/Remove**] button.

KincoBuilder files will be removed.

3.3 How to Start and Exit KincoBuilder

3.3.1 How to Start KincoBuilder

There are two ways to start KincoBuilder:

- Click the [**Start**] button and choose [**Programs**]>[**KINCO**]>[**KincoBuilder**].
- If you have created a desktop icon during installation, double click the icon on the desktop.

3.3.2 How to Quit KincoBuilder

There are three ways to exit KincoBuilder:

- Select [**File**] > [**Exit**] menu command
- Use the shortcut key **Alt+F4**
- Click the icon ☒ on the top-right corner of the main KincoBuilder window.

3.4 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.

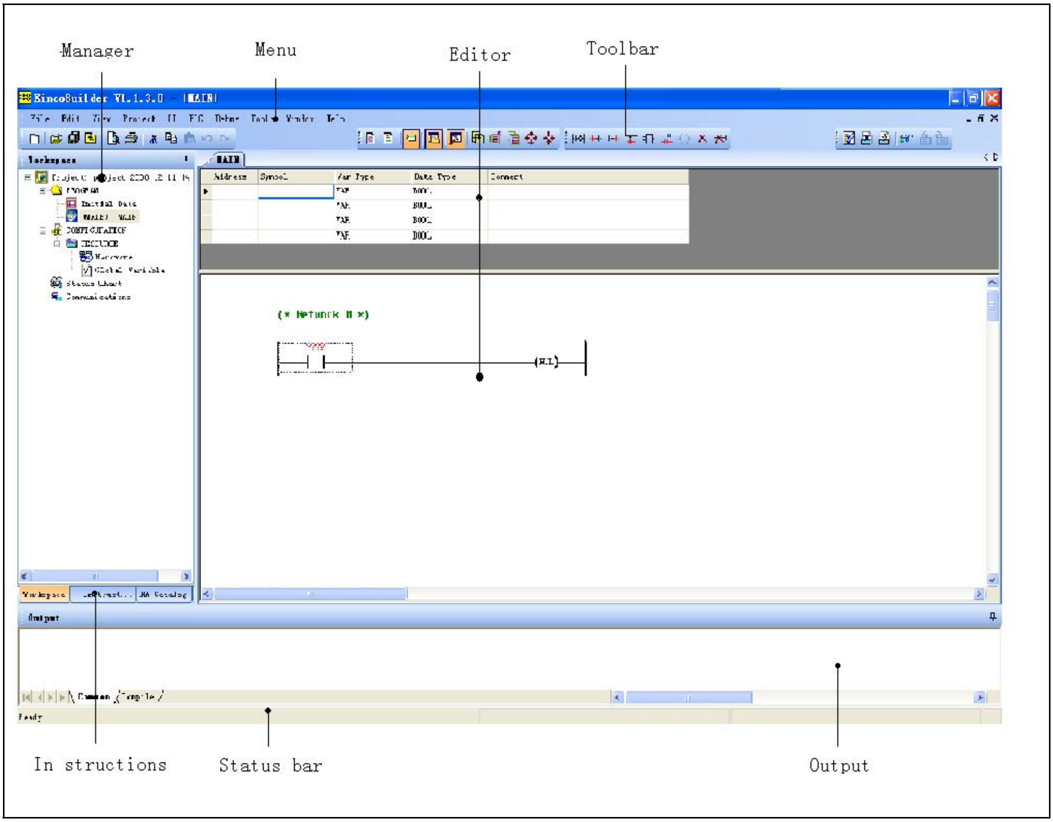


Figure 3-7 User Interface of KincoBuilder

▪ **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.

3.5 Using KincoBuilder to Create Programs for Your Applications

3.5.1 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 POUs, etc. In KincoBuilder, all of these data (including POUs, 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)	You can assign initial numerical values to 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.
	Main Program	It is the execution entry of the user program. The CPU module executes it once per scan cycle. Only 1 Main Program exists in a project.

	Interrupt routines (Optional)	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. At most 16 interrupt routines are allowed in a project.
	Subroutines (Optional)	The subroutines can only be executed when they are invoked by the main program or interrupt routines. 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. At most 16 subroutines are allowable in a project.
CONFIGURATION	Hardware	Here you can configure the KINCO-K3 modules used in your control system, including their addresses, function parameters, etc. The CPU module shall process the hardware configuration once at power on and then execute other tasks.
	Global variables (Optional)	Here you can declare the global variables required in the project.

Table 3-1 Project Components

3.5.2 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.

3.5.3 Importing and Exporting a Project

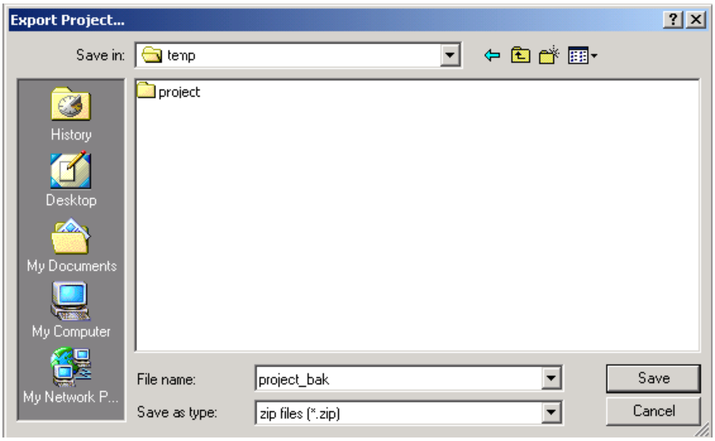
KincoBuilder provides [**File**]>[**Import**…] and [**File**]>[**Export**…] menu commands for you to backup and manage a project.

➢ [**Export**…]

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

① Select the [**File**]> [**Export**…] menu command.

The dialog box "Export Project…" appears, as shown in Figure 3-8.



Export Project...

Save in: temp

project

File name: project_bak

Save as type: zip files (*.zip)

Figure 3-8 Export the Project

② 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.

① Select the [**File**]> [**Import**…] menu command.

The dialog box "Import Project…" appears, as shown in Figure 3-9.

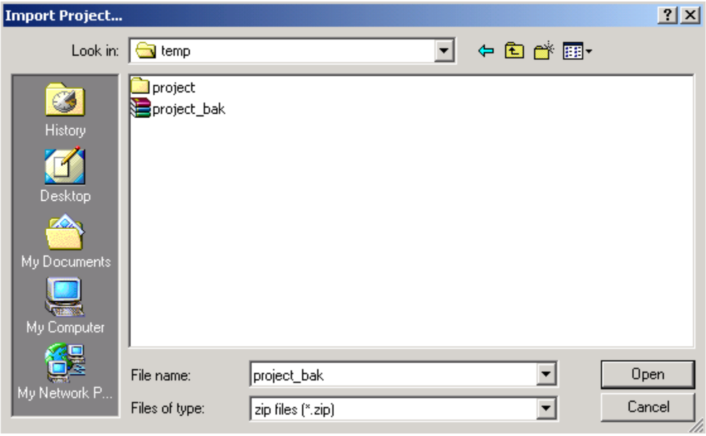


Figure 3-9 Import a Project: Select a backup file

② 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.

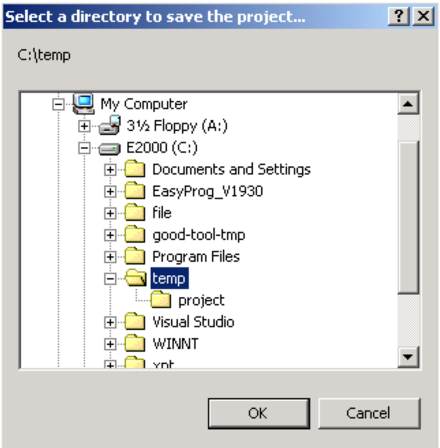


Figure 3-10 Import a Project: Select the destination directory

③ 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.

3.6 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 3-11:

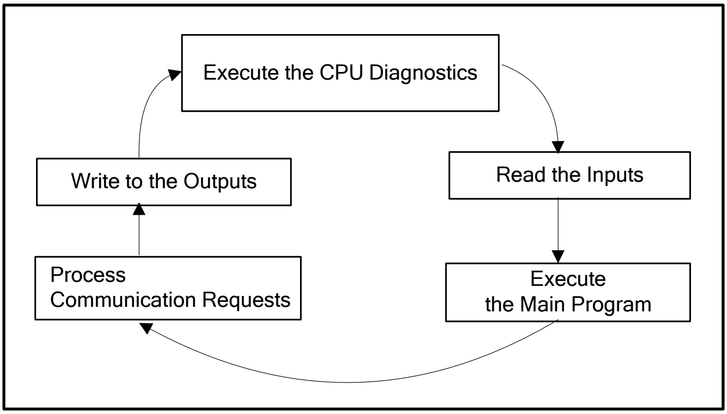


Figure 3-11 Scan Cycle

- ➢ 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-K3 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-K3 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.

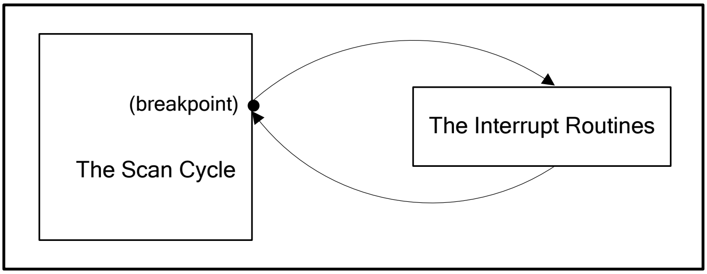


Figure 3-12 Execution of Interrupt Routines

3.7 How to connect the computer with the KINCO-K3

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-K3 PLC using KincoBuilder.

① 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.*

a) 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.

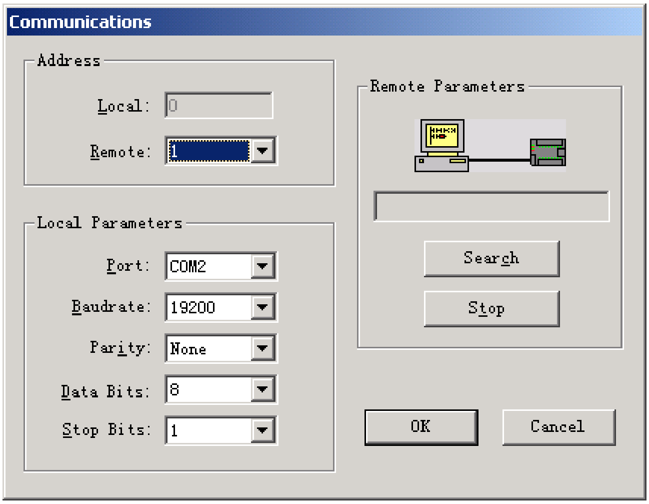


Figure 3-13 The “Communications ” Dialog Box

b) 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 co nfigure 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.*

③ After you have configured the communication parameters of the computer's COM port, you are ready to program the KINCO-K3 PLC.

3.8 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.

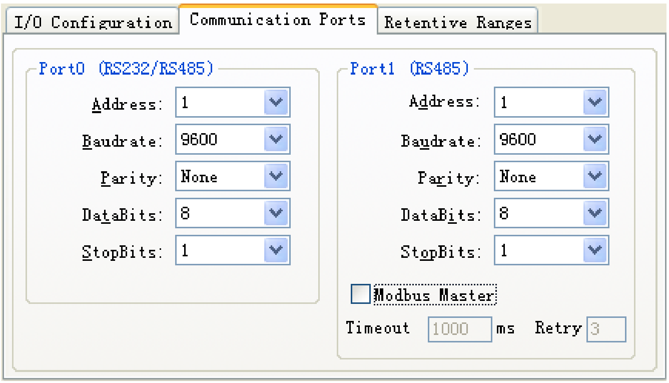
(1) First, open the "Hardware" window by using one of the following ways:

- 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.

(2) 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.



Port0 (RS232/RS485)

Address: 1
Baudrate: 9600
Parity: None
DataBits: 8
StopBits: 1

Port1 (RS485)

Address: 1
Baudrate: 9600
Parity: None
DataBits: 8
StopBits: 1

☐ Modbus Master

Timeout 1000 ms Retry 3

Figure 3-14 Communication Parameters

(3) 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.*

3.9 Example: Common Steps to Create a Project

In order to help the beginners to understand the KINCO-K3 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-K306-24DT CPU module;
- Control logic: Toggle Q0.0---Q0.7 in turn and cyclically. For better structure, we use two POUs: a subroutine named "Demo" to realize the control logic; the Main Program named "Main" in which "Demo" is invoked.

(1) Firstly, launch KincoBuilder.

(2) If necessary, modify the defaults used in KincoBuilder by using the following way:

- Select the [**Tools**]>[**Options**…] 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.

(3) Create a new project by using one of the following ways:

- Select the [**File**]>[**New 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 [**Save**], the new project shall be created.

For this example, let’s select “D:\temp” as the project directory, and name the project as “Example”.

(4) 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 3.8 How to modify the CPU’s communication parameters for detailed steps.

For this example, a KINCO-K306-24DT module with the default parameters is used.

(5) 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.

a) Main Program

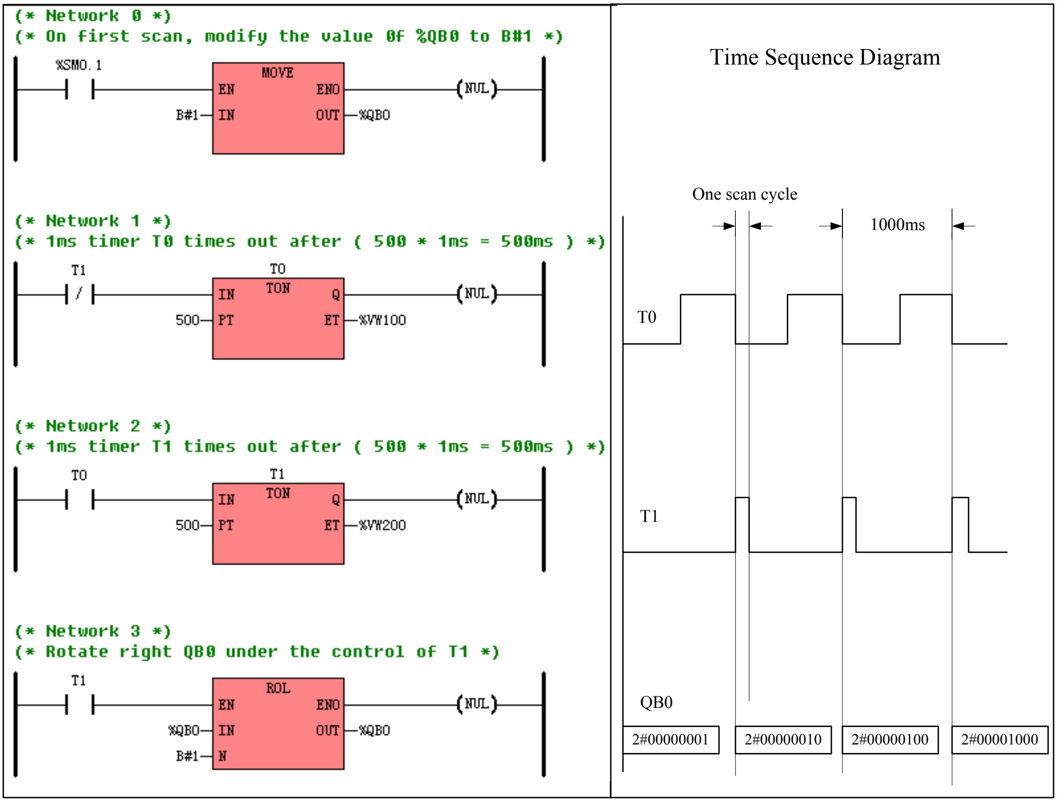
When creating a new project, KincoBuilder will automatically create an empty main program named "MAIN" at the same time.

b) 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 3-15.

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.



(* Network 0 *)
(* On first scan, modify the value 0f %QB0 to B#1 *)

(* Network 1 *)
(* 1ms timer T0 times out after (500 * 1ms = 500ms) *)

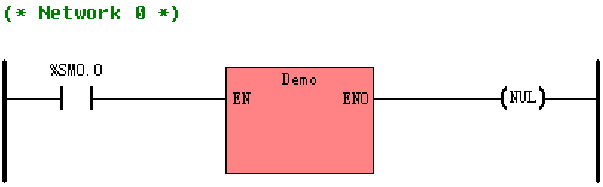
(* Network 2 *)
(* 1ms timer T1 times out after (500 * 1ms = 500ms) *)

(* Network 3 *)
(* Rotate right QB0 under the control of T1 *)

Figure 3-15 the Subroutine “Demo”

c) 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.



(* Network 0 *)

Figure 3-16 the Main Program

(6) 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.

(7) 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.

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**

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-K3 project.

(8) 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.

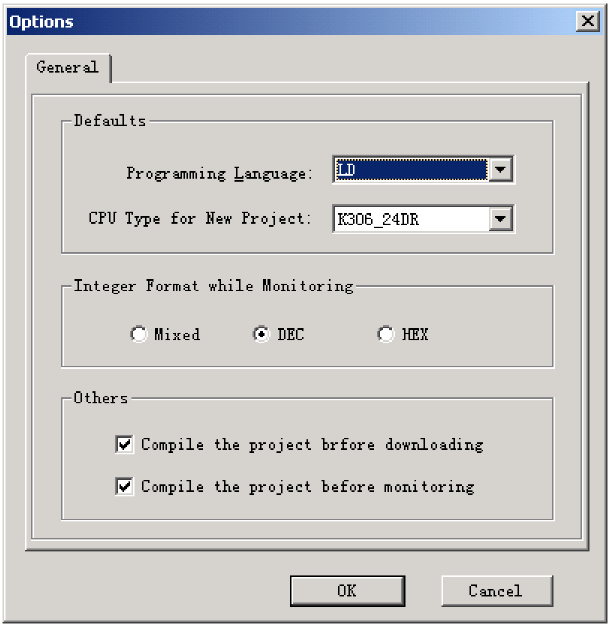
4 Chapter IV How to Use KincoBuilder … Basic Functions

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.

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:



Options

General

Defaults

Programming Language: LD

CPU Type for New Project: K306_24DR

Integer Format while Monitoring

○ Mixed ● DEC ○ HEX

Others

☑ Compile the project brfore downloading

☑ Compile the project before monitoring

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.

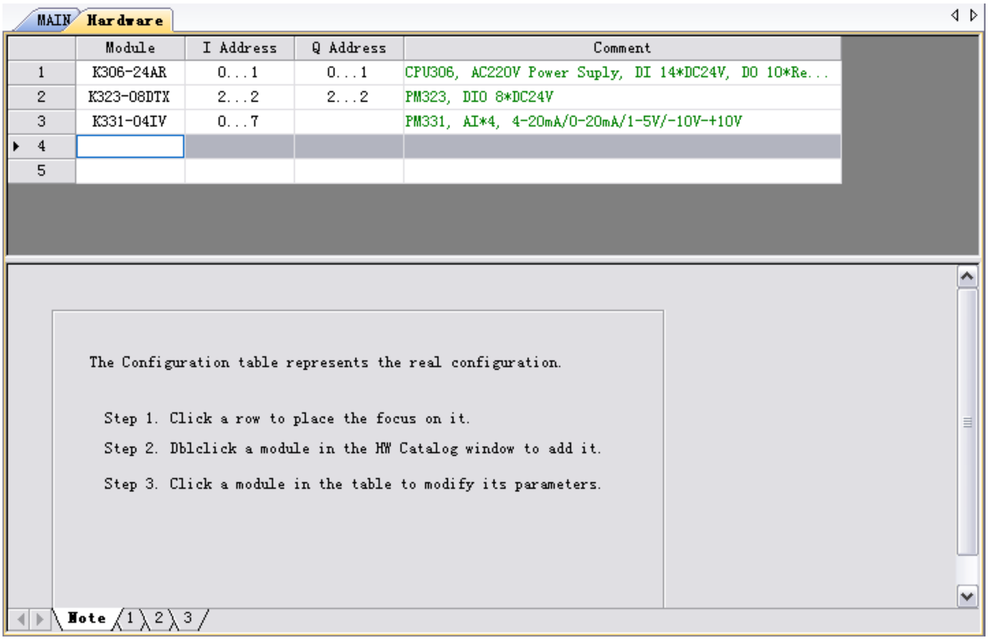
  In auto-hide mode, it shall hide automatically and shrink into an icon and stay at the border of the main KincoBuilder window; Point to this icon for a moment, the window shall appear.

➢ 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:



	Module	I Address	Q Address	Comment
1	K306-24AR	0...1	0...1	CPU306, AC220V Power Suply, DI 14*DC24V, DO 10*Re...
2	K323-08DTX	2...2	2...2	PM323, DIO 8*DC24V
3	K331-04IV	0...7		PM331, AI*4, 4-20mA/0-20mA/1-5V/-10V-+10V
▸ 4				
5				

The Configuration table represents the real configuration.

Step 1. Click a row to place the focus on it.

Step 2. Dblclick a module in the HW Catalog window to add it.

Step 3. Click a module in the table to modify its parameters.

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.

The hardware configuration parameters won't take effect unless they are downloaded into the CPU module. When the KINCO-K3 starts up, the CPU compares the preset configuration with the actual configuration of the PLC modules. If an error is detected, the CPU will go to STOP mode and the Err LED will turn on.

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 Add/Remove Modules

- **Add a module**

You can add a module using the following steps:

(1) 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.

(2) 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.

The maximum I/O channel numbers for CPU304, CPU306 and CPU308 are explicitly defined. If the number of all the channels on the added modules exceeds the limits, KincoBuilder will forbid continuing to add modules into the Cofiguration Table, 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.

4.3.3 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**].

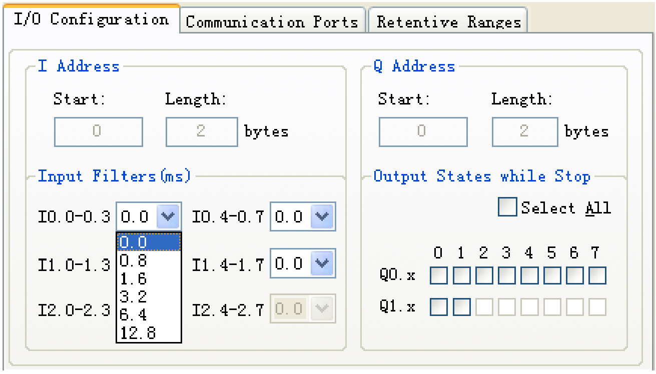
- [**Default**]: If you click this button, KincoBuilder will assign default parameters for the current module.
- [**Cancel**]: If you click this button, the original configuration of the current module will be restored.

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

4.3.3.1 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.



I Address

Start: 0 Length: 2 bytes

Input Filters (ms)

I0.0-0.3 0.0 I0.4-0.7 0.0

I1.0-1.3 I1.4-1.7 0.0

I2.0-2.3 I2.4-2.7 0.0

Q Address

Start: 0 Length: 2 bytes

Output States while Stop

□ Select All

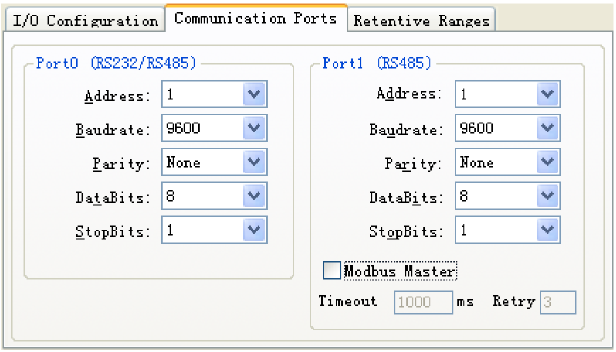
	0	1	2	3	4	5	6	7
Q0.x	□	□	□	□	□	□	□	□
Q1.x	□	□	□	□	□	□	□	□

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 a 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.



Port0 (RS232/RS485)

Address: 1
Baudrate: 9600
Parity: None
DataBits: 8
StopBits: 1

Port1 (RS485)

Address: 1
Baudrate: 9600
Parity: None
DataBits: 8
StopBits: 1

□ Modbus Master

Timeout 1000 ms Retry 3

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 is must be exclusive in the network.
- **Baudrate:** Select the desired baud rate. (2400, 4800, 9600, 19200 or 38400bps)
- **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

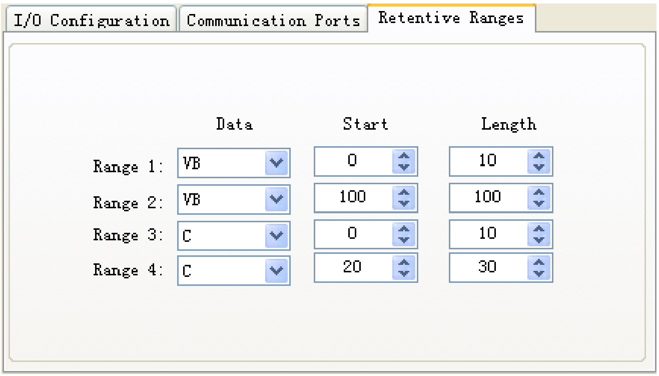
Port1 is a RS485 port. Some types of CPUs only have one serial port (Port0), and Port 1 is not provided.

- **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.

Please refer to Port1 described above for other parameters.

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.



	Data	Start	Length
Range 1:	VB	0	10
Range 2:	VB	100	100
Range 3:	C	0	10
Range 4:	C	20	30

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.

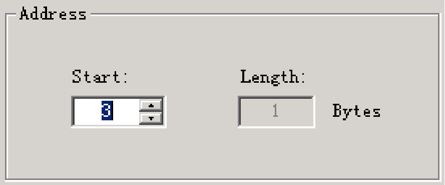
- **Range 2**
- **Range 3**
- **Range 4**

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.

4.3.3.2 Parameters of the DI Module

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



Address

Start: 3

Length: 1 Bytes

Figure 4-6 Parameters of the DI Module

➢ **Address**

• **Start**

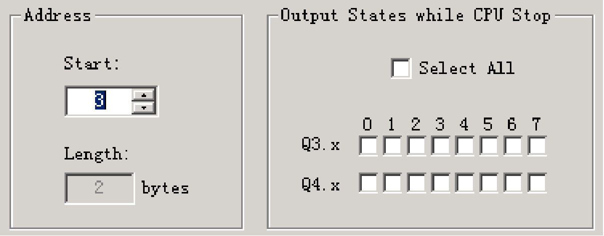
Enter the start byte address of the address range of this module in I area. The addresses for this module's channels are based on this start address.

• **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 %IB3, so the addresses of its channels are %I3.0 to %I3.7.

4.3.3.3 Parameters of the DO Module



Address

Start: 3

Length: 2 bytes

Output States while CPU Stop

□ Select All

	0	1	2	3	4	5	6	7
Q3.x	□	□	□	□	□	□	□	□
Q4.x	□	□	□	□	□	□	□	□

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**

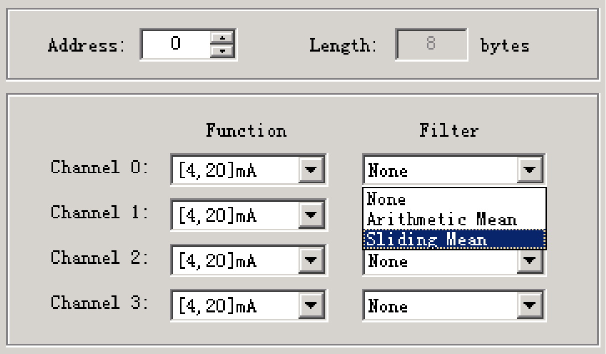
The length of this module's address range. This value is fixed, and it depends on the number of this module's DO channels.

As shown in Figure 4-7, the module has 8 DO channels, and its start address is %QB3, so the addresses of its channels are %Q3.0 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).

4.3.3.4 Parameters of the AI Module



Address: 0 Length: 8 bytes

	Function	Filter
Channel 0:	[4, 20]mA	None
Channel 1:	[4, 20]mA	
Channel 2:	[4, 20]mA	None
Channel 3:	[4, 20]mA	None

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.*

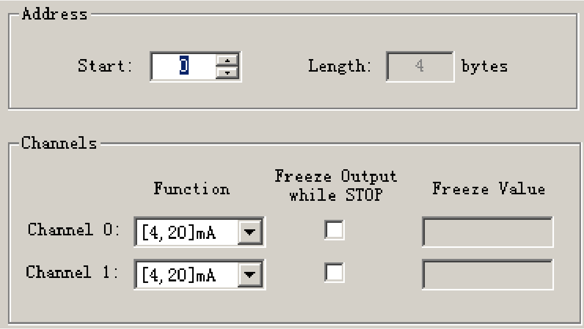
You can assign one of the following filters for a channel:

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.

4.3.3.5 Parameters of the AO Module



Address

Start: 0 Length: 4 bytes

Channels

	Function	Freeze Output while STOP	Freeze Value
Channel 0:	[4,20]mA	□	
Channel 1:	[4,20]mA	□	

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

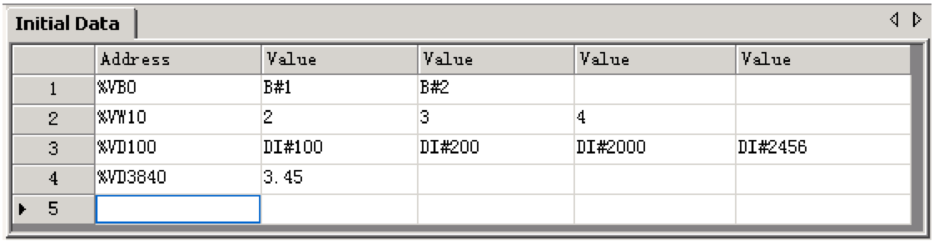
Select whether to set the analog output to a known value (**Freeze Value**) while the CPU stops. If the checkbox for an output is checked, the output shall keep at the freeze value while the CPU stops.

• 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.



	Address	Value	Value	Value	Value
1	%VB0	B#1	B#2		
2	%VW10	2	3	4	
3	%VD100	DI#100	DI#200	DI#2000	DI#2456
4	%VD3840	3.45			
▸ 5					

Figure 4-10 the Initial Data Table

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.

① Enter a direct variable, i.e. a direct address in the **Address** column.

② 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.

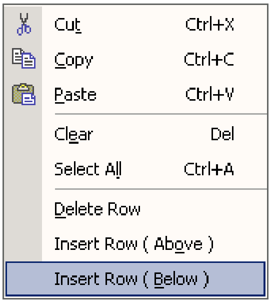
4.4.4 Editing the Initial Data Table

➢ 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.

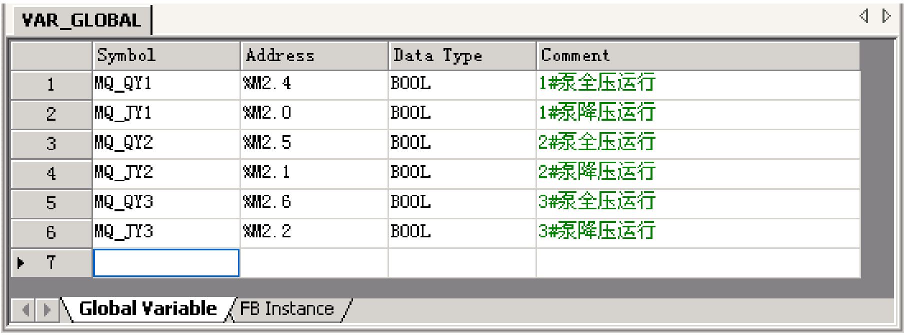
4.5 The Global Variable Table

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

- The **Global Variable** tab

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

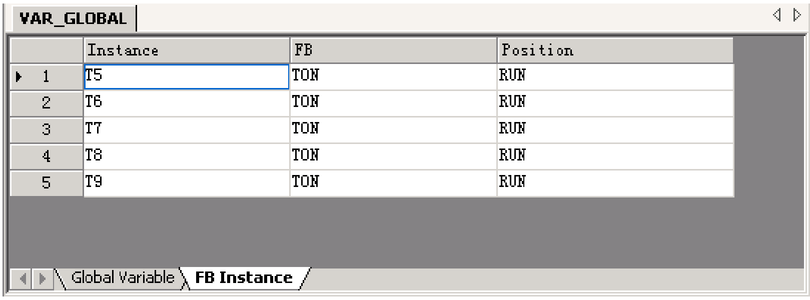
In this manual, “the Global Variable Table” usually indicates this tab.



	Symbol	Address	Data Type	Comment
1	MQ_QY1	%M2.4	BOOL	1#泵全压运行
2	MQ_JY1	%M2.0	BOOL	1#泵降压运行
3	MQ_QY2	%M2.5	BOOL	2#泵全压运行
4	MQ_JY2	%M2.1	BOOL	2#泵降压运行
5	MQ_QY3	%M2.6	BOOL	3#泵全压运行
6	MQ_JY3	%M2.2	BOOL	3#泵降压运行
▸ 7				

Figure 4-11 the Global Variable tab

- The **FB Instance** tab



	Instance	FB	Position
▸ 1	T5	TON	RUN
2	T6	TON	RUN
3	T7	TON	RUN
4	T8	TON	RUN
5	T9	TON	RUN

Figure 4-12 the FB Instance tab

As mentioned in 2.6.5 Usage 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 5 columns: **Symbol**, **Address**, **Data Type** and **Comment**.

① Open the Global Variable Table window and select the **Global Variable** tab.

② Enter the symbol name in the **Symbol** column and confirm it.

③ Enter the direct address in the **Address** column and confirm it.

④ Choose a data type from the drop list in the **Data Type** column.

⑤ (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 2.5 Variables for more information about the global variable.

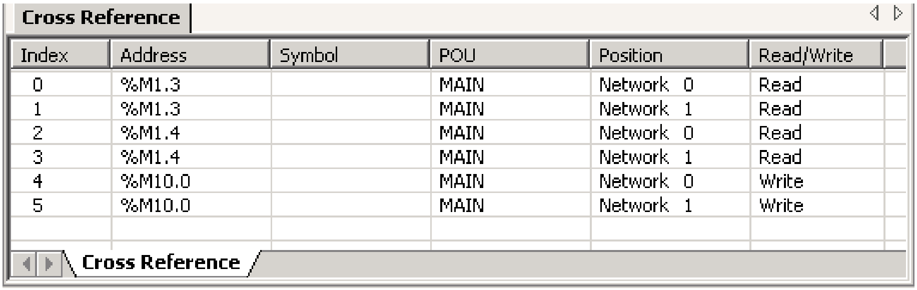
You can operate the Global Variable Table just as the Initial Data Table. Please refer to 4.4 The Initial Data Table for more information.

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	%M1.3		MAIN	Network 0	Read
1	%M1.3		MAIN	Network 1	Read
2	%M1.4		MAIN	Network 0	Read
3	%M1.4		MAIN	Network 1	Read
4	%M10.0		MAIN	Network 0	Write
5	%M10.0		MAIN	Network 1	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.

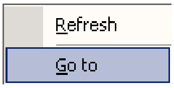
Double-click on a row in the Cross Reference Table, and you shall go to the corresponding part of your program.

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.

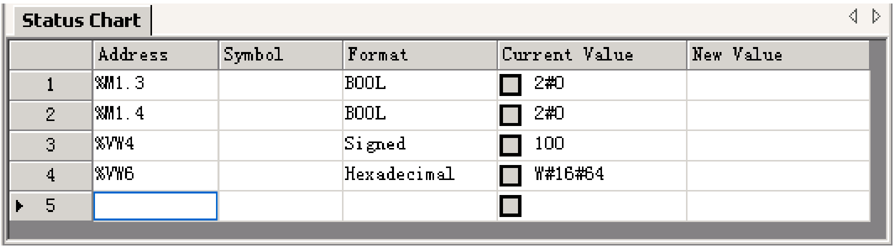


• **Refresh**: Refresh the table and display the latest cross-reference information.

• **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.



	Address	Symbol	Format	Current Value	New Value
1	%M1.3		BOOL	□ 2#0	
2	%M1.4		BOOL	□ 2#0	
3	%VW4		Signed	□ 100	
4	%VW6		Hexadecimal	□ W#16#64	
▸ 5				□	

Figure 4-14 the Status Chart

• **Address** Enter the direct 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.

4.8 Password Protection

The KINCO-K3 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.

4.8.1 Protection Privileges

The KINCO-K3 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:

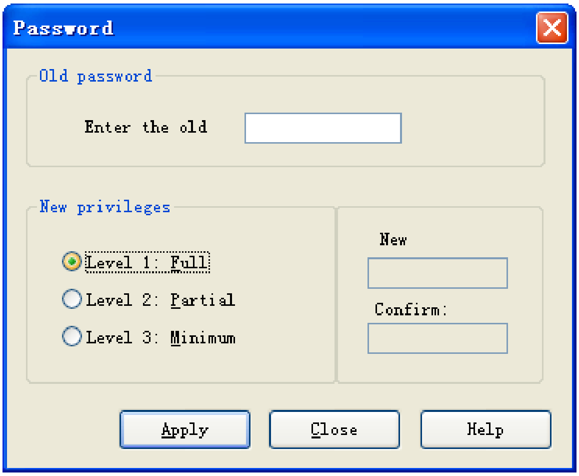


Fig. 4-15 the 'Password' Window

- **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 folloing: the station number 1, the baudrate is 9600, no parity, 8 data bits, 1 stop bit.

5 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:

```
label:
Operator    Operands                  (* Comment *)
```

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:

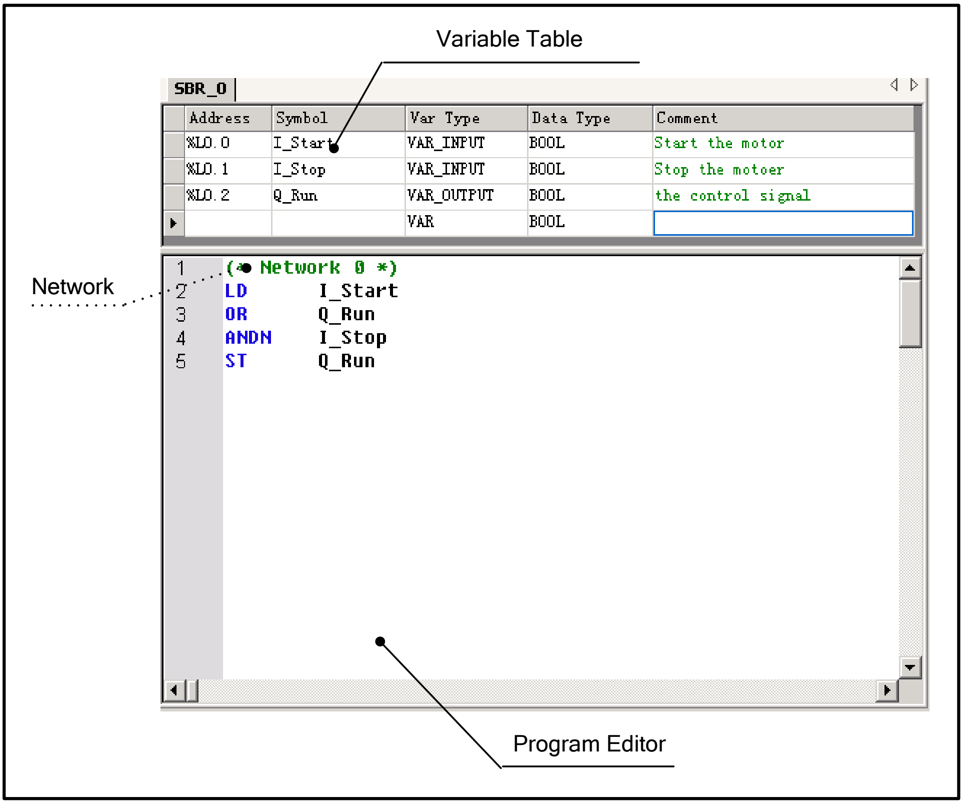
- POU type and POU name
- Variable declaration part
- Code part containing the instructions

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.



Address	Symbol	Var Type	Data Type	Comment
%L0.0	I_Start	VAR_INPUT	BOOL	Start the motor
%L0.1	I_Stop	VAR_INPUT	BOOL	Stop the motoer
%L0.2	Q_Run	VAR_OUTPUT	BOOL	the control signal
		VAR	BOOL	

```
(* Network 0 *)
LD      I_Start
OR      Q_Run
ANDN    I_Stop
ST      Q_Run
```

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 *)
```

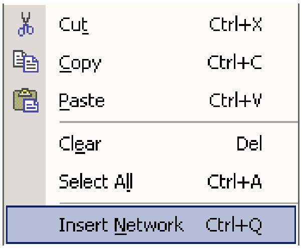
5.1.3.3 Other Operations

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:

(1) There is no error in the source IL program.

(2) 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”.

5.2 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.

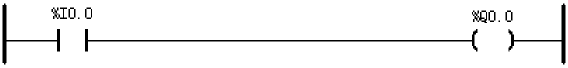


Figure 5-3 A Sample in LD

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 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 shall be considered ON all along. No state is defined for the right rail.

5.2.3 Standardized graphic symbols

(1) 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 1 or 0 respectively. The term *link state* shall be synonymous with the term *power flow*.

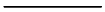
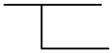
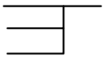
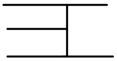
Symbol	Name	Description
	Horizontal link	A horizontal link element shall be indicated by a horizontal line. It transmits the state of the element on its immediate left to the element on its immediate right.
	Vertical link (With attached horizontal links)	The vertical link element shall consist of a vertical line intersecting with one or more horizontal link elements on each side. 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: - 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. The state of the vertical link shall be copied to all of the attached horizontal links on its right.
		
		

Table 5-2 Link elements

(2) 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 of the state of the horizontal link at its left side with an appropriate function of an associated Boolean variable. A contact does not modify the value of the associated Boolean variable.

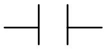
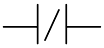
Symbol	Name	Description
*** 	Normally open contact	The state of the left link is copied to the right link if the state of the associated Boolean variable (indicated by "***") is ON. Otherwise, the state of the right link is OFF.
*** 	Normally closed contact	The state of the left link is copied to the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

(3) 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

(4) 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 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 shall be transferred to the designated network label unconditionally.
├── (1) ──>> Label	Conditional Jump	Program execution shall be transferred to the designated network label 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.

(5) 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 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.

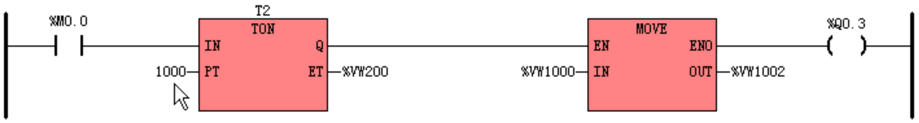
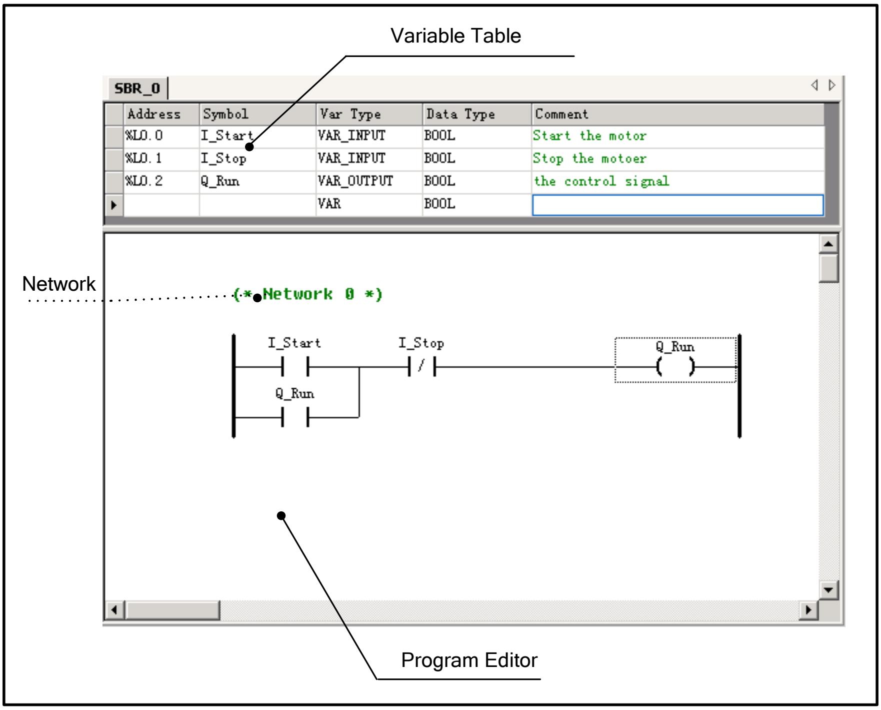


Figure 5-4 Functions and Function Blocks

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.



Address	Symbol	Var Type	Data Type	Comment
%L0.0	I_Start	VAR_INPUT	BOOL	Start the motor
%L0.1	I_Stop	VAR_INPUT	BOOL	Stop the motoer
%L0.2	Q_Run	VAR_OUTPUT	BOOL	the control signal
		VAR	BOOL	

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 an 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-lick 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.

(1) 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.

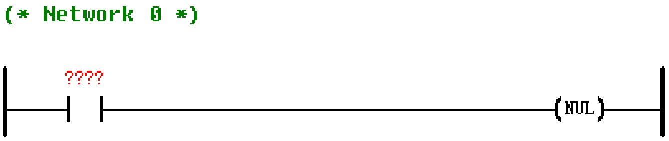
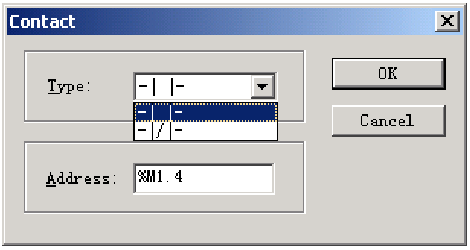


Figure 5-6 A New Network

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.

(2) 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.

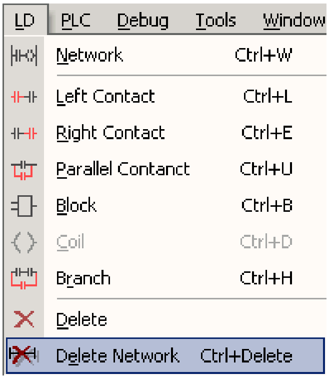


Address: %M1.4

Figure 5-7 A Contact Property Dialog Box

(3) 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:



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.

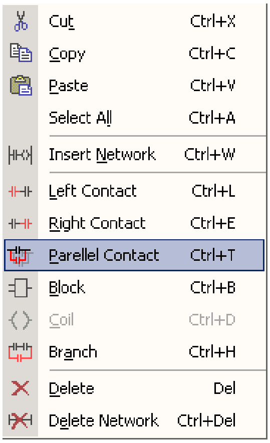
Branch: Draw a branch parallel to other elements.

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.



➢ 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:

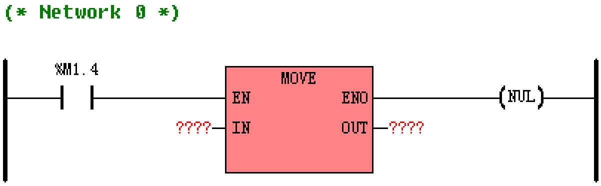
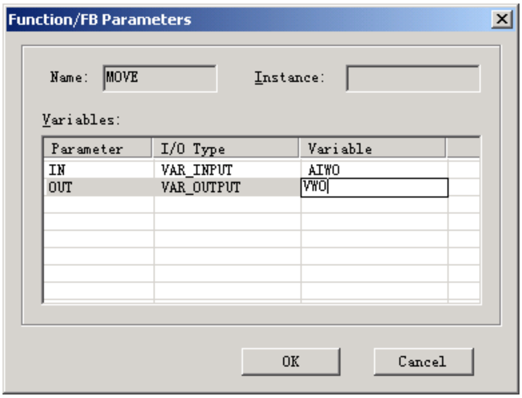


Figure 5-8 Adding other Elements

(4) 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.



Function/FB Parameters

Name: MOVE Instance:

Variables:

Parameter	I/O Type	Variable
IN	VAR_INPUT	AIW0
OUT	VAR_OUTPUT	VW0

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:

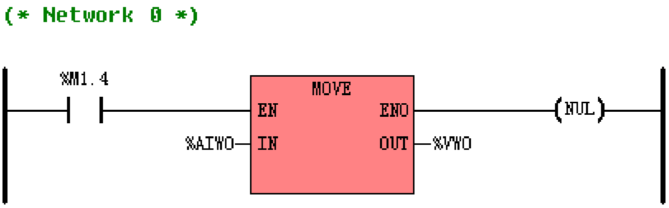


Figure 5-10 The Modified Network

(5) 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.

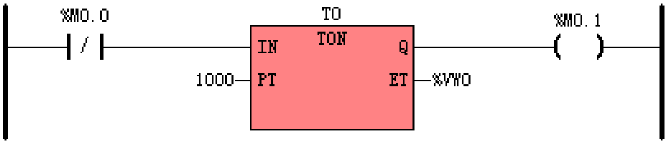
5.2.3.5 Online Monitoring

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

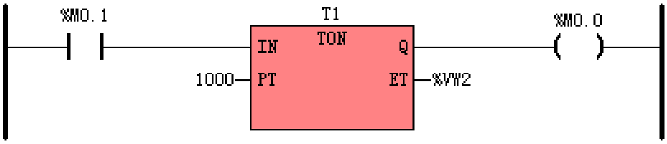
In this mode, all the PLC data status is displayed in the LD Editor window, and you are not allowed to edit the program.

5.2.3.6 Example

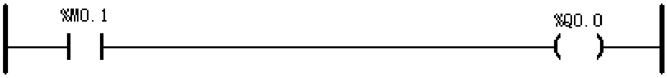
```
(* Network 0 *)
(* Start T0 with the output of T1, timing: 1000*1ms *)
```



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



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



6 Chapter VI KINCO-K3 Instruction Set

KINCO-K3 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	*bit* —| |—		☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	Normally closed contact	*bit* —|/|—		
IL	LD	LD *bit*	C	
	LDN	LDN *bit*		
	AND	AND *bit*	P	
	OR	OR *bit*		
	ANDN	ANDN *bit*		
	ORN	ORN *bit*		

Operand	Input/Output	Data Type	Acceptable Memory Areas
bit	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.

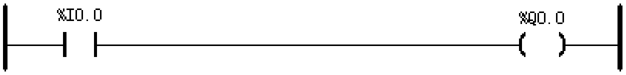
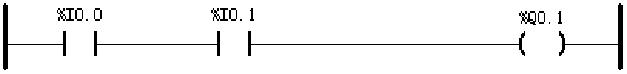
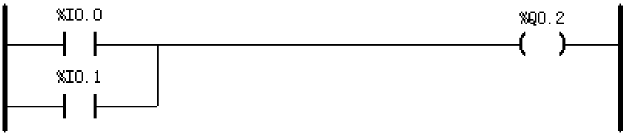
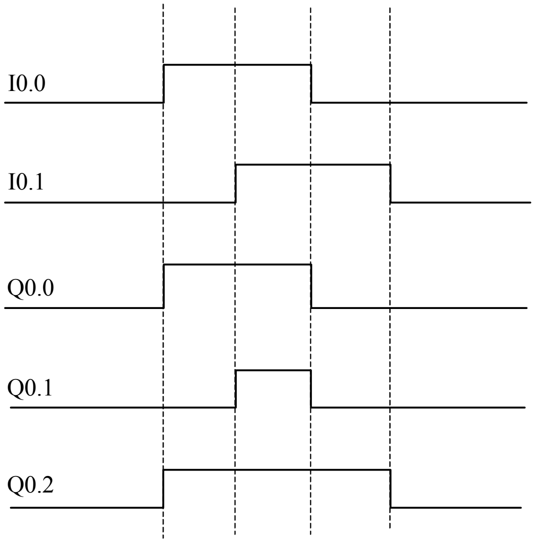
The Normally Closed contacts are represented by the *LDN*, *ANDN*, and *ORN* instructions.

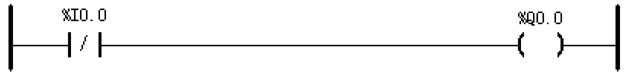
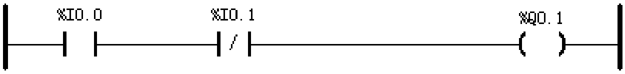
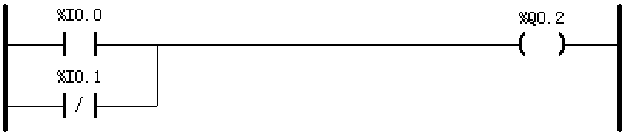
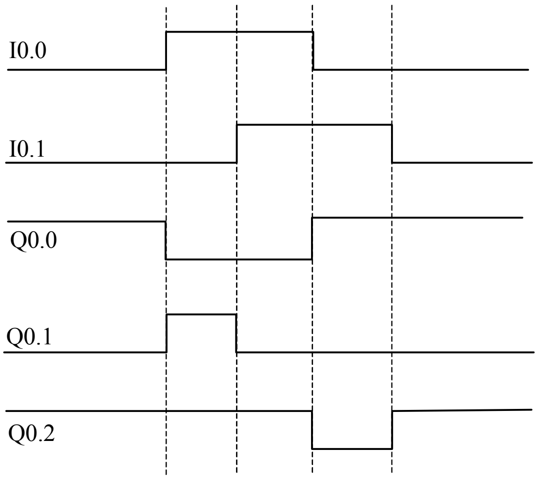
The *LDN* instruction loads the logical NOT of the *bit* value and sets the CR equal to the operation result.

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 %I0.0 ST %Q0.0
	LD %I0.0 AND %I0.1 ST %Q0.1
	LD %I0.0 OR %I0.1 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	*bit* —| I |—		☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
	Normally closed immediate contact	*bit* —|/I|—		
IL	LDI	LDI *bit*	C	
	LDNI	LDNI *bit*		
	ANDI	ANDI *bit*	P	
	ORI	ORI *bit*		
	ANDNI	ANDNI *bit*		
	ORNI	ORNI *bit*		

Operand	Input/Output	Data Type	Acceptable Memory Areas
bit	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 contanct, 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	Coil	*bit* —( )—		☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	Negated Coil	*bit* —(/)—		
	Set Coil	*bit* —(S)—		
	Reset Coil	*bit* —(R)—		
	Null coil	—(NUL)—		
IL	ST	ST *bit*	U	
	STN	STN *bit*		
	R	R *bit*		
	S	S *bit*		

Operand	Input/Output	Data Type	Acceptable Memory Areas
bit	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*.

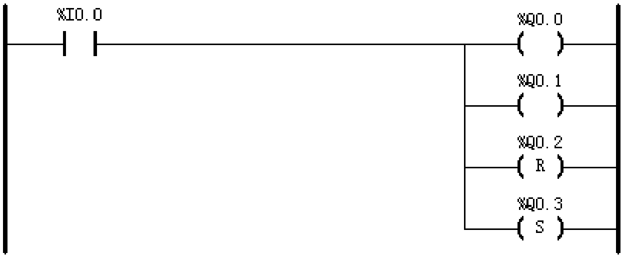
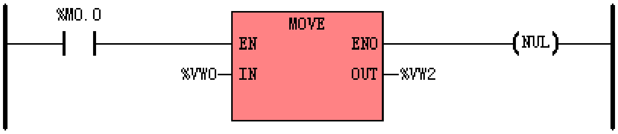
The *STN* instruction writes the inverse of the CR to the output image register for the *bit*.

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.

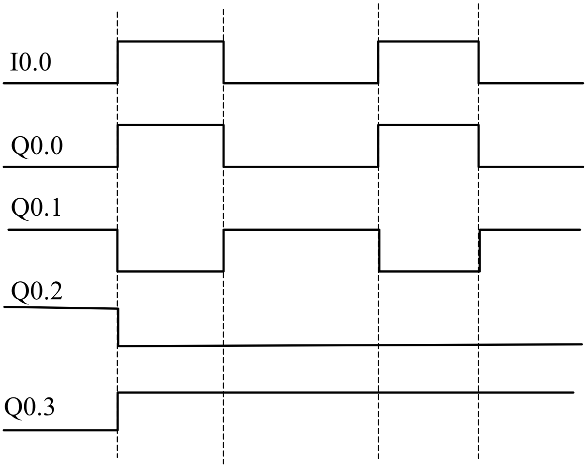
ST, *STN, R* and *S* instructions don’t influence the CR.

- Examples

LD	IL
	LD %I0.0 ST %Q0.0 STN %Q0.1 R %Q0.2 S %Q0.3
 %M0.0 MOVE EN ENO %VW0 IN OUT %VW2 NUL	LD %M0.0 MOVE %VW0, %VW2

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	Immediate Coil	*bit* —(I)—		□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
	Set Immediate Coil	*bit* —(SI)—		
	Reset Immediate Coil	*bit* —(RI)—		
IL	STI	STI *bit*	U	
	RI	RI *bit*		
	SI	SI *bit*		

Operand	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 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.

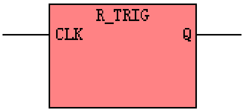
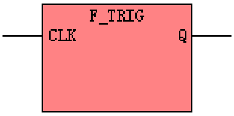
When the *RI* instruction is executed, if the CR is equal to 1, both the physical output (*bit*) and the corresponding output image register are set equal to 0 immediately, otherwise they remain unchanged.

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 Edge detection

➢ Description

	Name	Usage	Group	
LD	Rising edge detector			☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	Falling edge detector			
IL	R_TRIG	R_TRIG	P	
	F_TRIG	F_TRIG		

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

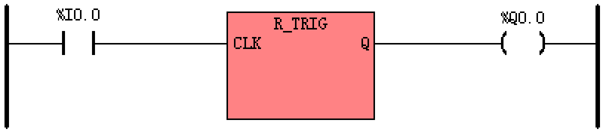
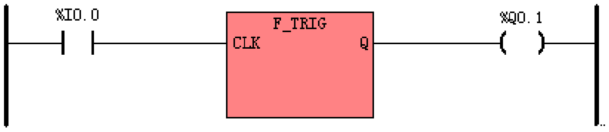
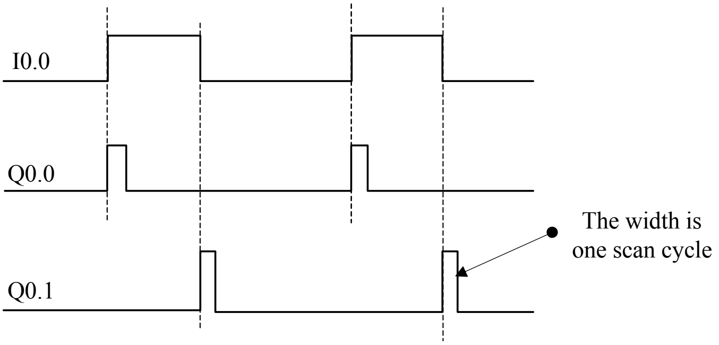
The function of the *F_TRIG* instruction is to detect the falling edge of the *CLK* input: following a 1-to-0 transition of the *CLK* input, the *Q* output is set to 1 for one scan cycle and then returns to 0.

- **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.

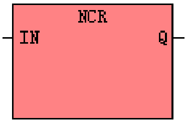
- Examples

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>
</td><td>LD %I0.0
R_TRIG
ST %Q0.0</td></tr>
<tr><td>
</td><td>LD %I0.0
F_TRIG
ST %Q0.1</td></tr>
<tr><th colspan="2">Time Sequence Diagram</th></tr>
<tr><td colspan="2">

The width is one scan cycle</td></tr>
</table>

6.2.6 NCR (NOT)

➢ Description

	Name	Usage	Group	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	NCR			
IL	NCR	NCR	P	

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

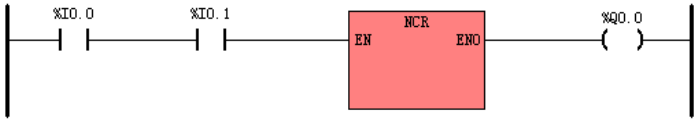
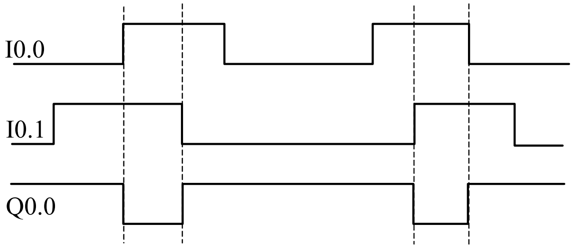
- **LD**

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
	LD %I0.0 AND %I0.1 NCR ST %Q0.0
Time Sequence Diagram	
	

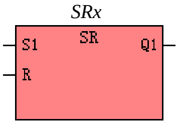
Bistable elements

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.

6.2.7.1 SR (Set Dominant Bistable)

➢ Description

	Name	**Usage**	**Group**	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	SR			
IL	SR	LD *S1* SR *SRx, R*	P	

Parameter	**Input/Output**	**Data Type**	**Acceptable Memory Areas**
SRx	-	SR instance	SR
S1	Input	BOOL	Power flow
R	Input	BOOL	I, Q, V, M, SM, L, T, C, RS, SR
Q1	Output	BOOL	Power flow

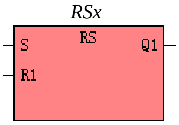
The Set Dominant Bistable (*SR*) is a bistable element where the set input dominates. If the set (*S1*) and reset (*R*) inputs are both 1, both the output *Q1* and the status value of *SRx* will be 1.

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

S1	***R***	***Q1, SRx***
0	0	Previous value
0	1	0
1	0	1
1	1	1

6.2.7.2 RS (Reset Dominant Bistable)

➢ Description

	Name	**Usage**	**Group**	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	RS			
IL	RS	LD *S* RS *RSx, R1*	P	

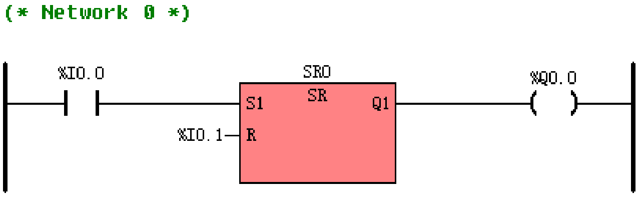
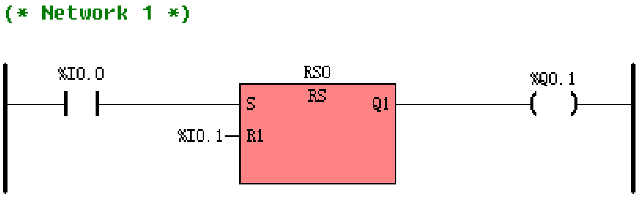
Parameter	**Input/Output**	**Data Type**	**Acceptable Memory Areas**
RSx	-	RS instance	RS
S	Input	BOOL	Power flow
R1	Input	BOOL	I, Q, V, M, SM, L, T, C, RS, SR
Q1	Output	BOOL	Power flow

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.

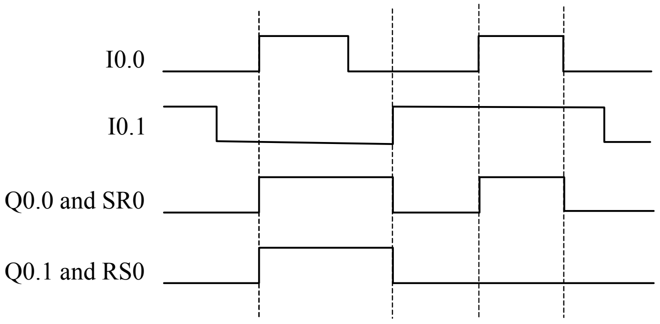
The following is a Truth Table for the *RS* Instruction:

R1	***S***	***Q1, SRx***
0	0	Previous value
0	1	1
1	0	0
1	1	0

6.2.7.3 Examples

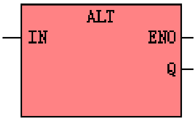
LD	**IL**
 (* Network 0 *) %I0.0 SR0 SR S1 Q1 %I0.1 R %Q0.0	(* Network 0 *) LD %I0.0 SR SR0, %I0.1 ST %Q0.0
 (* Network 1 *) %I0.0 RS0 RS S Q1 %I0.1 R1 %Q0.1	(* Network 1 *) LD %I0.0 RS RS0, %I0.1 ST %Q0.1

Time Sequence Diagram



6.2.8 ALT (Alternate)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	ALT			
IL	ALT	ALT *Q*	U	

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

- **LD**

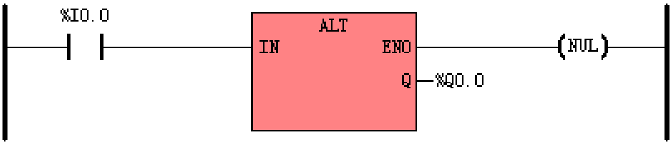
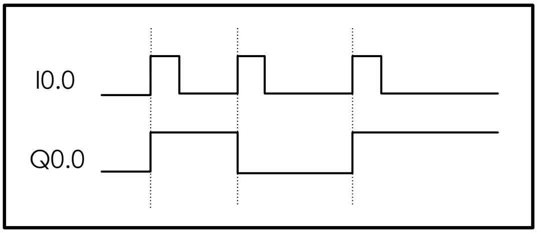
The ALT instruction changes the value of *Q* from 1 to 0 or from 0 to 1 on the rising edge of the *IN* input.

- **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.

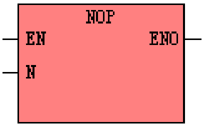
This instruction does not influence the CR.

- Examples

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>
</td><td>LD %I0.0
ALT %Q0.0</td></tr>
<tr><th colspan="2">Time Sequence Diagram</th></tr>
<tr><td colspan="2">
</td></tr>
</table>

6.2.9 NOP (No Operation)

➢ Description

	Name	Usage	Group	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	NOP			
IL	NOP	NOP *N*	U	

Parameter	Input/Output	Data Type	Acceptable Memory Areas
N	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.

6.2.10 Bracket Modifier

- Description

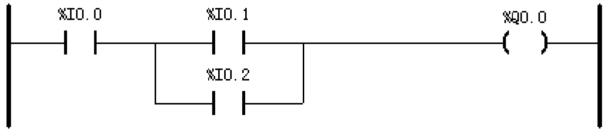
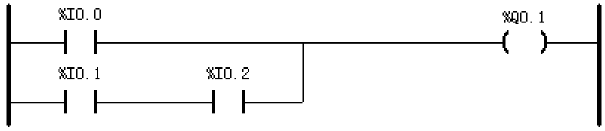
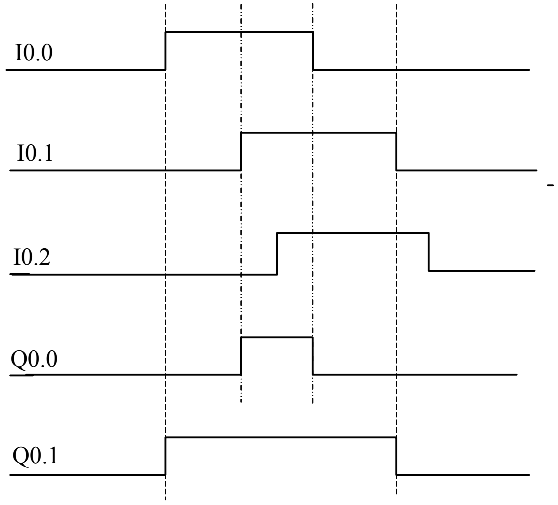
	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
IL	AND(	AND(	U	
	OR(	OR(		
	)	)	P	

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.

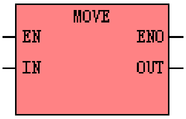
- Examples

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> </td><td>LD %I0.0
AND(
LD %I0.1
OR %I0.2
)
ST %Q0.0</td></tr>
<tr><td> </td><td>LD %I0.0
OR(
LD %I0.1
AND %I0.2
)
ST %Q0.1</td></tr>
<tr><th colspan="2">Timing Diagram</th></tr>
<tr><td colspan="2"> </td></tr>
</table>

6.3 Move Instructions

6.3.1 MOVE

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	MOVE			
IL	MOVE	MOVE *IN*, *OUT*	U	

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

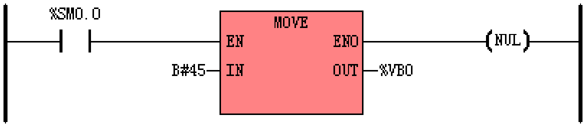
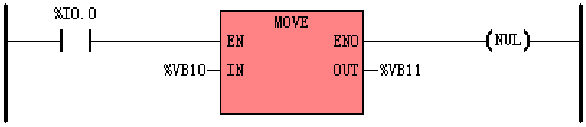
- **LD**

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

- **IL**

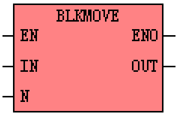
If the CR is 1, this instruction is executed, and it doesn't influence the CR.

➢ Examples

LD	 %SM0.0 MOVE EN ENO B#45 IN OUT %VB0 NUL	%SM0.0 is always ON, therefore the MOVE is always executed: B#45 is assigned to %VB0.
	 %I0.0 MOVE EN ENO %VB10 IN OUT %VB11 NUL	If %I0.0 is 0, the MOVE is not executed. If %I0.0 is 1, the value of %VB10 is assigned to %VB11.
IL	LD %SM0.0 (* The CR is created with %SM0.0 *) MOVE B#45, %VB0 (* B#45 is assigned to %VB0 *)	
	LD %I0.0 (* The CR is created with %I0.0 *) MOVE %VB10, %VB11 (* If the CR is 1, the value of %VB10 is assigned to %VB11. *) (*Otherwise, this statement is not executed, %VB11 remains unchanged *)	

6.3.2 BLKMOVE (Block Move)

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	BLKMOVE			
IL	BLKMOVE	BLKMOVE *IN*, *OUT*, *N*	U	

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

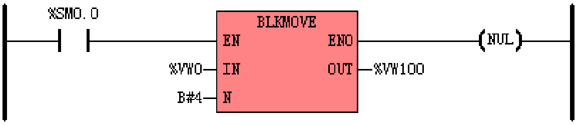
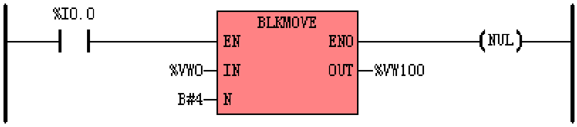
- **LD**

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

- **IL**

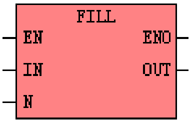
If the CR is 1, this instruction is executed, and it does not influence the CR.

➢ Examples

<table>
<tr><td rowspan="2"><b>LD</b></td><td>
</td><td>%SM0.0 is always ON, therefore the BLKMOVE is always executed: the data in %VW0 - %VW6 are moved into %VW100 - %VW106.</td></tr>
<tr><td>
</td><td>If %I0.0 is 1, the data in %VW0 - %VW6 are moved into %VW100 - %VW106. Otherwise, the BLKMOVE is not executed.</td></tr>
<tr><td rowspan="2"><b>IL</b></td><td colspan="2">LD %SM0.0 (* The CR is created with the% SM0.0 *)
BLKMOVE %VW0, %VW100, B#4 (* The data in VW0 - VW6 are moved into %VW100 - %VW106 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* The CR is created with %I0.0 *)
BLKMOVE %VW0, %VW100, B#4 (* If the CR is 0, this statement isn't executed *)
(* If the CR is 1, the data in %VW0 - %VW6 are moved into %VW100 - %VW106 *)</td></tr>
<tr><td><b>RESULT</b></td><td colspan="2">The following is an example:
<table><tr><td>VW0</td><td>VW2</td><td>VW4</td><td>VW6</td></tr><tr><td>0</td><td>10</td><td>20</td><td>30</td></tr><tr><td>VW100</td><td>VW102</td><td>VW104</td><td>VW106</td></tr><tr><td>0</td><td>10</td><td>20</td><td>30</td></tr></table></td></tr>
</table>

6.3.3 FILL (Memory Fill)

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	FILL			
IL	FILL	FILL *IN*, *OUT*, *N*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	BYTE	Constant
N	Input	BYTE	constant
OUT	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*.

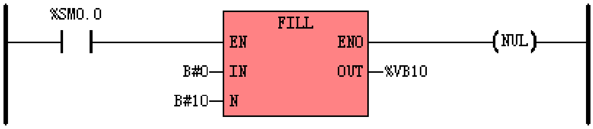
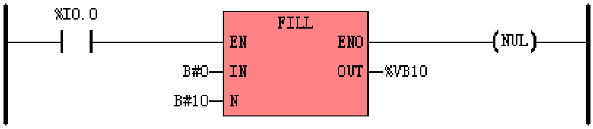
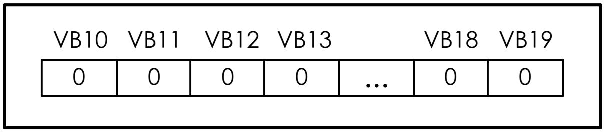
- **LD**

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

- **IL**

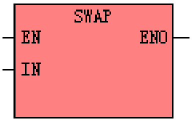
If the CR is 1, this instruction is executed, and it does not influence the CR.

- Examples

LD	 %SM0.0 FILL EN ENO B#0 IN OUT %VB10 B#10 N NUL	%SM0.0 is always ON, therefore the FILL is always executed: 10 variables from %VB10 to %VB19 are all set to B#0.
	 %I0.0 FILL EN ENO B#0 IN OUT %VB10 B#10 N NUL	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.
IL	LD %SM0.0 (* The CR is created with %SM0.0 *) FILL B#0, %VB10, B#10 (* 10 variables from %VB10 to %VB19 are all set to B#0 *)	
	LD %I0.0 (* The CR is created with %I0.0 *) FILL B#0, %VB10, B#10 (* If the CR is 1, 10 variables from %VB10 to %VB19 are all set to B#0 *) (* If the CR is 0, this statement is not executed *)	
RESULT	VB10 VB11 VB12 VB13 VB18 VB19 0 0 0 0 … 0 0	

6.3.4 SWAP

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	SWAP			
IL	SWAP	SWAP *IN*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	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*).

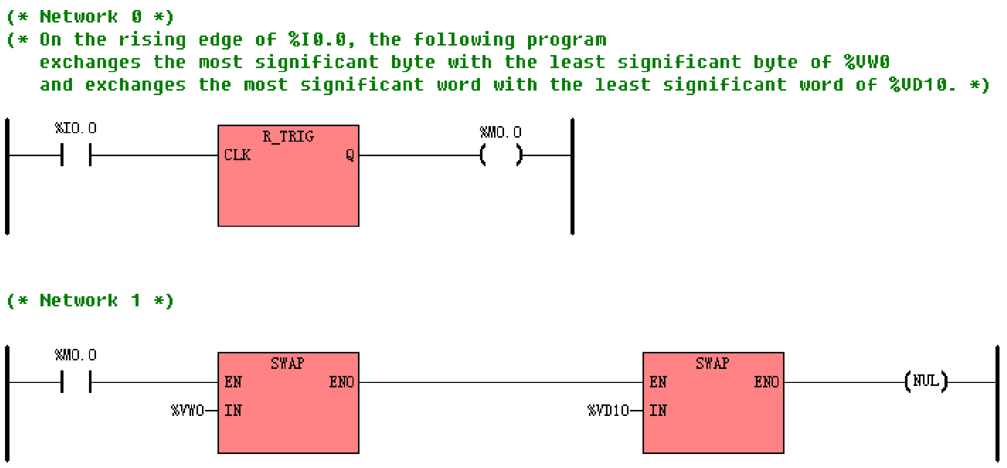
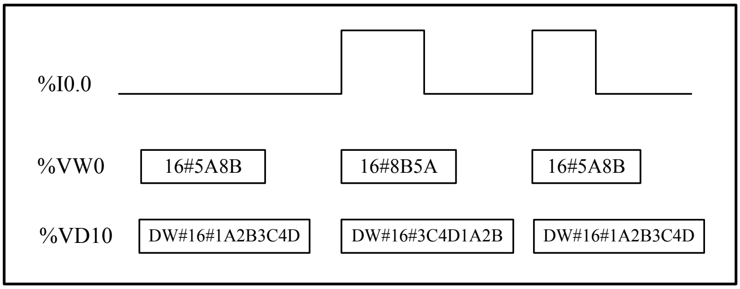
- **LD**

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

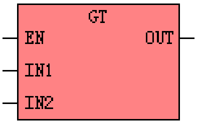
LD	 (* Network 0 *) (* On the rising edge of %I0.0, the following program exchanges the most significant byte with the least significant byte of %VW0 and exchanges the most significant word with the least significant word of %VD10. *) %I0.0 R_TRIG CLK Q %M0.0 (* Network 1 *) %M0.0 SWAP EN ENO %VW0 IN SWAP EN ENO %VD10 IN NUL
IL	(* Network 0 *) LD %I0.0 R_TRIG (* On the rising edge of %I0.0, *) SWAP %VW0 (* the most significant byte with the least significant byte of %VW0 are exchanged, *) SWAP %VD10 (* and the most significant word with the least significant word of %VD10 are exchanged. *)
结 果	Assume that the initial value of %VW0 is W#16#5A8B and the initial value of %Vd10 is DW#16#1A2B3C4D.  %I0.0 %VW0 16#5A8B 16#8B5A 16#5A8B %VD10 DW#16#1A2B3C4D DW#16#3C4D1A2B DW#16#1A2B3C4D

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	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	GT			
IL	GT	GT *IN1*, *IN2*	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN1	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
IN2	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant, pointer
OUT (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*;

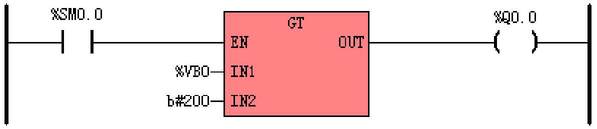
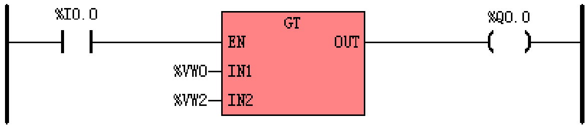
If *EN* is 0, this instruction is not executed, and *OUT* is set equal to 0.

- **IL**

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

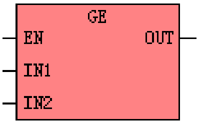
If CR is 0, this instruction is not executed, and CR remains 0.

➢ Examples

LD	 %SM0.0 GT EN OUT %VB0 IN1 b#200 IN2 %Q0.0	SM0.0 is always ON, therefore *GT* 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.
	 %I0.0 GT EN OUT %VW0 IN1 %VW2 IN2 %Q0.0	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: *GT* is not executed, and Q0.0 is set to be 0.
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	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	GE			
IL	GE	GE *IN1*, *IN2*	P	

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

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

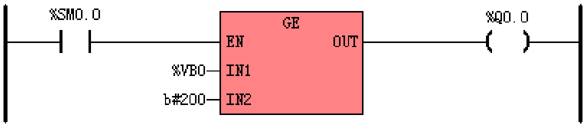
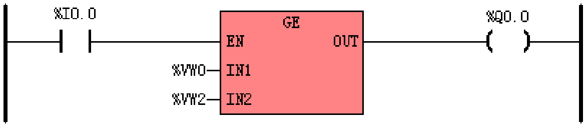
- **LD**

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**

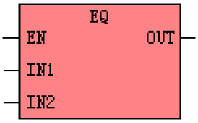
If CR is 1, this instruction compares *IN1* greater than or equal to *IN2* and the Boolean result is assigned to CR; If CR is 0, this instruction is not executed, and CR remains 0.

➢ Examples

LD	 %SM0.0 GE EN OUT %VB0 IN1 b#200 IN2 %Q0.0	SM0.0 is always ON, therefore *GE* 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.
	 %I0.0 GE EN OUT %VW0 IN1 %VW2 IN2 %Q0.0	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: *GE* is not executed, and Q0.0 is set to be 0.
IL	LD %SM0.0 (* CR is created with SM0.0 *) 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 *) ST %Q0.0 (* Q0.0 is set equal to CR *)	
	LD %I0.0 (* CR is created with I0.0 *) GE %VW0, %VW2 (* If CR is 1: if VW0 is greater than or equal to VW2, CR is set to be 1, *) (* otherwise CR is set to be 0; If CR is 0: GE is not executed, CR remains 0 *) ST %Q0.0 (* Q0.0 is set equal to CR *)	

6.4.3 EQ (Equal to)

- Description

	Name	Usage	Group	
LD	EQ			☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
IL	EQ	EQ *IN1*, *IN2*	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN1	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant
IN2	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant
OUT (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* equal to *IN2* and the Boolean result is assigned to *OUT*;

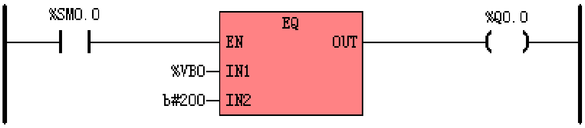
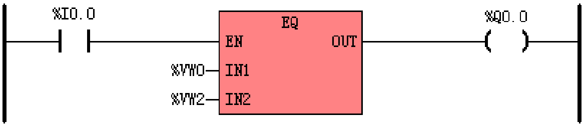
If *EN* is 0, this instruction is not executed, and *OUT* is set equal to 0.

- **IL**

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

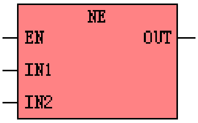
If CR is 0, this instruction is not executed, and CR remains 0.

- Examples

LD	 %SM0.0 EQ EN OUT %VB0 IN1 b#200 IN2 %Q0.0	SM0.0 is always ON, therefore *EQ* 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.
	 %I0.0 EQ EN OUT %VW0 IN1 %VW2 IN2 %Q0.0	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: *EQ* is not executed, and Q0.0 is set to be 0.
IL	LD %SM0.0 (* CR is created with SM0.0 *) EQ %VB0, B#200 (* If VB0 is 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 *) 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 *) (* If CR is 0: EQ is not executed, CR remains 0 *) ST %Q0.0 (* Q0.0 is set equal to CR *)	

6.4.4 NE (Not Equal to)

- Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	NE			
IL	NE	NE *IN1*, *IN2*	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN1	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant
IN2	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant
OUT (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*;

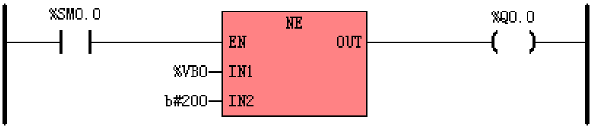
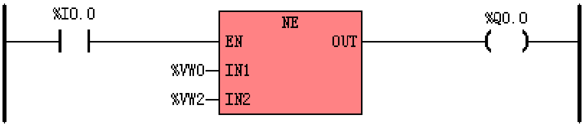
If *EN* is 0, this instruction is not executed, and *OUT* is set equal to 0.

- **IL**

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

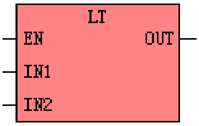
If CR is 0, this instruction is not executed, and CR remains 0.

- Examples

LD	 %SM0.0 NE EN OUT %VB0 IN1 b#200 IN2 %Q0.0	SM0.0 is always ON, therefore *NE* 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.
	 %I0.0 NE EN OUT %VW0 IN1 %VW2 IN2 %Q0.0	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: *NE* is not executed, and Q0.0 is set to be 0.
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	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	LT			
IL	LT	LT *IN1*, *IN2*	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN1	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant
IN2	Input	BYTE, INT, DINT, REAL	I, Q, M, V, L, SM, AI, AQ, T, C, HC, constant
OUT (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*;

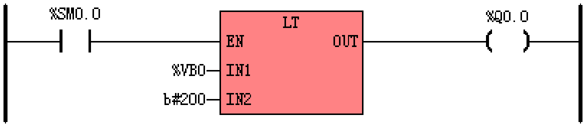
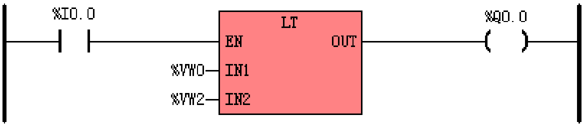
If *EN* is 0, this instruction is not executed, and *OUT* is set equal to 0.

- **IL**

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

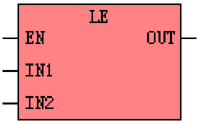
If CR is 0, this instruction is not executed, and CR remains 0.

- Examples

LD	 %SM0.0 LT EN OUT %VB0 IN1 b#200 IN2 %Q0.0	SM0.0 is always ON, therefore *LT* 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.
	 %I0.0 LT EN OUT %VW0 IN1 %VW2 IN2 %Q0.0	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: *LT* is not executed, and Q0.0 is set to be 0.
IL	LD %SM0.0 (* CR is created with SM0.0 *) LT %VB0, B#200 (* If VB0 is less 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 *) 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 *) (* If CR is 0: LT is not executed, CR remains 0 *) ST %Q0.0 (* Q0.0 is set equal to CR *)	

6.4.6 LE (Less than or Equal to)

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	LE			
IL	LE	LE *IN1*, *IN2*	P	

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

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

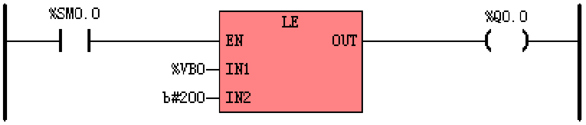
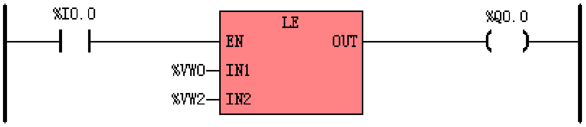
- **LD**

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.

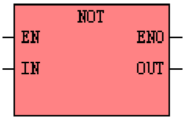
- Examples

LD	 %SM0.0 LE EN OUT %VB0 IN1 b#200 IN2 %Q0.0	SM0.0 is always ON, therefore *LE* 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.
	 %I0.0 LE EN OUT %VW0 IN1 %VW2 IN2 %Q0.0	If I0.0 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. If I0.0 is 0: *LE* is not executed, and Q0.0 is set to be 0.
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 %I0.0 (* CR is created with I0.0 *) 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	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	NOT			
IL	NOT	NOT *OUT*	U	

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

- **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*.

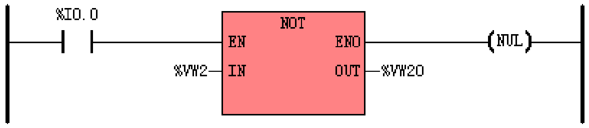
If *EN* is 0, this instruction is not executed.

- IL

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

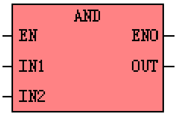
If CR is 0, this instruction is not executed.

- Examples

LD	 %I0.0 NOT EN ENO %VW2 IN OUT %VW20 NUL	If I0.0 is 0: *NOT* is not executed. If I0.0 is 1: *NOT* inverts each bit of VW2 and assigns the result to VW20.
IL	LD %I0.0 (* CR is created with I0.0 *) NOT %VW20 (* If CR is 1: *NOT* inverts each bit of VW20 and still stores the result in VW20 *) (* If CR is 0: *NOT* instruction is not executed *)	
Result	For the LD example, if *NOT* instruction is executed, the result will be as the following: Address VW2 Value W#16#5555 Address VW20 Value W#16#AAAA	

6.5.2 AND

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	AND			
IL	AND	AND *IN*, *OUT*	U	

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

- **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.

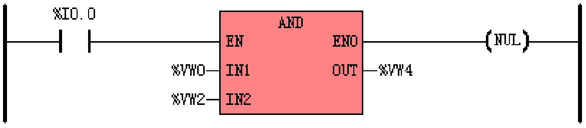
- **IL**

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.

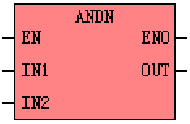
If CR is 0, this instruction is not executed.

➢ Examples

LD	 %I0.0 AND EN ENO %VW0 IN1 OUT %VW4 %VW2 IN2 NUL	If I0.0 is 0: *AND* is not executed. If I0.0 is 1: The *AND* instruction ANDs the corresponding bits of VW0 and VW2, and assigns the result to VW4.
IL	LD %I0.0 (* CR is created with I0.0 *) AND %VW0, %VW2 (* If CR is 1: The *AND* instruction ANDs the corresponding bits of VW0 and VW2, *) (* and still stores the result in VW2 *) (* If CR is 0: The *AND* instruction is not executed *)	
Result	For the LD example, if *AND* instruction is executed, the result will be as the following: Address VW0 VW2 Value W#16#129B W#16#960F Address VW4 Value W#16#120B	

6.5.3 ANDN

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	ANDN			
IL	ANDN	ANDN *IN*, *OUT*	U	

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

- **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.

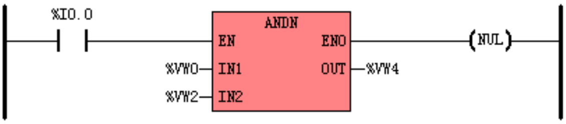
- **IL**

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.

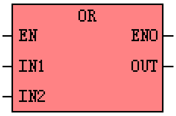
If CR is 0, this instruction is not executed.

➢ Examples

LD	 %I0.0 ANDN EN ENO %VW0 IN1 OUT %VW4 %VW2 IN2 (NUL)	If I0.0 is 0: *ANDN* is not executed. If I0.0 is 1: The *ANDN* instruction ANDs the corresponding bits of VW0 and VW2, then inverts each bit of the result, and assigns the final result to VW4.
IL	LD %I0.0 (* CR is created with I0.0 *) ANDN %VW0, %VW2 (* If CR is 1: The *ANDN* instruction ANDs the corresponding bits of VW0 and VW2, *) (* then inverts each bit of the result, and still stores the final result in VW2 *) (* If CR is 0: The *ANDN* instruction is not executed *)	
Result	For the LD example, if *ANDN* instruction is executed, the result will be as the following: Address VW0 VW2 Value W#16#129B W#16#960F Address VW4 Value W#16#EDF4	

6.5.4 OR

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	OR			
IL	OR	OR *IN*, *OUT*	U	

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

- **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.

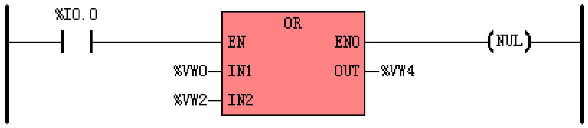
- **IL**

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.

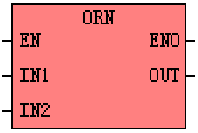
If CR is 0, this instruction is not executed.

- Examples

LD	 %I0.0 OR EN ENO %VW0 IN1 OUT %VW4 %VW2 IN2 NUL	If I0.0 is 0: *OR* is not executed. If I0.0 is 1: The *OR* instruction ORs the corresponding bits of VW0 and VW2, and assigns the result to VW4.
IL	LD %I0.0 (* CR is created with I0.0 *) OR %VW0, %VW2 (* If CR is 1: The *OR* instruction ORs the corresponding bits of VW0 and VW2, *) (* and still stores the result in VW2 *) (* If CR is 0: The *OR* instruction is not executed *)	
Result	For the LD example, if *OR* instruction is executed, the result will be as the following: 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			☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
IL	ORN	ORN *IN*, *OUT*	U	

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

- **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.

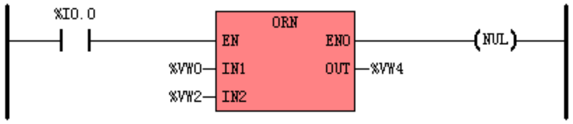
- **IL**

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.

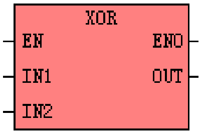
If CR is 0, this instruction is not executed.

➢ Examples

LD	 %I0.0 ORN EN ENO %VW0 IN1 OUT %VW4 %VW2 IN2 NUL	If I0.0 is 0: *ORN* is not executed. If I0.0 is 1: The *ORN* instruction ORs the corresponding bits of VW0 and VW2, then inverts each bit of the result, and assigns the final result to VW4.
IL	LD %I0.0 (* CR is created with I0.0 *) ORN %VW0, %VW2 (* If CR is 1: The *ORN* instruction ORs the corresponding bits of VW0 and VW2, *) (* then inverts each bit of the result, and still stores the final result in VW2 *) (* If CR is 0: The *ORN* instruction is not executed *)	
Result	For the LD example, if *ORN* instruction is executed, the result will be as the following: 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	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	XOR			
IL	XOR	XOR *IN*, *OUT*	U	

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

- **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.

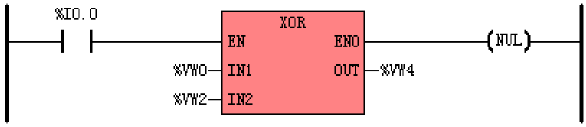
- **IL**

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

<table>
<tr><td><b>LD</b></td><td>

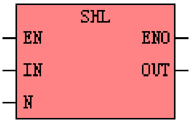
%I0.0
XOR
EN
ENO
%VW0
IN1
OUT
%VW4
%VW2
IN2
NUL

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

6.6 Shift/Rotate Instructions

6.6.1 SHL (Shift left)

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	SHL			
IL	SHL	SHL *OUT*, *N*	U	

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

- **LD**

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

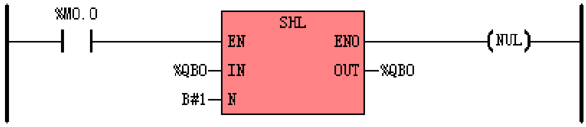
If *EN* is 1, this instruction shifts the value of *IN* to the left by *N* bits, and each bit is filled with a zero while it is shifted left. The result is assigned to *OUT*. If *EN* is 0, this instruction is not executed.

- **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.

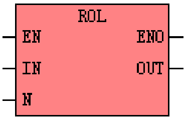
If CR is 0, this instruction is not executed.

➢ Examples

LD	 %M0.0 SHL EN ENO %QB0 IN OUT %QB0 B#1 N NUL	If M0.0 is 0: *SHL* isn't executed. If M0.0 is 1: *SHL* shifts QB0 to the left by 1 bit, and the result is still assigned to QB0.
IL	LD %M0.0 (* CR is created with M0.0 *) SHL %QB0, B#1 (* If CR is 1: *SHL* shifts QB0 to the left by 1 bit, and the result is still stored in QB0 *) (* If CR is 0: *SHL* instruction is not executed *)	
Result	For the LD example, if *SHL* instruction is executed, the result will be as the following: QB0 B#2#10000001 After 1st shift: QB0 B#2#00000010 After 2nd shift: B#2#00000100 After 3rd shift: B#2#00001000 After 4th shift: B#2#00010000	

6.6.2 ROL (Rotate left)

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	ROL			
IL	ROL	ROL *OUT*, *N*	U	

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

- **LD**

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

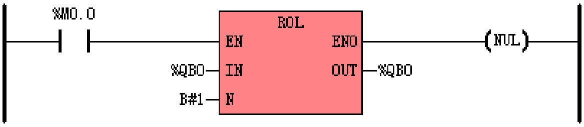
If *EN* is 1, this instruction rotates the value of *IN* to the left by *N* bits, and the MSB is rotated to the LSB. The result is assigned to *OUT*. If *EN* is 0, this instruction is not executed.

- **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.

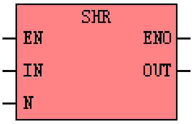
If CR is 0, this instruction is not executed.

- Examples

LD	 %M0.0 ROL EN ENO %QB0 IN OUT %QB0 B#1 N NUL	If M0.0 is 0: *ROL* isn't executed. If M0.0 is 1: *ROL* rotates QB0 to the left by 1 bit, and the MSB is rotated to the LSB. The result is still assigned to QB0.
IL	LD %M0.0 (* CR is created with M0.0 *) ROL %QB0, B#1 (* If CR is 1: *ROL* rotates QB0 to the left by 1 bit, and the result is still stored in QB0 *) (* If CR is 0: *ROL* instruction is not executed *)	
Result	For the LD example, if *ROL* instruction is executed, the result will be as the following: QB0 B#2#10100001 After 1st shift: QB0 B#2#01000011 After 2nd shift: B#2#10000110 After 3rd shift: B#2#00001101 After 4th shift: B#2#00011010	

6.6.3 SHR (Shift right)

- Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	SHR			
IL	SHR	SHR *OUT*, *N*	U	

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

- **LD**

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

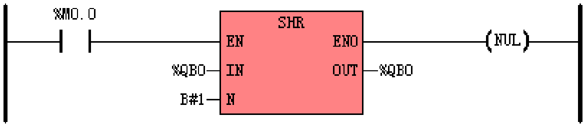
If *EN* is 1, this instruction shifts the value of *IN* to the right by *N* bits, and each bit is filled with a zero while it is shifted right. The result is assigned to *OUT*. If *EN* is 0, this instruction is not executed.

- **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.

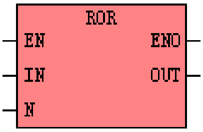
If CR is 0, this instruction is not executed.

➢ Examples

LD	 %M0.0 SHR EN ENO %QB0 IN OUT %QB0 B#1 N (NUL)	If M0.0 is 0: *SHR* isn't executed. If M0.0 is 1: *SHR* shifts QB0 to the right by 1 bit, and the result is still assigned to QB0.
IL	LD %M0.0 (* CR is created with M0.0 *) SHR %QB0, B#1 (* If CR is 1: *SHR* shifts QB0 to the right by 1 bit, and the result is still stored in QB0 *) (* If CR is 0: *SHR* instruction is not executed *)	
Result	For the LD example, if *SHR* instruction is executed, the result will be as the following: QB0 B#2#10000001 After 1st shift: QB0 B#2#01000000 After 2nd shift: B#2#00100000 After 3rd shift: B#2#00010000 After 4th shift: B#2#00001000	

6.6.4 ROR (Rotate right)

- Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	ROR			
IL	ROR	ROR *OUT*, *N*	U	

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

- **LD**

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

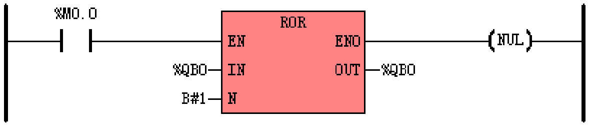
If *EN* is 1, this instruction rotates the value of *IN* to the right by *N* bits, and the LSB is rotated to the MSB. The result is assigned to *OUT*. If *EN* is 0, this instruction is not executed.

- **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.

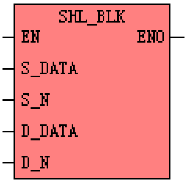
If CR is 0, this instruction is not executed.

- Examples

LD	 %M0.0 ROR EN ENO %QB0 IN OUT %QB0 B#1 N NUL	If M0.0 is 0: *ROR* isn't executed. If M0.0 is 1: *ROR* rotates QB0 to the right by 1 bit, and the LSB is rotated to the MSB. The result is still assigned to QB0.
IL	LD %M0.0 (* CR is created with M0.0 *) ROL %QB0, B#1 (* If CR is 1: *ROR* rotates QB0 to the right by 1 bit, and the result is still stored in QB0 *) (* If CR is 0: *ROR* instruction is not executed *)	
Result	For the LD example, if *ROR* instruction is executed, the result will be as the following: QB0 B#2#10100001 After 1st shift: QB0 B#2#11010000 After 2nd shift: B#2#01101000 After 3rd shift: B#2#00110100 After 4th shift: B#2#00011010	

6.6.5 SHL_BLK (Bit String Shift Left)

➢ Description

	Name	Usage	Group	
LD	SHL_BLK			□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
IL	SHL_BLK	SHL_BLK *S_DATA, S_N, D_DATA, D_N*	U	

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

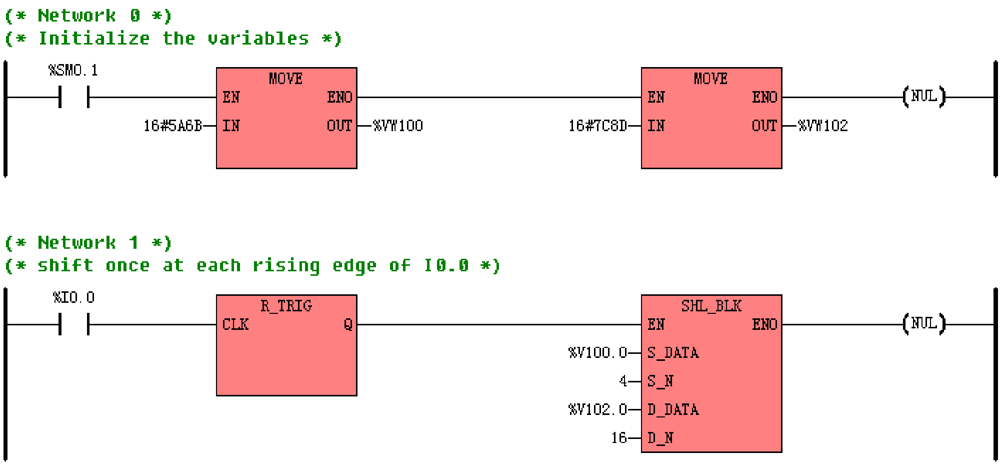
If *EN* is 1, this instruction is executed.

- **IL**

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

➢ Examples

LD



IL

```
(* 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
```

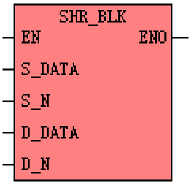
Result

The result is shown as the following:

	VW102				VW100			
	V103.7			V102.0	V101.7			V100.0
Initial value	0111	1100	1000	1101	0101	1010	0110	1011
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			□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
IL	SHR_BLK	SHR_BLK *S_DATA, S_N, D_DATA, D_N*	U	

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

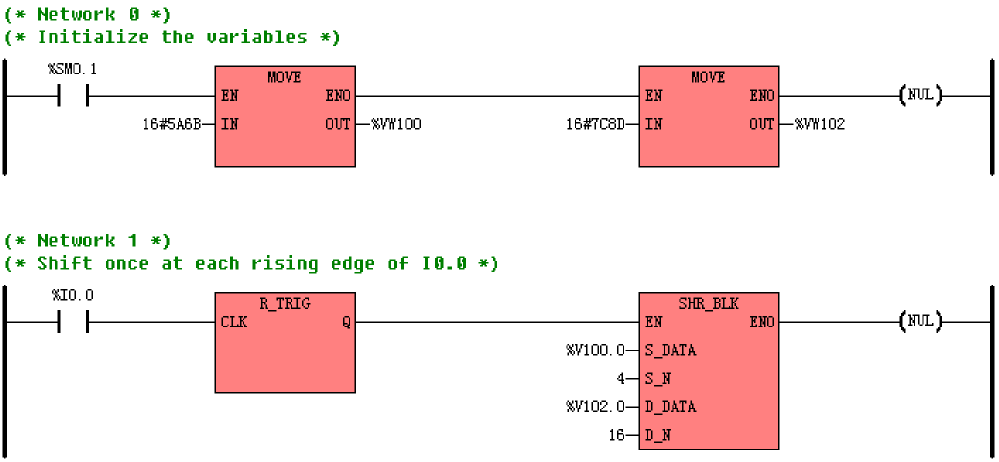
If *EN* is 1, this instruction is executed.

- **IL**

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

➢ Examples

LD



(* Network 0 *)
(* Initialize the variables *)

(* Network 1 *)
(* Shift once at each rising edge of I0.0 *)

IL

```
(* 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
SHR_BLK     %V100.0, 4, %V102.0, 16
```

Result

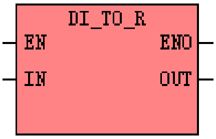
The result is shown as the following:

	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	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	DI_TO_R			
IL	DI_TO_R	DI_TO_R *IN*, *OUT*	U	

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

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

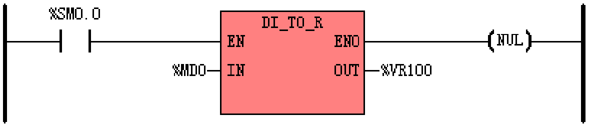
- **LD**

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

- **IL**

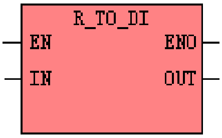
If CR is 1, this instruction is not executed, and it does not influence CR.

- Examples

<table>
<tr><td><b>LD</b></td><td>
</td><td>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.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0 (* CR is created SM0.0 *)
DI_TO_R %MD0, %VR100 (* The value of MD0 is converted to a REAL value that is assigned to VR100 *)</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is shown as the following:
<table><tr><th>MD0</th><th>VR100</th></tr><tr><td>DI#123</td><td>123.0</td></tr><tr><td>DI#-9876</td><td>-9876.0</td></tr></table></td></tr>
</table>

6.7.2 R_TO_DI (REAL To DINT)

- Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	R_TO_DI			
IL	R_TO_DI	R_TO_DI *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	REAL	V, L, constant
OUT	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.

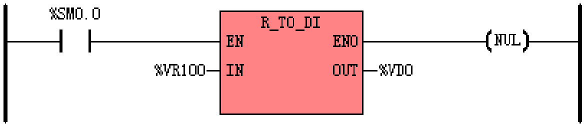
- **LD**

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

- **IL**

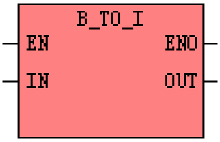
If CR is 1, this instruction is executed, and it does not influence CR.

➢ Examples

<table>
<tr><td><b>LD</b></td><td>
</td><td>SM0.0 is always ON, therefore <i>R_TO_DI</i> is always executed: The value of VD4000 is converted to a DINT value that is assigned to VD0.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0 (* CR is created SM0.0 *)
R_TO_DI %VD4000, %VD0 (* The value of VD4000 is converted to a DINT value that is assigned to VD0 *)</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is shown as the following:
<table>
<tr><th>VR100</th><th>VD0</th></tr>
<tr><td>123.4</td><td>DI#123</td></tr>
<tr><td>5213.6</td><td>DI#5214</td></tr>
</table></td></tr>
</table>

6.7.3 B_TO_I (BYTE To INT)

- Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	B_TO_I			
IL	B_TO_I	B_TO_I *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	BYTE	I, Q, M, V, L, SM, Constant
OUT	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**

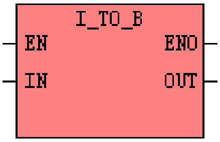
If *EN* is 1, this instruction is executed.

- **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	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	I_TO_B			
IL	I_TO_B	I_TO_B *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	INT	I, Q, M, V, L, SM, AI, AQ, T, C, Constant
OUT	Output	BYTE	Q, M, V, L, SM

This instruction assigns the least byte of the input *IN* to the *OUT*.

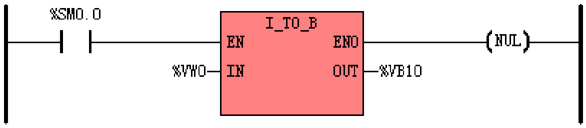
- **LD**

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

- **IL**

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

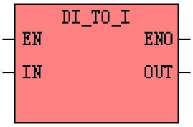
- Examples

<table>
<tr><td><b>LD</b></td><td>
%SM0.0
I_TO_B
EN
ENO
%VW0
IN
OUT
%VB10
NUL

</td><td>SM0.0 is always 1, so I_TO_B is always be executed: assigns VB0 (the least byte of VW0) to VB10.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0
I_TO_B %VW0, %VB10</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is shown as the following:
<table>
<tr><th>VW0</th><th>VB10</th></tr>
<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>
</table></td></tr>
</table>

6.7.5 DI_TO_I (DINT To INT)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	DI_TO_I			
IL	DI_TO_I	DI_TO_I *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	DINT	I, Q, M, V, L, SM, HC, Constant
OUT	Output	INT	Q, M, V, L, SM, AQ

This instruction assigns the least word of the input *IN* to the *OUT*.

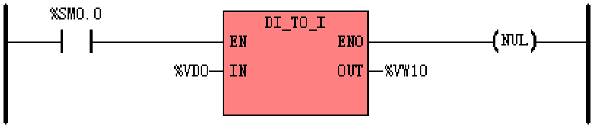
- **LD**

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

- **IL**

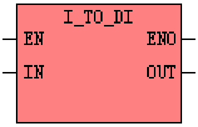
If CR is 1, this instruction is executed, and it does not influence CR.

- Examples

<table>
<tr><td><b>LD</b></td><td> </td><td>SM0.0 is always 1, so DI_TO_I is always be executed: assigns VW0 (the least word of VD0) to VW10.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0
DI_TO_I %VD0, %VW10</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is shown as the following:
<table>
<tr><th>VD0</th><th>VW10</th></tr>
<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>
</table></td></tr>
</table>

6.7.6 I_TO_DI (INT To DINT)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	I_TO_DI			
IL	I_TO_DI	I_TO_DI *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	INT	I, Q, M, V, L, SM, AI, AQ, T, C, Constant
OUT	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**

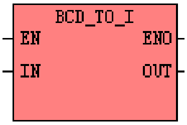
If *EN* is 1, this instruction is executed.

- **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	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	BCD_TO_I			
IL	BCD_TO_I	BCD_TO_I *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	WORD	I, Q, M, V, L, SM, Constant
OUT	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.

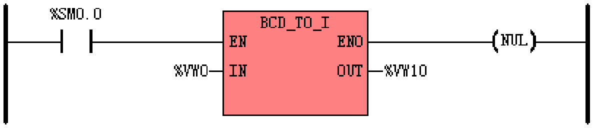
- **LD**

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

- **IL**

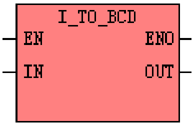
If CR is 1, this instruction is executed, and it does not influence CR.

- Examples

<table>
<tr><td><b>LD</b></td><td>
</td><td>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.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0
BCD_TO_I %VW0, %VW10</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is shown as the following:
<table>
<tr><th>VW0</th><th>VW10</th></tr>
<tr><td>16#99</td><td>99</td></tr>
<tr><td>16#4567</td><td>4567</td></tr>
<tr><td>16#9999</td><td>9999</td></tr>
</table></td></tr>
</table>

6.7.8 I_TO_BCD (INT To BCD)

➢ Description

	Name	Usage	Group	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	I_TO_BCD			
IL	I_TO_BCD	I_TO_BCD *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	INT	I, Q, M, V, L, SM, AI, AQ, T, C, Constant
OUT	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.

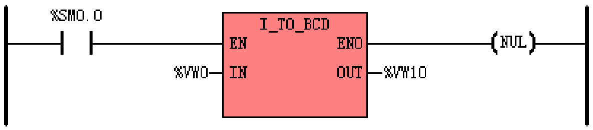
- **LD**

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

- **IL**

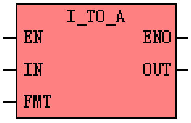
If CR is 1, this instruction is executed, and it does not influence CR.

- Examples

<table>
<tr><td><b>LD</b></td><td>
</td><td>SM0.0 is always 1, so I_TO_ BCD is always be executed: converts VW0 to a BCD value and assigns it to VW10.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0
I_TO_ BCD %VW0, %VW10</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is shown as the following:
<table>
<tr><th>VW0</th><th>VW10</th></tr>
<tr><td>99</td><td>16#99</td></tr>
<tr><td>4567</td><td>16# 4567</td></tr>
<tr><td>9999</td><td>16# 9999</td></tr>
</table></td></tr>
</table>

6.7.9 I_TO_A (INT To ASCII)

➢ Description

	Name	Usage	Group	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	I_TO_A			
IL	I_TO_A	I_TO_A *IN, OUT, FMT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	INT	I, Q, M, V, L, SM, AI, AQ, T, C, Constant
FMT	Input	BYTE	I, Q, M, V, L, SM
OUT	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) *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.

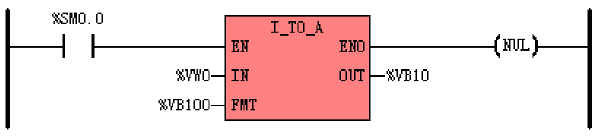
- **LD**

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

- **IL**

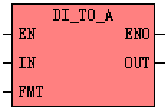
If CR is 1, this instruction is executed, and it does not influence CR.

➢ Examples

<table>
<tr><td><b>LD</b></td><td>
</td><td>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.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0
I_TO_A %VW0, %VB10, %VB100</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is as the following:
<table>
<tr><th>VB100</th><th>VW0</th><th colspan="8">Result</th></tr>
<tr><td></td><td></td><td>VB10</td><td></td><td></td><td></td><td></td><td></td><td></td><td>VB17</td></tr>
<tr><td>B#3</td><td>12</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>‘0’</td><td>‘.’</td><td>‘0’</td><td>‘1’</td><td>‘2’</td></tr>
<tr><td></td><td>-23456</td><td>32</td><td>45</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>‘2’</td><td>‘3’</td><td>‘.’</td><td>‘4’</td><td>‘5’</td><td>‘6’</td></tr>
</table></td></tr>
</table>

6.7.10 DI_TO_A (DINT To ASCII)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	DI_TO_A			
IL	DI_TO_A	DI_TO_A *IN, OUT, FMT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	DINT	I, Q, M, V, L, SM, HC, Constants
FMT	Input	BYTE	I, Q, M, V, L, SM
OUT	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.

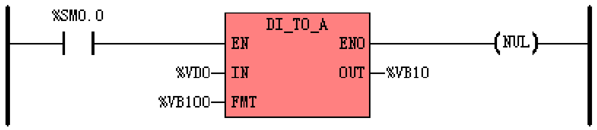
- **LD**

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	 %SM0.0 DI_TO_A EN ENO %VD0 IN OUT %VB10 %VB100 FMT (NUL)	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.
IL	LD %SM0.0 DI_TO_A %VD0, %VB10, %VB100	

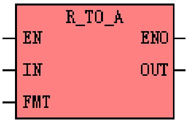
Result

The result is as the following:

VB 100	**VD 0**	**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
		‘ ’	‘ ’	‘ ’	‘ ’	‘-’	‘1’	‘2’	‘3’	‘.’	‘4’	‘5’	‘6’

6.7.11 R_TO_A (REAL To ASCII)

➢ Description

	Name	**Usage**	**Group**	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	R_TO_A			
IL	R_TO_A	R_TO_A *IN*, *OUT*, *FMT*	U	

Operands	**Input/Output**	**Data Type**	**Acceptable Memory Areas**
IN	Input	REAL	V, L, Constants
FMT	Input	BYTE	I, Q, M, V, L, SM
OUT	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).

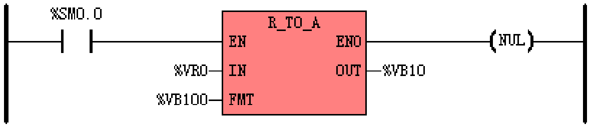
- **LD**

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

- **IL**

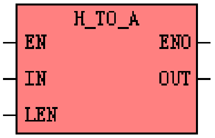
If CR is 1, this instruction is executed, and it does not influence CR.

- Examples

<table>
<tr><td><b>LD</b></td><td>
</td><td>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.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0
R_TO_A %VR0, %VB10, %VB100</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is as the following:
<table>
<tr><th>VB100</th><th>VR0</th><th colspan="8">Result</th></tr>
<tr><td></td><td></td><td>VB10</td><td></td><td></td><td></td><td></td><td></td><td></td><td>VB17</td></tr>
<tr><td>B#16#83</td><td>123.4</td><td>32</td><td>49</td><td>50</td><td>51</td><td>46</td><td>52</td><td>48</td><td>48</td></tr>
<tr><td></td><td></td><td>‘ ’</td><td>‘1’</td><td>‘2’</td><td>‘3’</td><td>‘.’</td><td>‘4’</td><td>‘0’</td><td>‘0’</td></tr>
<tr><td></td><td>-123.4567</td><td>45</td><td>49</td><td>50</td><td>51</td><td>46</td><td>52</td><td>53</td><td>55</td></tr>
<tr><td></td><td></td><td>‘-’</td><td>‘1’</td><td>‘2’</td><td>‘3’</td><td>‘.’</td><td>‘4’</td><td>‘5’</td><td>‘7’</td></tr>
</table></td></tr>
</table>

6.7.12 H_TO_A (Hexadecimal To ASCII)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	H_TO_A			
IL	H_TO_A	R_TO_A *IN, OUT, LEN*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	BYTE	I, Q, M, V, L, SM
LEN	Input	BYTE	I, Q, M, V, L, SM, Constants
OUT	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.

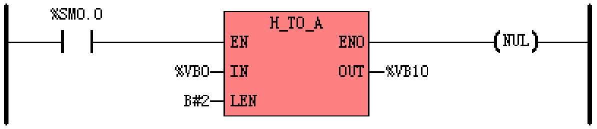
- **LD**

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

- **IL**

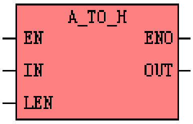
If CR is 1, this instruction is executed, and it does not influence CR.

➢ Examples

<table>
<tr>
<td><b>LD</b></td>
<td>
</td>
<td>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.</td>
</tr>
<tr>
<td><b>IL</b></td>
<td colspan="2">LD %SM0.0
H_TO_A %VB0, %VB10, B#2</td>
</tr>
<tr>
<td><b>Result</b></td>
<td colspan="2">The result is as the following:
<table>
<tr><th>VB0</th><th>VB1</th><th colspan="4">Result</th></tr>
<tr><td></td><td></td><td>VB10</td><td></td><td></td><td>VB13</td></tr>
<tr><td>B#16#1A</td><td>B#16#2B</td><td>49</td><td>65</td><td>50</td><td>66</td></tr>
<tr><td></td><td></td><td>‘1’</td><td>‘A’</td><td>‘2’</td><td>‘B’</td></tr>
<tr><td>B#16#7C</td><td>B#16#8D</td><td>55</td><td>67</td><td>56</td><td>68</td></tr>
<tr><td></td><td></td><td>‘7’</td><td>‘C’</td><td>‘8’</td><td>‘D’</td></tr>
</table>
</td>
</tr>
</table>

6.7.13 A_TO_H (ASCII to Hexadecimal)

➢ Description

	Name	Usage	Group	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	A_TO_H			
IL	A_TO_H	A_TO_H *IN*, *OUT*, *LEN*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	BYTE	I, Q, M, V, L, SM
LEN	Input	BYTE	I, Q, M, V, L, SM, Constants
OUT	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

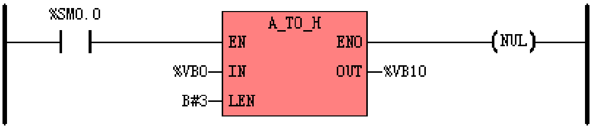
- **LD**

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

- **IL**

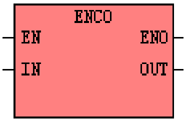
If CR is 1, this instruction is executed, and it does not influence CR.

- Examples

<table>
<tr><td><b>LD</b></td><td>
</td><td>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.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0
A_TO_H %VB0, %VB10, B#3</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is as the following:
<table>
<tr><th>VB0</th><th>VB1</th><th>VB2</th><th>VB10</th><th>VB11</th></tr>
<tr><td>51
‘3’</td><td>56
‘8’</td><td>54
‘6’</td><td>B#16#38</td><td>B#16#6x</td></tr>
<tr><td>55
‘7’</td><td>65
‘A’</td><td>49
‘1’</td><td>B#16#7A</td><td>B#16#1x</td></tr>
</table>
Note: x stands for this half byte (4 bits) keeps the original value.</td></tr>
</table>

6.7.14 ENCO (Encoding)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	ENCO			
IL	ENCO	ENCO *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	WORD	I, Q, M, V, L, SM, Constant
OUT	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.

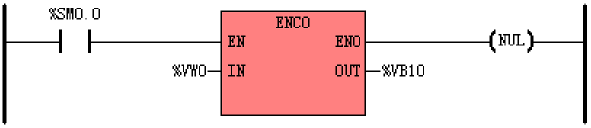
- **LD**

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

- **IL**

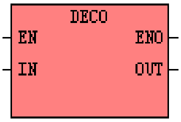
If CR is 1, this instruction is executed, and it does not influence CR.

- Examples

<table>
<tr><td><b>LD</b></td><td>
</td><td>SM0.0 is always 1, so ENCO is always executed: writes the bit number of the first bit equal to 1 into VB10.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0
ENCO %VW0, %VB10</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is as the following:
<table>
<tr><td></td><td colspan="16"><b>VW0</b></td><td><b>VB10</b></td></tr>
<tr><td>(MSB)</td><td>15</td><td></td><td></td><td>12</td><td></td><td></td><td>9</td><td></td><td></td><td></td><td></td><td>4</td><td></td><td></td><td></td><td>0 (LSB)</td><td></td></tr>
<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>B#9</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>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>B#4</td></tr>
</table></td></tr>
</table>

6.7.15 DECO (Decoding)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	DECO			
IL	DECO	DECO *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	BYTE	I, Q, M, V, L, SM, Constant
OUT	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.

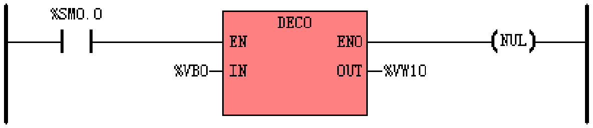
- **LD**

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

- **IL**

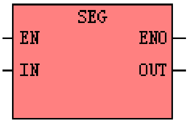
If CR is 1, this instruction is executed, and it does not influence CR.

- Examples

<table>
<tr><td><b>LD</b></td><td> </td><td>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.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0
DECO %VB0, %VW10</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is as the following:
<table>
<tr><td><b>VB0</b></td><td></td><td colspan="16"><b>VW10</b></td><td></td></tr>
<tr><td></td><td>(MSB)</td><td>15</td><td></td><td></td><td></td><td></td><td></td><td>9</td><td></td><td></td><td></td><td></td><td>4</td><td></td><td></td><td></td><td>0</td><td>(LSB)</td></tr>
<tr><td>B#9</td><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></td></tr>
<tr><td>B#16#D4</td><td></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><td>0</td><td></td></tr>
</table>
</td></tr>
</table>

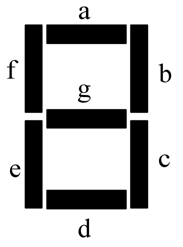
6.7.16 SEG (7-segment Display)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	SEG			
IL	SEG	SEG *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	BYTE	I, Q, M, V, L, SM, Constant
OUT	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*.

IN (LSD)	Display	*OUT* (- g f e d c b a)		*IN* (LSD)	Display	*OUT* (- 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**

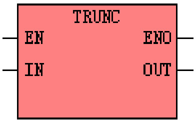
If *EN* is 1, this instruction is executed.

- **IL**

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

6.7.17 TRUNC (Truncate)

- Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	TRUNC			
IL	TRUNC	TRUNC *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	REAL	V, L, Constant
OUT	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.

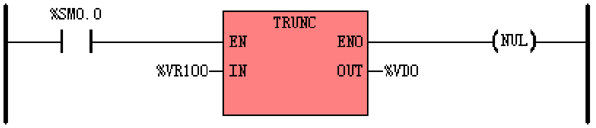
- **LD**

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

- **IL**

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

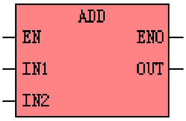
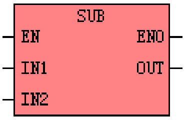
- Examples

LD	 %SM0.0 TRUNC EN ENO %VR100 IN OUT %VD0 NUL	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.
IL	LD %SM0.0 TRUNC %VR100, %VD0	
Result	The result is as the following: 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			☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	SUB			
IL	ADD	ADD *IN1*, *OUT*	U	
	SUB	SUB *IN1*, *OUT*		

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN1	Input	INT, DINT, REAL	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant
IN2	Input	INT, DINT, REAL	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant
OUT	Output	INT, DINT, REAL	Q, AQ, M, V, L, SM

- **LD**

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

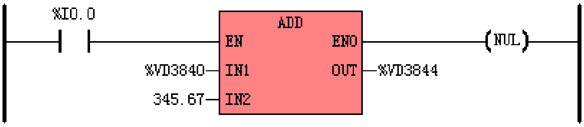
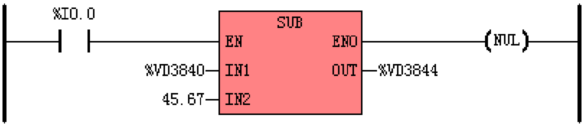
If *EN* is 1, the role that the *ADD* instruction plays is: *OUT=IN1+IN2*, and the role that the *SUB* instruction plays is: *OUT=IN1-IN2*.

- **IL**

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

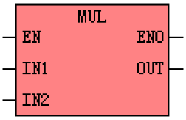
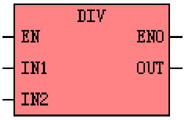
If CR is 1, the role that the *ADD* instruction plays is: *OUT=OUT+IN1*, and the role that the *SUB* instruction plays is: *OUT=OUT - IN1*. The *ADD* and *SUB* instructions won't influence CR.

- Examples

LD	 %I0.0 ADD EN ENO %VD3840 IN1 OUT %VD3844 345.67 IN2 NUL	If I0.0 is 0: *ADD* isn't executed. If I0.0 is 1: The instruction adds VD3840 and 345.67, and assigns the result to VD3844.
	 %I0.0 SUB EN ENO %VD3840 IN1 OUT %VD3844 45.67 IN2 NUL	If I0.0 is 0: *SUB* isn't executed. If I0.0 is 1: The instruction subtracts 45.67 from VD3840, and assigns the result to VD3844.
IL	LD %I0.0 ADD 345.67, %VD3840	(* CR is created with I0.0 *) (* If CR is 1: VD3840 = VD3840 +245.67 *) (* If CR is 0: the instruction isn't executed *)
	LD %I0.0 SUB 45.67, %VD3840	(* CR is created with I0.0 *) (* If CR is 1: VD3840 = VD3840 - 45.67 *) (* If CR is 0: the instruction isn't executed *)

6.8.2 MUL and DIV

➢ Description

	Name	Usage	Group	
LD	MUL			☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	DIV			
IL	MUL	MUL *IN1*, *OUT*	U	
	DIV	DIV *IN1*, *OUT*		

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN1	Input	INT, DINT, REAL	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant
IN2	Input	INT, DINT, REAL	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant
OUT	Output	INT, DINT, REAL	Q, AQ, M, V, L, SM

- **LD**

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

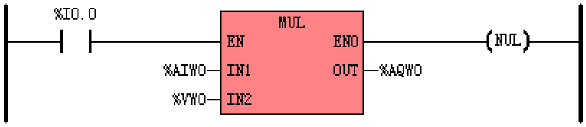
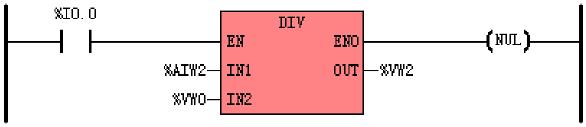
If *EN* is 1, the role that the *MUL* instruction plays is: *OUT=IN1×IN2*, and the role that the *DIV* instruction plays is: *OUT=IN1÷IN2*.

- **IL**

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

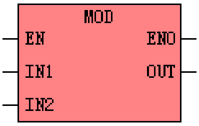
If CR is 1, the role that the *MUL* instruction plays is: *OUT=OUT×IN1*, and the role that the *DIV* instruction plays is: *OUT=OUT ÷ IN1*. The *MUL* and *DIV* instructions won't influence CR.

➢ Examples

LD	 %I0.0 MUL EN ENO %AIW0 IN1 OUT %AQW0 %VW0 IN2 NUL	If I0.0 is 0: *MUL* isn't executed. If I0.0 is 1: The instruction multiplies AIW0 and VW0, and assigns the result to AQW0.
	 %I0.0 DIV EN ENO %AIW2 IN1 OUT %VW2 %VW0 IN2 NUL	If I0.0 is 0: *DIV* isn't executed. If I0.0 is 1: The instruction divides AIW2 by VW0, and assigns the result to VW2.
IL	LD %I0.0 MUL %AIW0, %VW0	(* CR is created with I0.0 *) (* If CR is 1: VW0 = VW0 × AIW0 *) (* If CR is 0: the instruction isn't executed *)
	LD %I0.0 DIV %AIW2, %VW0	(* CR is created with I0.0 *) (* If CR is 1: VW0 = VW0 ÷ AIW2 *) (* If CR is 0: the instruction isn't executed *)

6.8.3 MOD (Modulo-Division)

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	MOD			
IL	MOD	MOD *IN1*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN1	Input	BYTE, INT, DINT	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant
IN2	Input	BYTE, INT, DINT	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant
OUT	Output	BYTE, INT, DINT	Q, AQ, M, V, L, SM

- **LD**

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

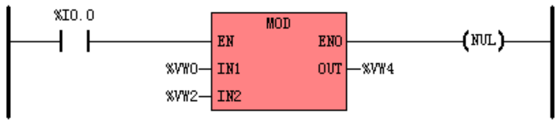
If *EN* is 1, this instruction divides *IN1* by *IN2*, and assigns the remainder to *OUT*.

- **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

<table>
<tr><td><b>LD</b></td><td>

%I0.0
MOD
EN
ENO
%VW0
IN1
OUT
%VW4
%VW2
IN2
NUL

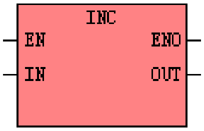
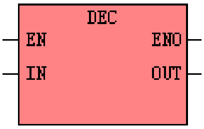
</td><td>If I0.0 is 0: <i>MOD</i> is not executed.
If I0.0 is 1: VW0 is divided by VW2, and the remainder is assigned to VW4.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %I0.0 (* CR is created with I0.0 *)
MOD %VW0, %VW4 (* If CR is 1: VW4 is divided by VW0, and the remainder is still stored in VW4 *)
(* If CR is 0: this instruction is not executed *)</td></tr>
<tr><td><b>Result</b></td><td colspan="2">For the LD example, if <i>MOD</i> instruction is executed, the result is shown as the following:

Address VW0 VW2
Value 8 3

Address VW4
Value 2</td></tr>
</table>

6.8.4 INC and DEC

➢ Description

	Name	Usage	Group	
LD	INC			☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	DEC			
IL	INC	INC *OUT*	U	
	DEC	DEC *OUT*		

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	BYTE, INT, DINT	I, Q, AI, AQ, M, V, L, SM, T, C, HC, constant
OUT	Output	BYTE, INT, DINT	Q, AQ, M, V, L, SM

- **LD**

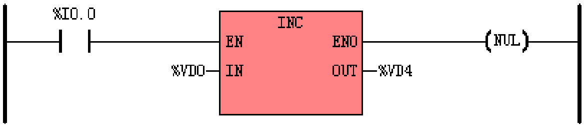
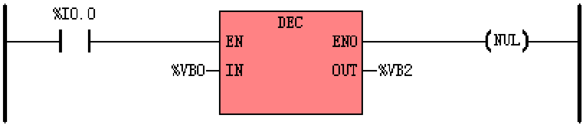
The *IN* and *OUT* must be of the same data type.

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**

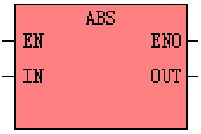
If CR is 1, the role that the *INC* instruction plays is: *OUT=OUT+1*, and the role that the *DEC* instruction plays: *OUT = OUT - 1*. They do not influence CR.

- Examples

LD	 %I0.0 INC EN ENO %VD0 IN OUT %VD4 NUL	If I0.0 is 0: *INC* isn't executed. If I0.0 is 1: VD4 = VD0 + DI#1.
	 %I0.0 DEC EN ENO %VB0 IN OUT %VB2 NUL	If I0.0 is 0: *DEC* isn't executed. If I0.0 is 1: VB2 = VB0 – B#1.
IL	LD %I0.0 (* CR is created with I0.0 *) INC %VD4 (* If CR is 1: VD4 =VD4 + DI#1 *) (* If CR is 0: this instruction isn't executed *)	
	LD %I0.0 (* CR is created with I0.0 *) DEC %VB2 (* If CR is 1: VB2 = VB2 - B#1 *) (* If CR is 0: this instruction isn't executed *)	

6.8.5 ABS (Absolute Value)

➢ Description

	Name	Usage	Group	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	ABS			
IL	ABS	ABS *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	INT, DINT, REAL	I, Q, V, M, L, SM, T, C, AI, AQ, HC, Constant, Pointer
OUT	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**

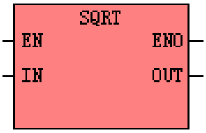
If *EN* is 1, this instruction is executed.

- **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	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	SQRT			
IL	SQRT	SQRT *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	REAL	V, L, Constant, Pointer
OUT	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**

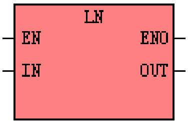
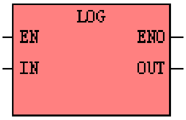
If *EN* is 1, this instruction is executed.

- **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			☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
	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**

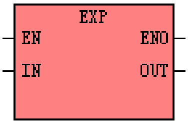
If *EN* is 1, this instruction is executed.

- **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	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	EXP			
IL	EXP	EXP *IN*, *OUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	REAL	V, L, Constant, Pointer
OUT	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**

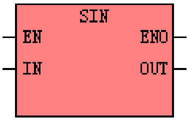
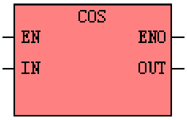
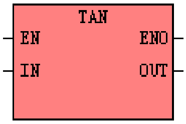
If *EN* is 1, this instruction is executed.

- **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			☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
	COS			
	TAN			
IL	SIN	SIN *IN*, *OUT*	U	
	COS	COS *IN*, *OUT*		
	TAN	TAN *IN*, *OUT*		

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	REAL	V, L, Constant, Pointer
OUT	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**

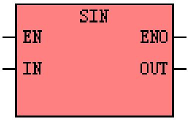
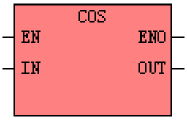
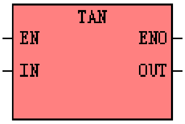
If *EN* is 1, this instruction is executed.

- **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			□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
	ACOS			
	ATAN			
IL	ASIN	ASIN *IN*, *OUT*	U	
	ACOS	ACOS *IN*, *OUT*		
	ATAN	ATAN *IN*, *OUT*		

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	REAL	V, L, Constant, Pointer
OUT	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* = 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* = 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* = 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)—		☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	JMP	lbl —(JMP)—		
	JMPC	lbl —(JMPC)—		
	JMPCN	lbl —(JMPCN)—		
IL	LBL	*lbl*:	U	
	JMP	JMP *lbl*		
	JMPC	JMPC *lbl*		
	JMPCN	JMPCN *lbl*		

Operand	Description
lbl	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 to the network label specified by *lbl*.

The *JMPC* instruction is used to transfer program execution to the network label specified by *lbl* when the horizontal link state on its left is true.

The *JMPCN* instruction is used to transfer program execution to the network label 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.

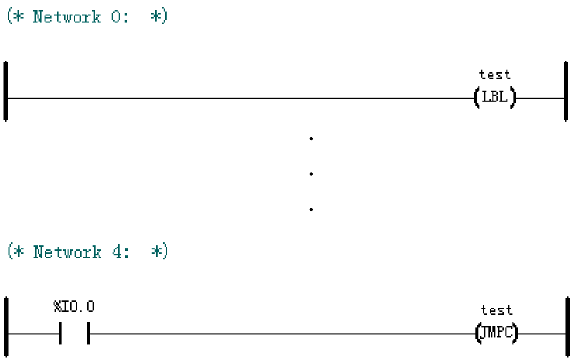
The *JMP* instruction is used to unconditionally transfer program execution to the label specified by *lbl*.

The *JMPC* instruction is used to transfer program execution to the label specified by *lbl* when CR is 1.

The *JMPCN* instruction is used to transfer program execution to the label specified by *lbl* when CR is 0.

The jump instruction and its destination label must always exist within the same POU.

➢ Examples

LD	IL
 (* Network 0: *) test LBL (* Network 4: *) %I0.0 test JMPC	(* NETWORK 0 *) test: … (* NETWORK 4 *) LD %I0.0 JMPC test

6.9.2 Return Instructions

Notice: Return instructions can only be used in subroutines and interrupt routines.

- Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	RETC	—(RETC)—		
	RETCN	—(RETCN)—		
IL	RETC	RETC	U	
	RETCN	RETCN		

- **LD**

The *RETC* instruction is used to terminate a subroutine or an interrupt routine and transfer program execution back to the calling entry when the horizontal link state on its left is true.

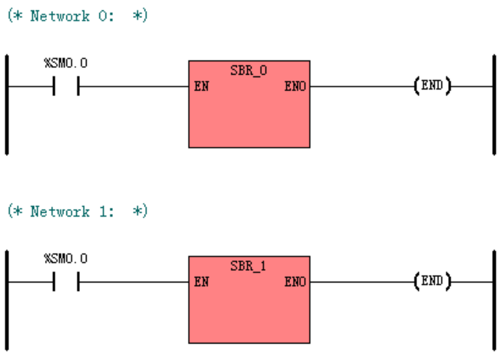
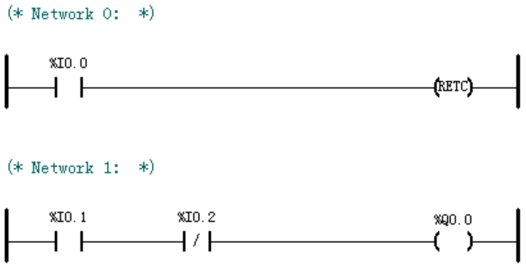
The *RETCN* instruction is used to terminate a subroutine or an interrupt routine and transfer program execution back to the calling entry when the horizontal link state on its left is false.

- **IL**

The *RETC* instruction is used to terminate a subroutine or an interrupt routine and transfer program execution back to the calling entry when CR is 1.

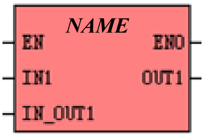
The *RETCN* instruction is used to terminate a subroutine or an interrupt routine and transfer program execution back to the calling entry when CR is 0.

- Examples

LD	**Main program:**  (* Network 0: *) %SM0.0 SBR_0 EN ENO END (* Network 1: *) %SM0.0 SBR_1 EN ENO END **Subroutine SBR_0:**  (* Network 0: *) %I0.0 RETC (* Network 1: *) %I0.1 %I0.2 %Q0.0	**For SBR_0:** If I0.0 is 0, the instructions are executed sequentially. If I0.0 is 1, program execution is transferred back to the calling entry in the main program, and the KINCO-K3 continues to execute the instructions in Network 1.
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 *) RETC (* If CR is 1, SBR_0 shall be terminate and program execution is transferred *) (* back to the calling 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	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	CAL			
IL	CAL	CAL *NAME*, actual parameter 1, actual parameter 2, …	U	

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**

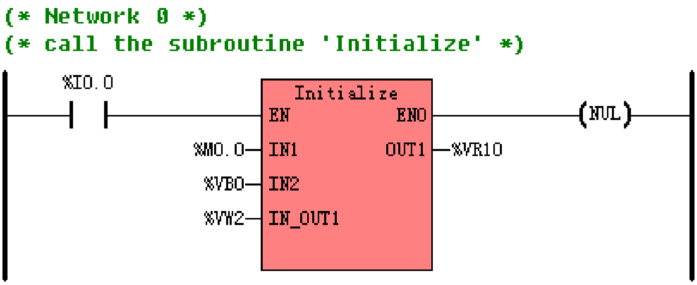
If CR is 1, the subroutine will be called and executed.

The CAL instruction does not influence CR, but CR may be changed in the subroutine.

➢ Examples

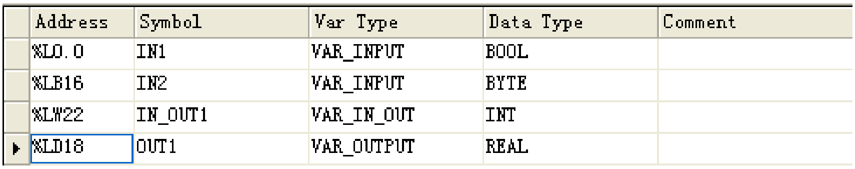
LD

Main program:



```
(* Network 0 *)
(* call the subroutine 'Initialize' *)
```

The Local Variable Table of the subroutine 'Initialize':



Address	Symbol	Var Type	Data Type	Comment
%L0.0	IN1	VAR_INPUT	BOOL	
%LB16	IN2	VAR_INPUT	BYTE	
%LW22	IN_OUT1	VAR_IN_OUT	INT	
%LD18	OUT1	VAR_OUTPUT	REAL	

IL

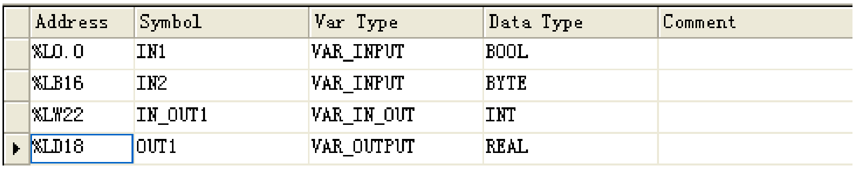
Main Program:

(* Network 0 *)

(*call the subroutine 'Initialize'*)

```
LD      %I0.0
CAL     Initialize, %M0.0, %VB0, %VW2, %VR10
```

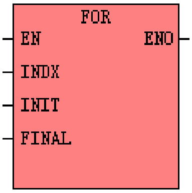
The Local Variable Table of the subroutine 'Initialize':



Address	Symbol	Var Type	Data Type	Comment
%L0.0	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			□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
	NEXT			
IL	FOR	FOR *INDX, INIT, FINAL*	U	
	NEXT	NEXT		

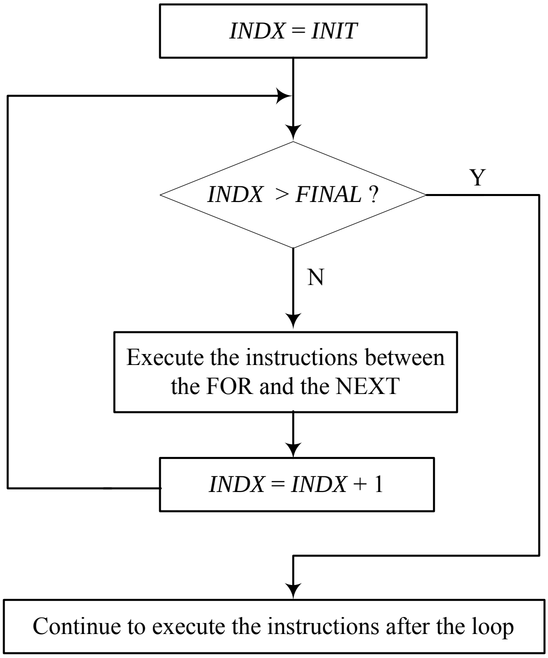
Operands	Input/Output	Data Type	Acceptable Memory Areas
INDX	Input	INT	M, V, L, SM
INIT	Input	INT	M, V, L, SM, T, C, Constant
FINAL	Output	INT	M, V, L, SM, T, C, Constant

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:


INDX = *INIT*
INDX > *FINAL* ?
Y
N
Execute the instructions between the FOR and the NEXT
INDX = *INDX* + 1
Continue to execute the instructions after the loop

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.

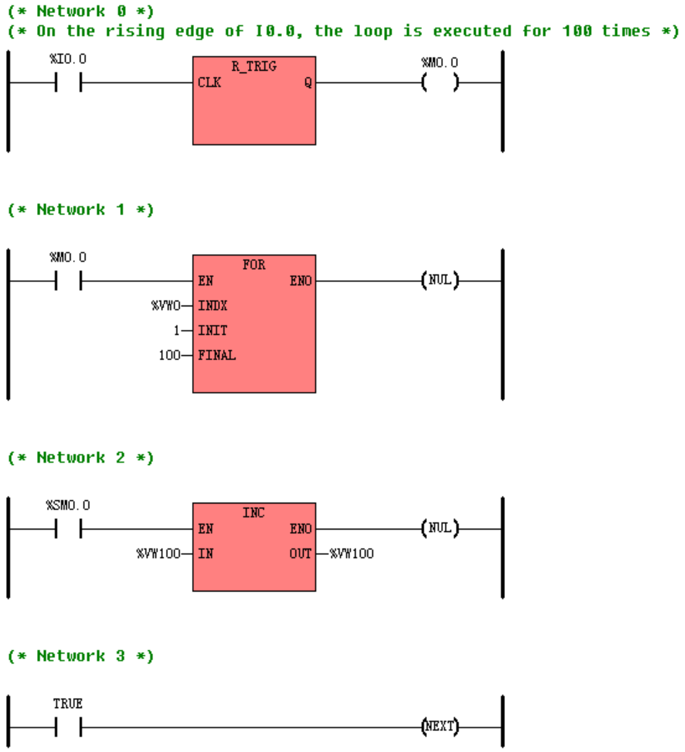
- **LD**

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

- **IL**

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

- **Example**

LD	 (* Network 0 *) (* On the rising edge of I0.0, the loop is executed for 100 times *) %I0.0 R_TRIG CLK Q %M0.0 (* Network 1 *) %M0.0 FOR EN ENO %VW0 INDX 1 INIT 100 FINAL NUL (* Network 2 *) %SM0.0 INC EN ENO %VW100 IN OUT %VW100 NUL (* Network 3 *)
IL	(* Network 0 *) (*On the rising edge of I0.0, the loop is executed for 100 times*) LD %I0.0 R_TRIG ST %M0.0 (* Network 1 *) LD %M0.0 FOR %VW0, 1, 100 (* Network 2 *) LD %SM0.0 INC %VW100 (* Network 3 *) LD TRUE NEXT

6.9.5 END (Terminate the scan cycle)

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	END	—(END)—		
IL	END	END	U	

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	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	STOP	—(STOP)—		
IL	STOP	STOP	U	

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	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	WDR	—(WDR)—		
IL	WDR	WDR	U	

This instruction re-triggers the system watchdog timer of the CPU.

Using the WDR instructin 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-K3 to quickly respond to special internal or external predefined events. The KINCO-K3 supports tens of events each of which is assigned with a unique event number.

If you want to enable an interrupt, you must use the *ATCH* instruction to att

You can use the *DTCH* instruction to break the attachment between the interrupt eve ach an interrupt event (specified by the event number) to the interrupt routine (specified by the routine name) that you want to execute when the event occurs. nt and the interrupt routine. The *Detach* instruction makes the interrupt return to be disabled.

6.10.1 How the KINCO-K3 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-K3 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-K3

The KINCO-K3 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	Priority
32	PORT 1: XMT complete	Communication Port Interrupts	Highest
31	PORT 1: RCV complete		↓
30	PORT 0: XMT complete		
29	PORT 0: RCV complete		
28	PTO 0 complete	I/O Interrupts	
27	PTO 1 complete		
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	HSC2 CV=PV		
11	HSC2 direction changed		
10	HSC2 external reset		
9	HSC3 CV=PV		
8	HSC4 CV=PV		
7	HSC4 direction changed		
6	HSC4 external reset		
5	HSC5 CV=PV		

4	Timed interrupt 1. Its period is specified in SMW24, unit: ms, range: 1~65535ms.	Time Interrupts	
3	Timed interrupt 0. Its period is specified in SMW22, unit: ms, range: 1~65535ms.		
2	Timer T3　ET=PT		
1	Timer T2　ET=PT		Lowest

Table 6-1 Interrupt Events

6.10.5 ENI (Enable Interrupt), DISI (Disable Interrupt)

➢ Description

	Name	Usage	Group	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	ENI	—(ENI)—		
	DISI	—(DISI)—		
IL	ENI	ENI	U	
	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.

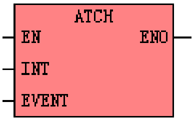
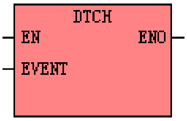
- **IL**

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			☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	DTCH			
IL	ATCH	ATCH *INT*, *EVENT*	U	
	DTCH	DTCH *EVENT*		

Operands	Input/Output	Data Type	Description
INT	Input		The name of an existing interrupt routine
EVENT	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.

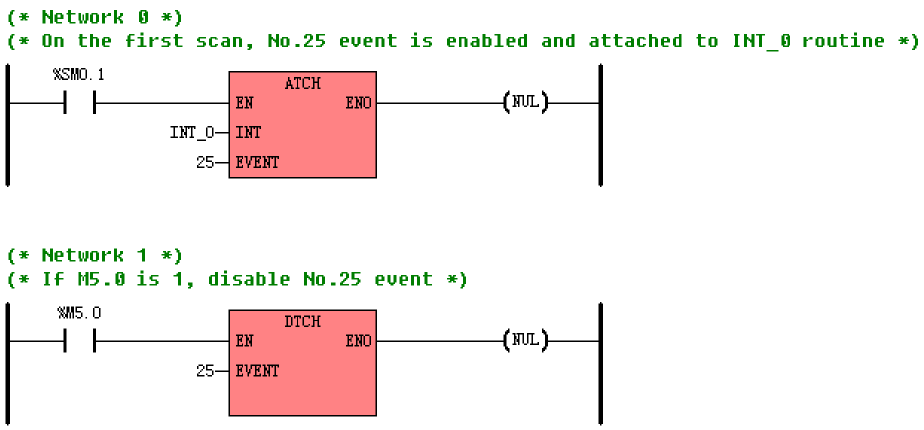
If *EN* is 1, the *DTCH* instruction breaks the attachment between the interrupt event (specified by the event number *EVENT*) and its interrupt routine, and makes the interrupt event return to be disabled.

- **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

LD	 (* Network 0 *) (* On the first scan, No.25 event is enabled and attached to INT_0 routine *) %SM0.1 ATCH EN ENO INT_0 INT 25 EVENT NUL (* Network 1 *) (* If M5.0 is 1, disable No.25 event *) %M5.0 DTCH EN ENO 25 EVENT NUL
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 *)

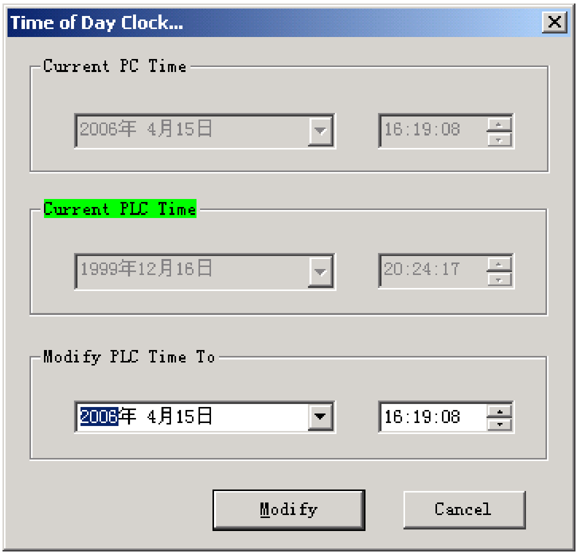
6.11 Clock Instructions

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 conducts 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.



Time of Day Clock...

Current PC Time: 2006年 4月15日 16:19:08

Current PLC Time: 1999年12月16日 20:24:17

Modify PLC Time To: 2006年 4月15日 16:19:08

Figure 6-1 Adjusting the RTC

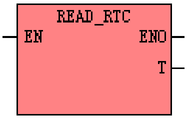
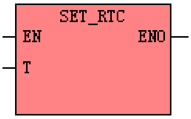
- **Current PC Time**: Indicate the current date and time of the current PC.
- **Current PLC Time**: Indicate the current date and time of the RTC of the online CPU module. Its background being green indicates that the CPU module communicates with the PC successfully, and its

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.
- **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.

6.11.2 READ_RTC and SET_RTC

- Description

	Name	Usage	Group	
LD	READ_RTC			☐ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	SET_RTC			
IL	READ_RTC	READ_RTC *T*	U	
	SET_RTC	SET_RTC *T*		

Operands	Input/Output	Data Type	Acceptable Memory Areas
T	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, BCD coding, hereinafter the same.

Table 6-2 The Time Buffer

Notice:

(1) You are recommended to adjust the RTC correctly using [**PLC**]>[**Time of Day Clock…**] menu command before using it.

(2) 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.

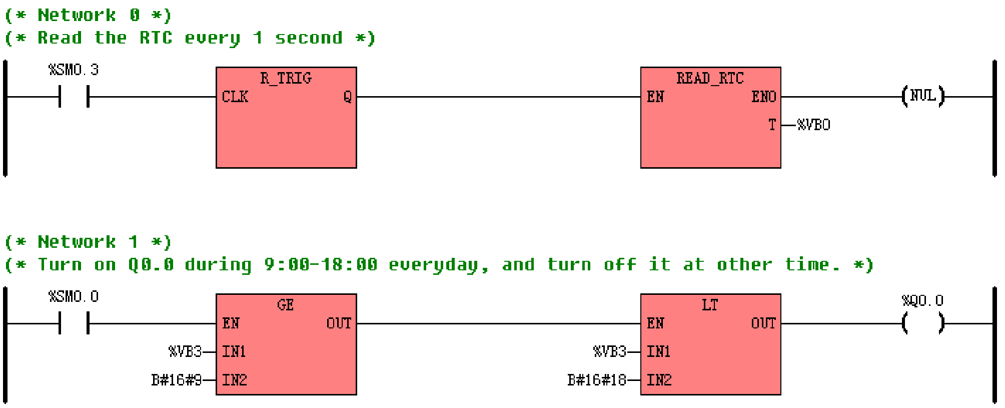
- **LD**

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	 (* Network 0 *) (* Read the RTC every 1 second *) %SM0.3 R_TRIG CLK Q READ_RTC EN ENO T %VB0 NUL (* Network 1 *) (* Turn on Q0.0 during 9:00-18:00 everyday, and turn off it at other time. *) %SM0.0 GE EN OUT %VB3 IN1 B#16#9 IN2 LT EN OUT %VB3 IN1 B#16#18 IN2 %Q0.0
IL	(* Network 0 *) (*Read the RTC every 1 second*) LD %SM0.3 R_TRIG READ_RTC %VB0 (* Network 1 *) (*Turn on Q0.0 during 9:00-18:00 everyday, and turn off it at other time.*) LD %SM0.0 GE %VB3, B#16#9 LT %VB3, B#16#18 ST %Q0.0

6.12 Communication Instructions

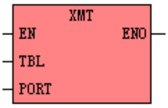
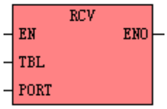
These instructions are used for free-protocol communication. Free-protocol communication mode allows your program to entirely control the communication ports of the CPU. You can use free-protocol communication mode to implement user-defined communication protocols to communicate with all kinds of intelligent devices. ASCII and binary protocols are both supported.

The CPU module is integrated with 1 or 2 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 3.8 How to modify the CPU's communication parameters for detailed information.

6.12.1 XMT and RCV

➢ Description

	Name	Usage	Influence	
LD	XMT			☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	RCV			
IL	XMT	XMT *TBL*, *PORT*	U	
	RCV	RCV *TBL*, *PORT*		

Operands	Input/Output	Data Type	Acceptable Memory Areas
TBL	Input	BYTE	I, Q, M, V, L, SM
PORT	Input	INT	Constant (0 or 1)

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 and 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.

In IL, CR decides whether to execute the *XMT* and *RCV* instructions. They won't influence CR.

➢ Status Registers and Control Registers in SM area for Free-protocol Communication

Besides XMT and RCV instructions, som 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.

(1) SMB86 --- Receive Status Register

Bit (read-only)		Status	Description
PORT 0	PORT 1		
SM86.0		1	A parity error is detected, but receive shall not be terminated.
SM86.1		1	Receive was terminated because of receiving the maximum character number. (see SMB94)
SM86.2		1	Receive was terminated because of receiving a character Overtime. (See SMW92)
SM86.3		1	Receive was terminated because of System Overtime.
SM86.4		-	Reserved.
SM86.5		1	Receive was terminated because of receiving the user-defined End character (see SMB89).
SM86.6		1	Receive was terminated because of the errors in the parameters or missing the Start or End condition.
SM86.7		1	Receive was terminated because of the user disable command (See SM87.7)

(2) SMB87 --- Receive Control Register

Bit		Status	Description
PORT 0	PORT 1		
SM87.0		-	Reserved.
SM87.1		0	Disenable XMT-complete and RCV-complete interrupts.
		1	Enable XMT-complete and RCV-complete interrupts.
SM87.2		0	Ignore SMW92.
		1	Terminate receive if the time in SMW92 is exceeded while receiving a character.
SM87.3		-	Reserved.
SM87.4		0	Ignore SMW90.
		1	Turn to effective receive if the time interval in SMW90 is exceeded.
SM87.5		0	Ignore SMB89.
		1	Enable the user-defined End character in SMB89.

SM87.6		0	Ignore SMB88.
		1	Enable the user-defined Start character in SMB88
SM87.7		0	Disenable RCV function. This condition prevails over any other conditions.
		1	Enable RCV function.

(3) Other Control Registers

PORT 0	PORT 1	Description
SMB88		To store the user-defined receive Start character. After executing the *RCV* 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. SM87.6 should be set to be 1 to enable SMB88.
SMB89		To store the user-defined receive End character. 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. SM87.5 should be set to be 1 to enable SMB89.
SMW90		To store the user-defined receive Ready time (Range: 1~60,000ms). After executing the *RCV* 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. SM87.4 should be set to be 1 to enable SMW90.
SMW92		To store the user-defined receiving a character Overtime (Range: 1~60,000ms). After executing the *RCV* 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. SM87.2 should be set to be 1 to enable SMW92.

SMW94		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. If this value is set to be 0, the *RCV* instruction will return directly.

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.

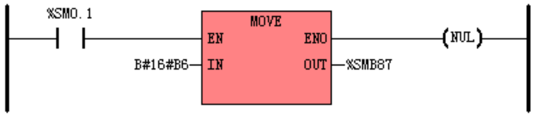
- Examples

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.

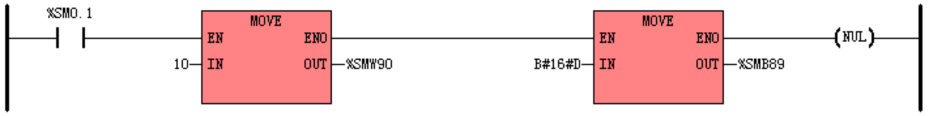
LD

MAIN Program:

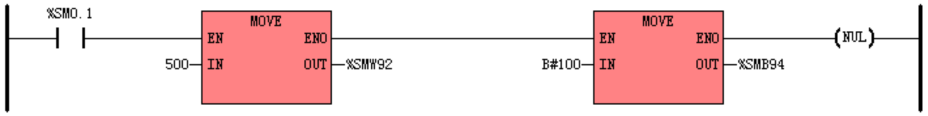
```
(* Network 0 *)
(* The following program is to initialize free-protocol communication.
At first,configure the Start and End conditions of the effective Receive state. *)
```



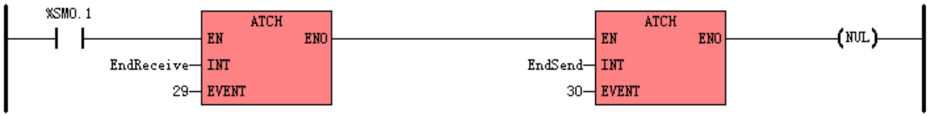
```
(* Network 1 *)
(* The receive Ready time is set to be 10ms,
The receive End character is set to be RETURN character whose ASCII is 13. *)
```



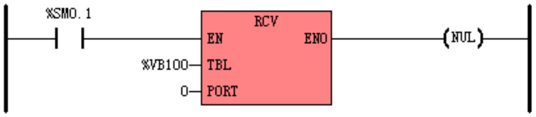
```
(* Network 2 *)
(* The receiving a character Overtime is set to be 500ms,
The maximum number of characters to be received is set to be 100. *)
```



```
(* Network 3 *)
(* Attach the RCV-complete event to the EndReceiver routine,
Attach the XMT-complete event to the EndSendroutine *)
```



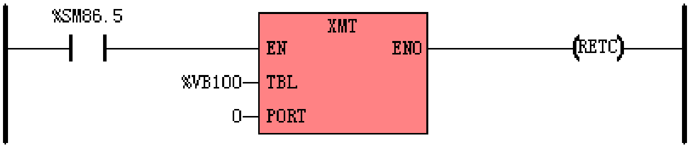
```
(* Network 4 *)
(* Start the Receive task once on the first scan. *)
```



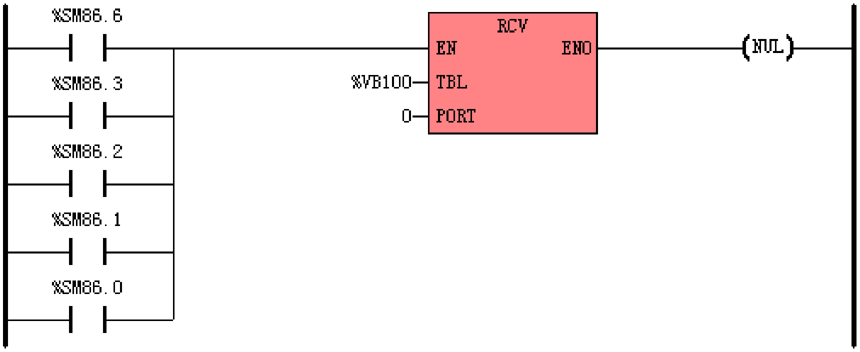
LD

EndReceive (INT00): The RCV-complete interrupt routine

```
(* Network 0 *)
(* If receiving the receive End character,
then transmit bach the data received and return. *)
```

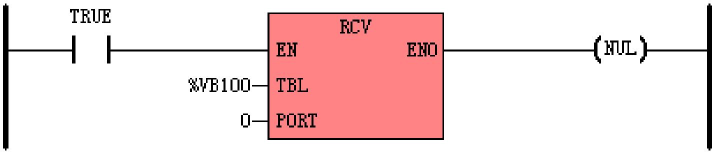


```
(* Network 1 *)
(* if receive is completed abnormally, then restart receive. *)
```



EndSend (INT01): XMT-complete interrupt routine

```
(* Network 0 *)
(* Restart receive after the transmition 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 transmition is completed. *)
LD          TRUE
RCV          %VB100, 0
```

6.12.2 Modbus RTU Master Instructions

The Modbus RTU protocol is widely used in the industrial field. The KINCO-K3 provides the Modbus RTU Master instructions, and you can call them directly to make the KINCO-K3 as a Modbus RTU master.

Note: these instructions are supported only by PORT1.

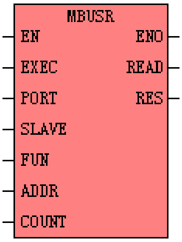
The general steps of the Modbus master programming are described as followings:

（1） Configure the communication parameters of Port1 in the **Hardware** Window. Please refer to 3.8 How to modify the CPU's communication parameters and 4.3.3.1 Parameters of the CPU for more details.

（2） Call the instructions MBUSR and MBUSW in the program.

6.12.2.1 MBUSR (Modbus RTU Master Read)

➢ Description

	Name	Usage	Group	
LD	MBUSR	 MBUSR EN ENO EXEC READ PORT RES SLAVE FUN ADDR COUNT		□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
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 (1)
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.

MSB							LSB
7	6	5	4	3	2	1	0

Bit 7 --- Indicates whether the communication has been finished or not: 0 = not finished, 1 = finished.
Bit 6 --- Reserved。
Bit 5 --- Illegal *SLAVE*.
Bit 4 --- Illegal *COUNT*.
Bit 3 --- Illegal *ADDR*.
Bit 2 --- 1 = The specified port is busy.
Bit 1 --- 1 = Time out
Bit 0 --- 1 = The received message is wrong because of CRC error, frame error, etc.

- **LD**

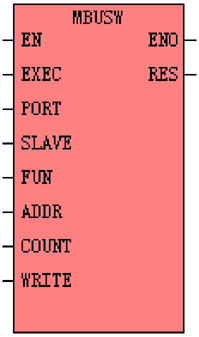
If *EN* is 1, this instruction is executed.

- **IL**

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

6.12.2.2 MBUSW (Modbus RTU Master Write)

- Description

	Name	Usage	Group	
LD	MBUSW	 MBUSW EN ENO EXEC RES PORT SLAVE FUN ADDR COUNT WRITE		□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
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 (1)
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.

MSB							LSB
7	6	5	4	3	2	1	0

Bit 7 --- Indicates whether the communication has been finished or not: 0 = not finished, 1 = finished.
Bit 6 --- Reserved。
Bit 5 --- Illegal *SLAVE*.
Bit 4 --- Illegal *COUNT*.
Bit 3 --- Illegal *ADDR*.
Bit 2 --- 1 = The specified port is busy.
Bit 1 --- 1 = Time out
Bit 0 --- 1 = The received message is wrong because of CRC error, frame error, etc.

- **LD**

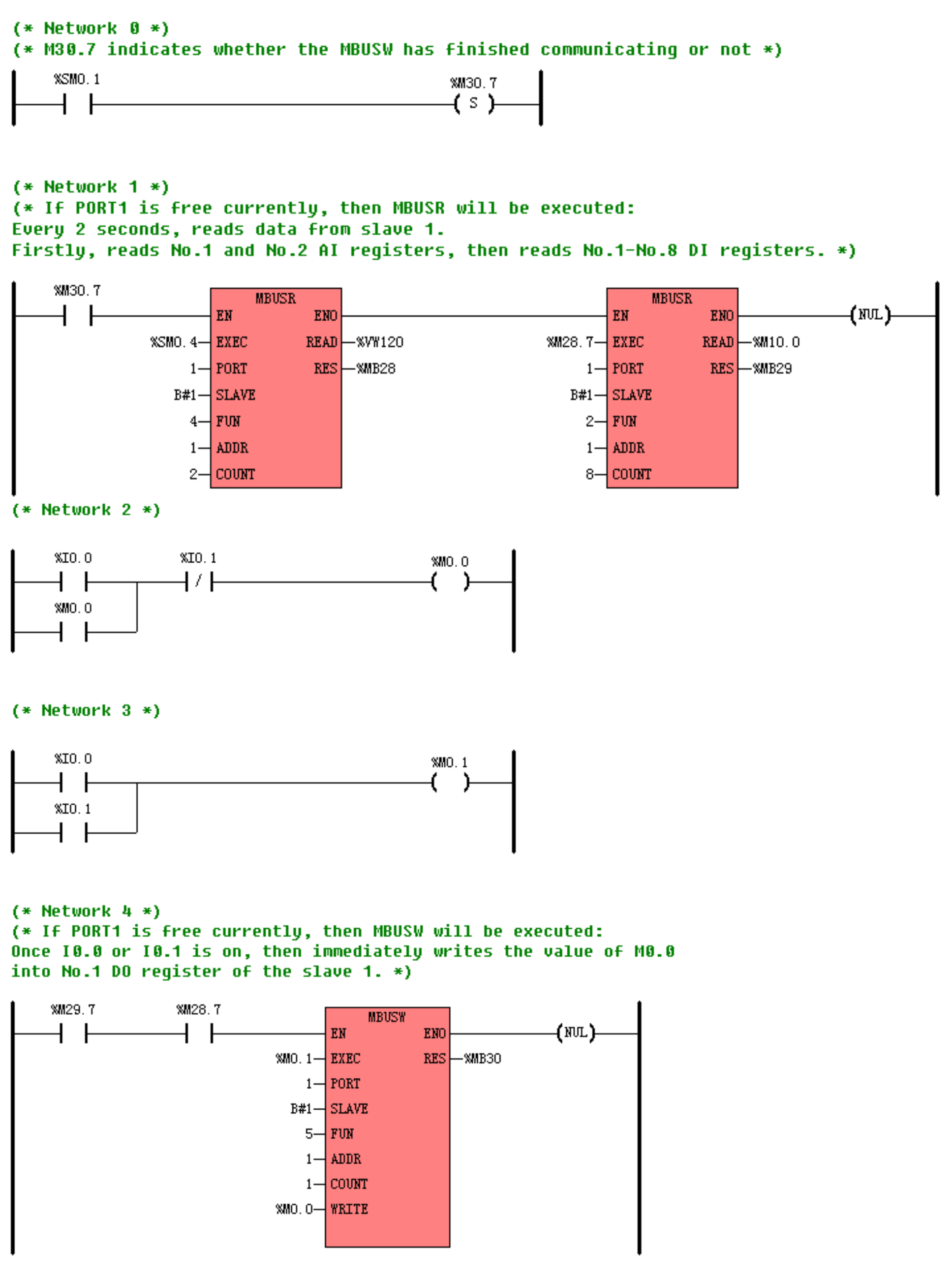
If *EN* is 1, this instruction is executed.

- **IL**

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

6.12.2.3 Example for MBUSR and MBUSW

LD



```
(* Network 0 *)
(* M30.7 indicates whether the MBUSW has finished communicating or not *)

(* Network 1 *)
(* If PORT1 is free currently, then MBUSR will be executed:
Every 2 seconds, reads data from slave 1.
Firstly, reads No.1 and No.2 AI registers, then reads No.1-No.8 DI registers. *)

(* Network 2 *)

(* Network 3 *)

(* Network 4 *)
(* If PORT1 is free currently, then MBUSW will be executed:
Once I0.0 or I0.1 is on, then immediately writes the value of M0.0
into No.1 DO register of the slave 1. *)
```

IL

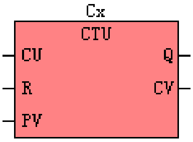
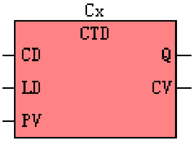
```
(* Network 0 *)
(* M30.7 indicates whether the MBUSW has finished communicating or not*)
LD          %SM0.1
S           %M30.7
(* Network 1 *)
(* If PORT1 is free currently, then MBUSR will be executed: *)
(* Every 2 seconds, reads data from slave 1. *)
(* Firstly, reads No.1 and No.2 AI registers, then reads No.1-No.8 DI registers.*)
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
(* Network 2 *)
LD          %I0.0
OR          %M0.0
ANDN        %I0.1
ST          %M0.0
(* Network 3 *)
LD          %I0.0
OR          %I0.1
ST          %M0.1
(* Network 4 *)
(* If PORT1 is free currently, then MBUSW will be executed: *)
(* Once I0.0 or I0.1 is on, then immediately writes the value of M0.0 *)
(* into No.1 DO register of the slave 1.*)
LD          %M29.7
AND         %M28.7
MBUSW       %M0.1, 1, B#1, 5, 1, 1, %M0.0, %MB30
```

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 2.6.5 Function Block and Function Block Instance for more detailed information.

- Description

	Name	Usage	Group	
LD	CTU			☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	CTD			
IL	CTU	CTU *Cx*, *R*, *PV*	P	
	CTD	CTD *Cx*, *LD*, *PV*		

Operands	Input/Output	Data Type	Acceptable Memory Areas
Cx	-	Counter instance	C
CU	Input	BOOL	Power flow
R	Input	BOOL	I, Q, M, V, L, SM, T, C, RS, SR
CD	Input	BOOL	Power flow
LD	Input	BOOL	I, Q, M, V, L, SM, T, C, RS, SR
PV	Input	INT	I, Q, M, V, L, SM, AI, AQ, constant
Q	Output	BOOL	Power flow
CV	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).

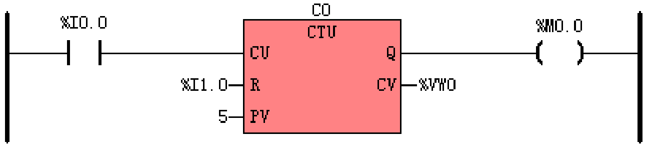
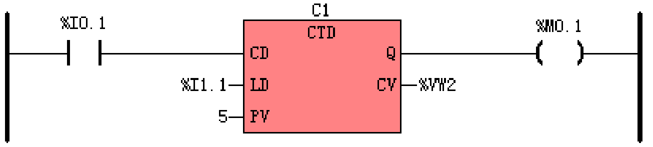
The *CTD* counter counts down on the rising edge of the *CD* input. When the current value *CV* is equal to or greater than the preset value *PV*, both the counter output *Q* and the status bit of *Cx* are set to be 1. *Cx* is reset and *PV* is loaded into *CV* when the load input *LD* is enabled. When the counter reaches *PV*, it continues counting until it reaches and keeps at 0.

- **IL**

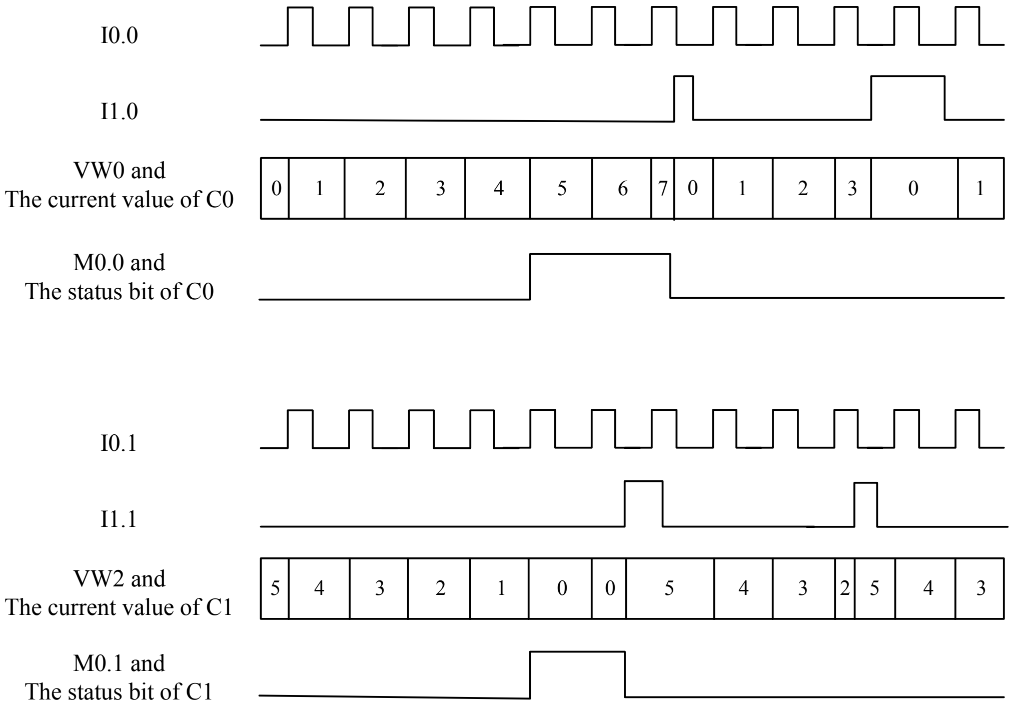
The *CTU* counter counts up on the rising edge of CR. When the current value of *Cx* is equal to or greater than the preset value *PV*, the counter status bit are set to be 1. *Cx* is reset when the reset input *R* is enabled. When the counter reaches *PV*, it continues counting until it reaches and 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

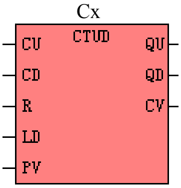
LD	IL
(* Network 0: *)  C0 CTU %I0.0 CU Q %M0.0 %I1.0 R CV %VW0 5 PV	(* NETWORK 0 *) LD %I0.0 CTU C0, %I1.0, 5 ST %M0.0
(* Network 1: *)  C1 CTD %I0.1 CD Q %M0.1 %I1.1 LD CV %VW2 5 PV	(* NETWORK 1 *) LD %I0.1 CTD C1, %I1.1, 5 ST %M0.1

Time Sequence Diagram



6.13.2 CTUD (Up-Down Counter)

- Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	CTUD			
IL	CTUD	CTUD *Cx, CD, R, LD, PV, QD*	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
Cx	-	Counter instance	C
CU	Input	BOOL	Power flow
CD	Input	BOOL	I, Q, M, V, L, SM, T, C, RS, SR
R	Input	BOOL	I, Q, M, V, L, SM, T, C, RS, SR
LD	Input	BOOL	I, Q, M, V, L, SM, T, C, RS, SR
PV	Input	INT	I, Q, M, V, L, SM, AI, AQ, constant
QU	Output	BOOL	Power flow
QD	Output	BOOL	Q, M, V, L, SM
CV	Output	INT	Q, M, V, L, SM, AQ

- **LD**

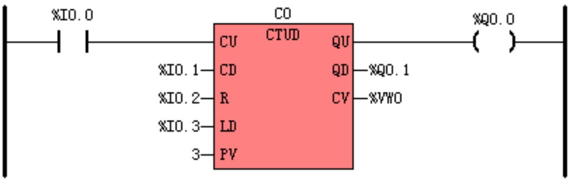
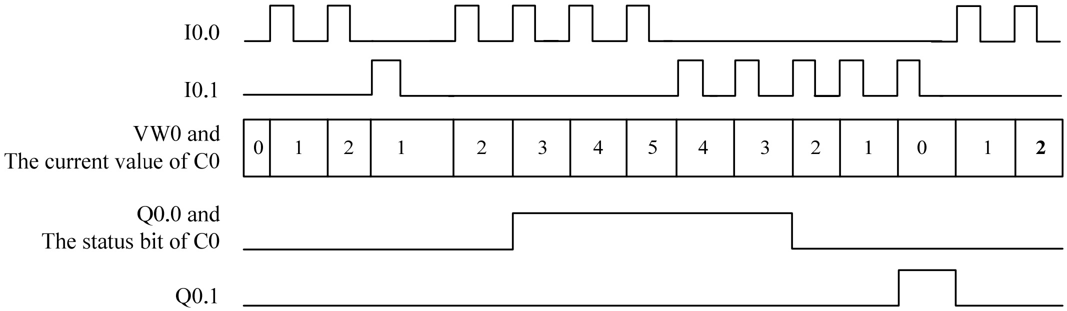
The *CTUD* counter counts up on the rising edge of the *CU* input and counts down on the rising edge of the *CD* input, and the current counter value *Cx* is assigned to *CV*. When *CV* is equal to or greater than the preset value *PV*, both *QU* and the status bit of *Cx* are set to 1, otherwise they 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*.

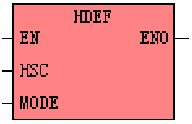
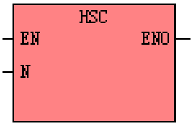
➢ Example

LD	**IL**
 %I0.0 C0 CTUD CU QU %Q0.0 %I0.1 CD QD %Q0.1 %I0.2 R CV %VW0 %I0.3 LD 3 PV	LD %I0.0 CTUD C1, %I0.1, %I0.2, %I0.3, 4 ,%Q0.1 ST %Q0.0
Time Sequence Diagram	
	

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			☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
	HSC			
IL	HDEF	HDEF *HSC*, *MODE*	U	
	HSC	HSC *N*		

Operands	Input/Output	Data Type	Description
HSC	Input	INT constant (0~5)	HSC number
MODE	Input	INT constant (0~11)	Operations mode
N	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.

6.13.3.1 High-speed Couters Supported by the KINCO-K3

Feature	CPU304	CPU306
High-speed counters	2 counters (HSC0 and HSC1)	6 counters (HSC0 to HSC5)
Single phase	2 at 20KHz	6 at 30KHz
Two phase	2 at 10KHz	4 at 20KHz.

HSC3 and HSC5 have one operation mode; HSC0 and HSC4 have 7 operation modes; and 11 modes for HSC1 and HSC2. All the high-speed counters have the same function in the same operation mode.

Each input of a high-speed counter functions as follows:

- When the reset input is true, it clears the current value all along until it is false.
- When the start input is true, the counter is allowed to count. When it is false, the current value remains unchanged and the clock inputs shall be ignored.
- If the reset input is true and the start input is false, the reset input is ignored and the current value remains unchanged. If the start input and the reset input are all true, the current value shall be cleared.
- For the single-phase counter with external direction control, if the direction input is true, the counter counts up. If the direction input is false, the counter counts down.

6.13.3.2 Operation Modes and Inputs of the High-speed Counters

HSC 0				
Mode	**Description**	**I0.1**	**I0.0**	**I0.5**
0	Single-phase up/down counter with internal direction control	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 B	Clock A	

HSC 1

Mode	Description	I0.3	I0.7	I1.2	I1.3
0	Single-phase up/down counter with internal direction control			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 B	Clock A
10		Reset			
11		Reset	Start		

HSC 2					
Mode	**Description**	**I0.6**	**I1.1**	**I1.4**	**I1.5**
0	Single-phase up/down counter with internal direction control			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 B	Clock A
10		Reset			
11		Reset	Start		

HSC 3		
Mode	**Description**	**I0.0**

0	Single-phase up/down counter with internal direction control	Clock

HSC 4				
Mode	**Description**	**I0.2**	**I1.0**	**I1.1**
0	Single-phase up/down counter with internal direction control	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 Down	Clock Up	
9	A/B phase quadrature counter	Clock B	Clock A	

HSC 5		
Mode	**Description**	**I0.3**
0	Single-phase up/down counter with internal direction control	Clock

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.

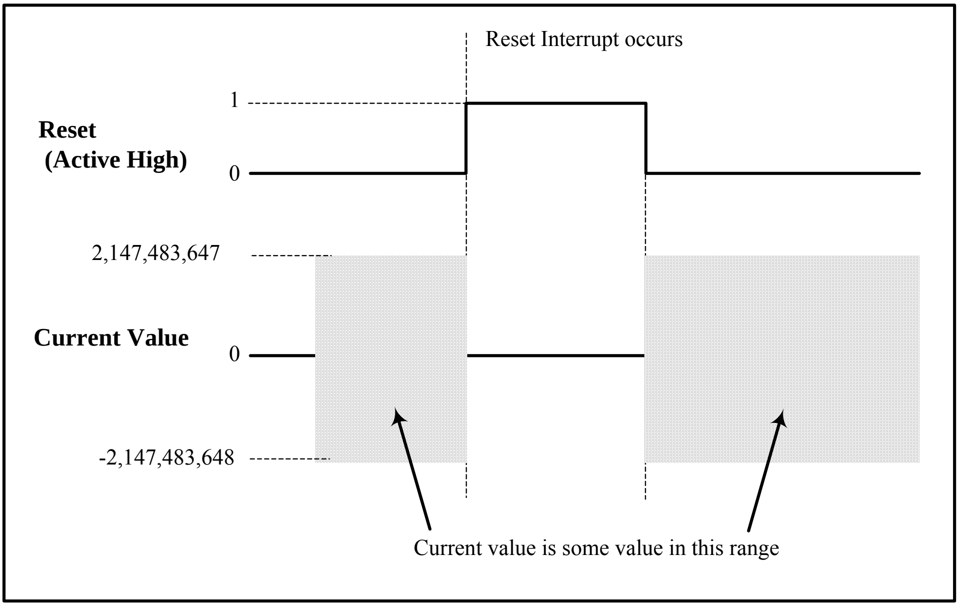


Figure 6-2 Time Sequence with Reset and without Start

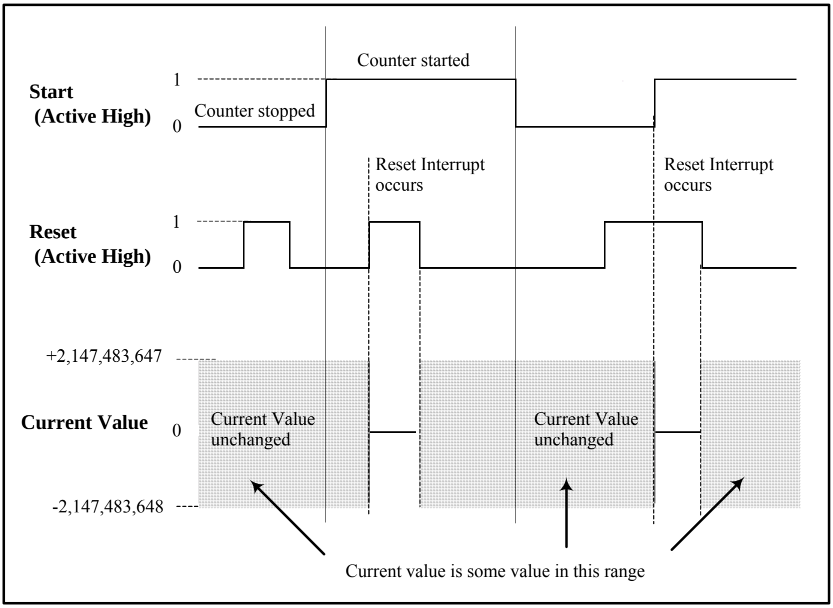


Figure 6-3 Time Sequence with Reset and Start

HSC0, HSC1, HSC2 and HSC4 have 3 control bits which are used to select the active level of the reset and start inputs and to select 1x or 4x counting rate (limited to quadrature counters). These bits are in the control byte of the relevant counter, and they take effect only when the *HSC* instruction is executed.

These bits are described in the following table.

HSC0	**HSC1**	**HSC2**	**HSC4**	**Description**
SM37.0	SM47.0	SM57.0	SM147.0	Control bit for active level of Reset: 0 = Active High; 1 = Active Low
SM37.1	SM47.1	SM57.1	SM147.1	Control bit for active level of Start: 0 = Active High; 1 = Active Low
SM37.2	SM47.2	SM57.2	SM147.2	Control bit for counting rate of quadrature counter: 0 = 4x counting rate; 1 = 1x counting rate

Table 6-3 Active Level for Reset, Start and 1x/4x Control Bits

Before executing the *HSC* instruction, these control bits must be set to the expected status. Otherwise, the counter will select the default settings for the operation mode selected, and the default settings are: the reset input and the start input are active high, and the quadrature counting rate is 1x (one time the input clock frequency). Once the *HSC* instruction is executed, the counter configuration cannot be modified.

- **Examples of All Modes**

The following time sequence diagrams show how a counter operates according to its mode.

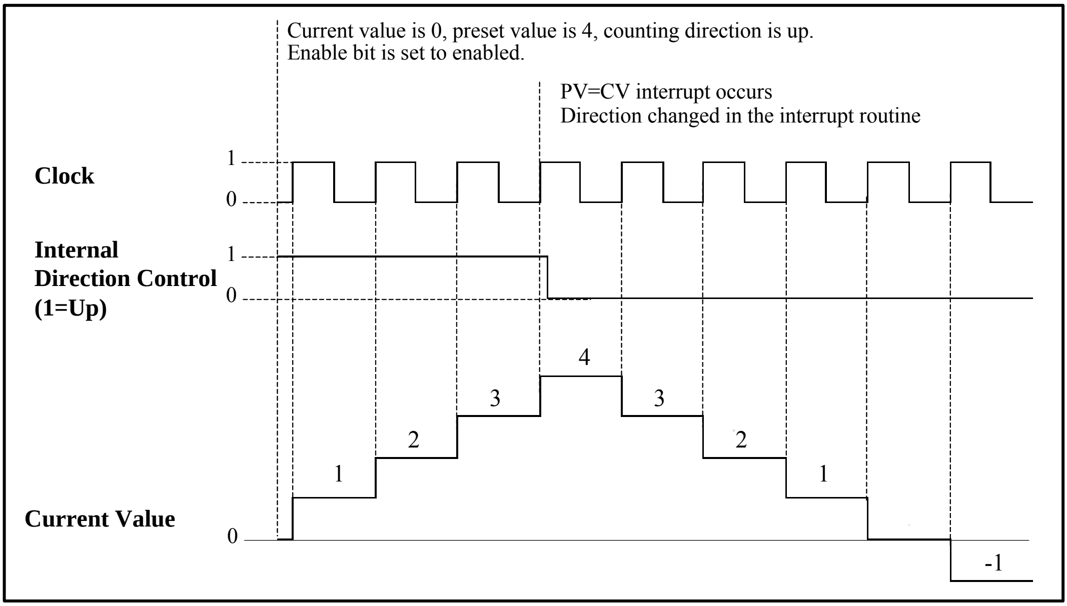

Current value is 0, preset value is 4, counting direction is up.
Enable bit is set to enabled.
PV=CV interrupt occurs
Direction changed in the interrupt routine
Clock
1
0
Internal
Direction Control
(1=Up)
1
0
Current Value
0
1
2
3
4
3
2
1
-1

Figure 6-4 Time Sequence of Mode 0, 1 or 2

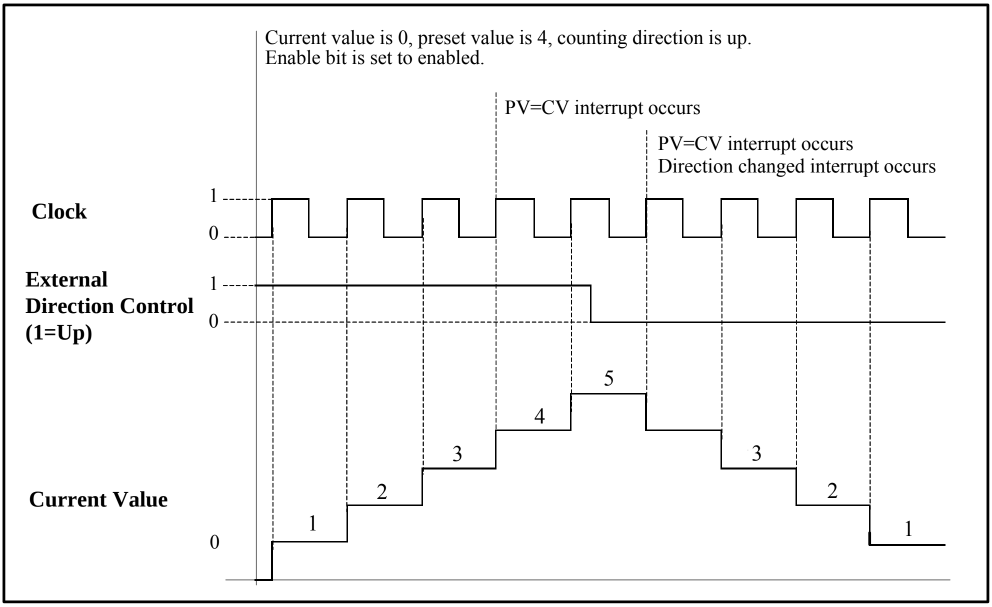

Current value is 0, preset value is 4, counting direction is up.
Enable bit is set to enabled.
PV=CV interrupt occurs
PV=CV interrupt occurs
Direction changed interrupt occurs
Clock
1
0
External
Direction Control
(1=Up)
1
0
Current Value
0
1
2
3
4
5
3
2
1

Figure 6-5 Time Sequence of Mode 3, 4 or 5

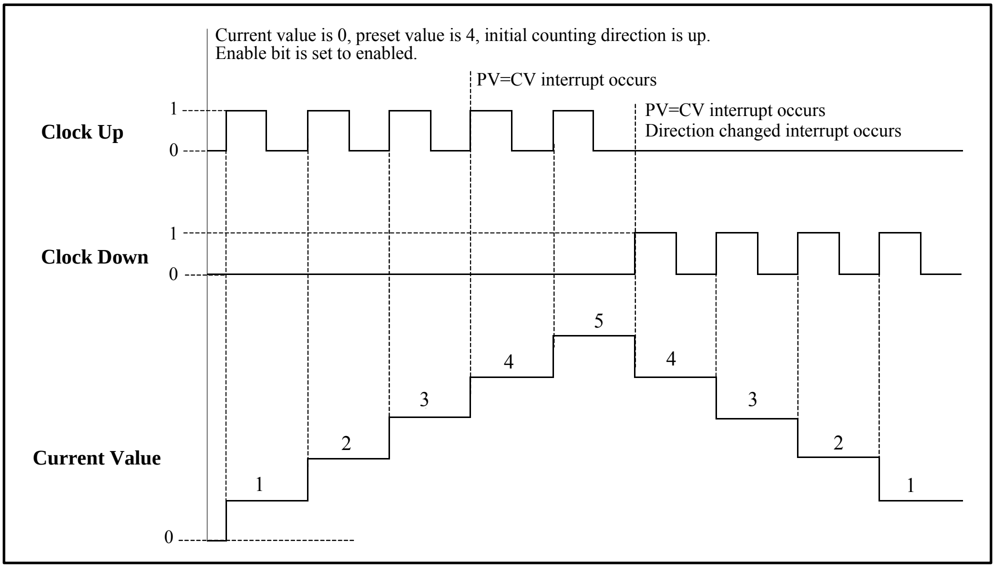

Current value is 0, preset value is 4, initial counting direction is up.
Enable bit is set to enabled.
PV=CV interrupt occurs
PV=CV interrupt occurs
Direction changed interrupt occurs
Clock Up
1
0
Clock Down
1
0
Current Value
0
1
2
3
4
5
4
3
2
1

Figure 6-6 Time Sequence of Mode 6, 7 or 8

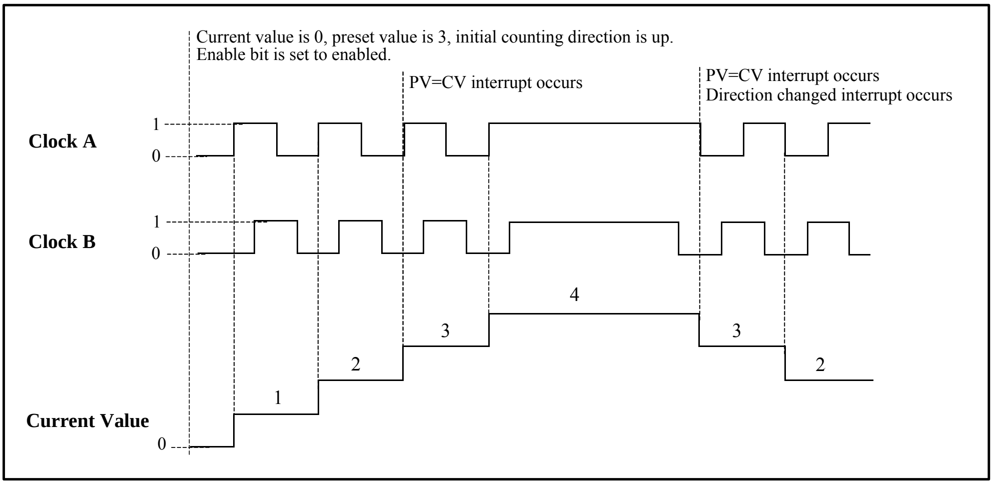

Current value is 0, preset value is 3, initial counting direction is up.
Enable bit is set to enabled.
PV=CV interrupt occurs
PV=CV interrupt occurs
Direction changed interrupt occurs
Clock A
1
0
Clock B
1
0
Current Value
0
1
2
3
4
3
2

Figure 6-7 Time Sequence of Mode 9, 10 or 11 (Quadrature, 1x rate)

6.13.3.4 Configuring the Control Byte

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.

HSC0	HSC1	HSC2	HSC3	HSC4	HSC5	Description
SM37.3	SM47.3	SM57.3	SM137.3	SM147.3	SM157.3	Counting direction: 0 = Up; 1 = Down
SM37.4	SM47.4	SM57.4	SM137.4	SM147.4	SM157.4	Write the counting direction to the HSC: 0 = No; 1 = Yes
SM37.5	SM47.5	SM57.5	SM137.5	SM147.5	SM157.5	Write the new preset value to the HSC: 0 = No; 1 = Yes
SM37.6	SM47.6	SM57.6	SM137.6	SM147.6	SM157.6	Write the new current value to the HSC: 0 = No; 1 = Yes
SM37.7	SM47.7	SM57.7	SM137.7	SM147.7	SM157.7	Enable the HSC: 0 = Disable; 1 = Enable

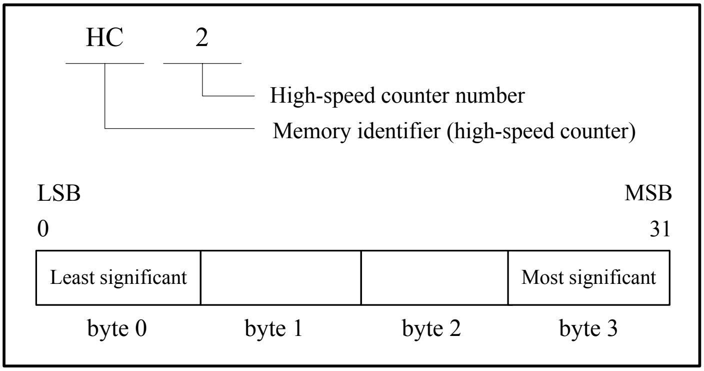
- **Configuring Current Value and Preset Value**

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.



Figue 6-8 Accessing the Current Value of a High-Speed Counter

➢ 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-K3 for details.

6.13.3.5 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	HSC2	HSC3	HSC4	HSC5	Description
SM36.0	SM46.0	SM56.0	SM136.0	SM146.0	SM156.0	Reserved
SM36.1	SM46.1	SM56.1	SM136.1	SM146.1	SM156.1	Reserved
SM36.2	SM46.2	SM56.2	SM136.2	SM146.2	SM156.2	Reserved

SM36.3	SM46.3	SM56.3	SM136.3	SM146.3	SM156.3	Reserved
SM36.4	SM46.4	SM56.4	SM136.4	SM146.4	SM156.4	Reserved
SM36.5	SM46.5	SM56.5	SM136.5	SM146.5	SM156.5	Current counting direction: 0 = Down; 1= Up
SM36.6	SM46.6	SM56.6	SM136.6	SM146.6	SM156.6	Current value equal to preset value: 0 = Not equal; 1 = Equal
SM36.7	SM46.7	SM56.7	SM136.7	SM146.7	SM156.7	Current value greater than preset value: 0 = Not greater than; 1 = Greater than

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 similarsteps.

1) In the initialization subroutine, load the desired control status into SMB37.

   For example (1x counting rate), SMB37 = b#16#FC indicates:

   - Enable HSC0

- Write a new current value to HSC0
- Write a new preset value to HSC0
- Set the direction to be up-counter
- Set the start input and the reset input to be active high

2) Load the desired current value (32-bit) into SMD38. If 0 is loaded, SMD38 is cleared.
3) Load the desired preset value (32-bit) into SMD42.
4) (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.
5) (Optional) Attach the direction-changed event (event 17) to an interrupt routine to respond in real time to a direction-changed event.
6) (Optional) Attach the external reset event (event 16) to an interrupt routine to respond in real time to an external reset event
7) Execute the *HDEF* instruction with the *HSC* input set to be 0 and the *MODE* input set to 9.
8) 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).

1) Load the desired control status into SMB37:

SMB37 = b#16#90: Enable the counter,
Set the new direction to be down-counter

2) 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.

1) Load the desired control status into SMB37:

SMB37 = b#16#C0 Enable the counter

Allow writing the new current value to HSC0.

2) Load the desired current value into SMD38. If 0 is loaded, SMD38 is cleared.

3) 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.

1) Load the desired control status into SMB37:

SMB37 = b#16#A0 Enable the counter

Allow writing the new preset value to HSC0.

2) Load the desired preset value into SMD42.

3) 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.

1) Load the desired control status into SMB37:

SMB37 = b#16#00 Disable the counter;

2) Execute the *HSC* instruction to cause the CPU to disable the counter.

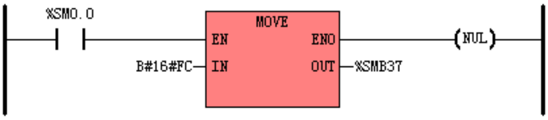
6.13.3.7 Examples

The following example alse uses HSC0.

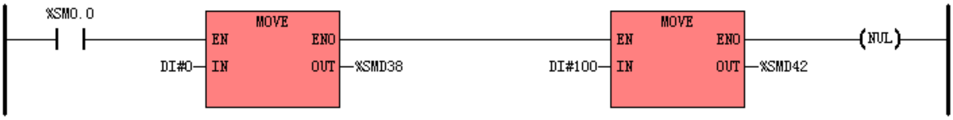
LD

The initialization subroutine: Initialize

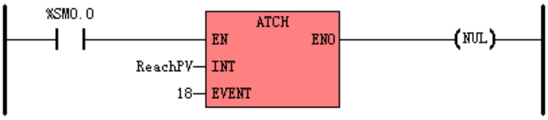
```
(* Network 0 *)
(* 1x counting rate; Enable HSC0; Allow updating current value and preset value;
   Up-counter; Set the start input and the reset input to be active high *)
```



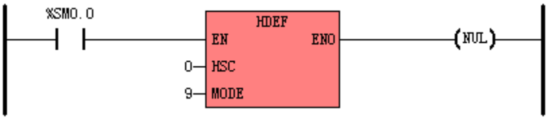
```
(* Network 1 *)
(* Set the new current value and new preset value *)
```



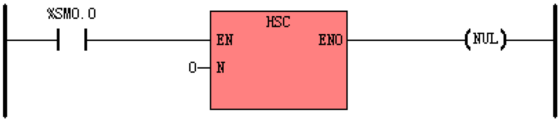
```
(* Network 2 *)
(* Attach the CV = PV event (event 18) to ReachPV interrupt routine *)
```



```
(* Network 3 *)
(* Define HSC0 to be in mode 9 *)
```



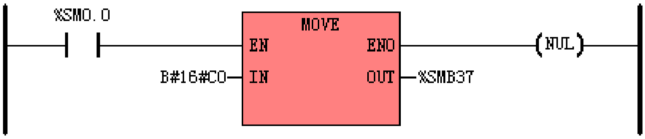
```
(* Network 4 *)
(* Configure and start HSC0 *)
```



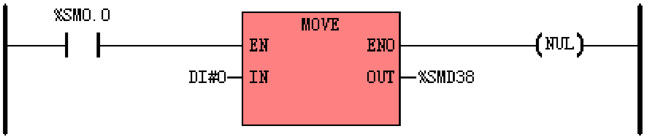
LD

The interrupt routine: ReachPV

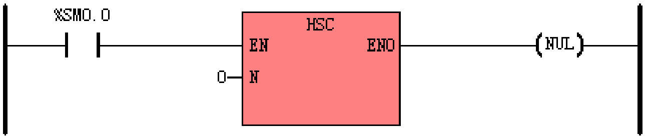
(* Network 0 *)
(* Allow updating current value *)



(* Network 1 *)
(* Set the new current value to be 0 to re-count *)

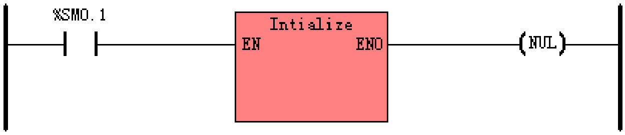


(* Network 2 *)
(* Configure and restart HSC0 *)



Main program:

(* Network 0 *)
(* Call Initialize subroutine *)



IL

The initialization subroutine: Initialize

```
(* Network 0 *)
(* 1x counting rate; Enable HSC0; Allow updating current value AND preset value; *)
(* Up-counter; Set the start input and the reset input to be active high *)
LD          %SM0.0
MOVE        B#16#FC, %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 *)
(*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
```

IL

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
HSC        0
```

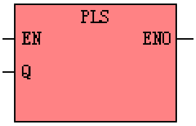
Main program:

```
(* 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).

➢ Description

	Name	Usage	Group	
LD	PLS			☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
IL	PLS	PLS *Q*	U	

Operands	Input/Output	Data Type	Description
Q	Input	INT constant (0 or 1)	Assign pulse output channel: 0 represents output through Q0.0; 1 represents output through Q0.1.

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 High-speed Pulse Train Output Supported by the KINCO-K3

The KINCO-K3 provides two PTO/PWM pulse generators that can be used to output either a high-speed pulse train or a pulse-width modulated wave, and the output pulse frequency can reach 20kHz. Thereof, one generator is assigned to Q0.0, called PWM0 or PTO0; the other is assigned to Q0.1, called PWM1 or PTO1.

The PTO/PWM pulse generators and the output image area share the memory address Q0.0 and Q0.1. When Q0.0

or Q0.1 is used for a PTO or PWM function, the PTO/PWM generator controls the output and prohibits the normal use of this output channel. When the PTO/PWM generator is inactive, the output image area shall take over the control of the output channel.

Some registers are provided in SM area for each PTO/PWM generator: a control byte (8-bit), a cycle time and pulse width value (16-bit unsigned integer), and a pulse count value (32-bit unsigned double integer). Once these memories have been configured according to the desired values, the desired operation can be fulfilled by executing the *PLS* instruction. Default values for all control bits, cycle time, pulse width and pulse count values are 0.

Notice: Make sure not to use the PTO and PWM functions if Q0.0 and Q0.1 are relay-output!

➢ **PWM (Pulse-Width Modulation)**

PWM provides a continuous pulse output with a fixed cycle time and a variable duty cycle, and you can control the cycle time and the pulse width.

The cycle time and the pulse width time can be specified in either microsecond or millisecond increments. The cycle time range is 50~65535µs or 2~65535ms. The pulse width time range is 0~65535µs or 0~65535 ms. If the pulse width time is greater than the cycle time value, the duty cycle is set to be 100% automatically and the output is on continuously. If the pulse width time is 0, the duty cycle is set to be 0% and the output is off.

You can use on of the following two methods to update the characteristic of a PWM waveform:

- **Synchronous update**

A synchronous update can be used if time base (µs or ms) needn't change. With a synchronous update, the variation of the waveform characteristics occurs on a cycle boundary, and a smooth transition is provided. The typical PWM operation is to change the pulse width while the cycle time keeps constant, so time base doesn't need to change.

- **Asynchronous Update**

If the time base of the PWM generator has to be changed, an asynchronous update can be used. An asynchronous update may prohibit the PWM function instantaneously and result in asynchrony to the PWM waveform, and this may cause the controlled equipment to vibrate undesirably. Thus, you are recommended to choose a time base suitable for all of your desired cycle time values to use synchronous PWM updates.

The control bit SM67.4 or SM77.4 specifies the update metod used when the PLS instruction is executed to make the changes take effect. In case that the time base is changed, an asynchronous update shall occur regardless of the update conrol bit.

➢ PTO (Pulse Train Output)

PTO provides a square wave (50% duty cycle) output, and you can control the cycle time (in either microsecond or millisecond increments) and the number of the output pulses.

The cycle time range is 50~65535μs or 2~65535ms. In case the cycle time is specified as an odd number (such as 35 ms), some distortion in the duty cycle may occur. The pulse number range is 1~4,294,967,295. If the specified pulse number is 0, the pulse count defaults to 1.

PTO can produce a single pulse train. In addition, PTO supports the pipelining of multiple pulse trains using a pulse profile: a new pulse train output will start immediately as soon as the active pulse train output is finished.

• Single-Segment Pipelining

In single-segment pipelining, it is necessary to update the relevant SM registers for next pulse train output. Once the initial PTO segment is started, the SM registers must be modified immediately according to the requirement of the second waveform and then re-execute the *PLS* instruction. The configurations of the second pulse train are kept in a pipeline until the first pulse train is complete. In the pipeline, only one PTO segment can be stored at one time; once the first pulse train is complete, the output of the second pulse train starts immediately, and the pipeline is changed to be available for the next pulse train configuration. Repeat this procedure to configure the next pulse train.

The transition between the trains is smooth except the following conditions: the time base is changed, or the active pulse train has finished but the CPU does not get the new pulse train configurations by the execution of the *PLS* instruction.

- **Multi-Segment Pipelining**

In multi-segment pipelining, the CPU automatically reads the configurations of each pulse train segment from a profile table located in V area.

In this mode, time base shall be stored in SMB67 (corresponding to PTO0) or SMB167 (corresponding to PTO1). The starting V area offset of the profile table is stored in SMW168 (corresponding to PTO0) or SMW178 (corresponding to PTO1). The time base can be in either microsecond or millisecond, it shall be applied to all cycle values in the profile table, and cannot be modified during the profile execution. Execute the *PLS* instruction to start multi-segment operation.

The length of each segment is 8 bytes, including a cycle time value (16-bit, WORD), a cycle time increment value (16-bit, INT), and a pulse count value (32-bit, DWORD).

The following table describes the format of the profile table.

Byte offset[1]	Length	Segment	Description
0	16-bit	-	The number of segments (1 to 64)
1	16-bit	1	Initial cycle time (2 to 65535 times of the time base)
3	16-bit		Cycle time increment for each pulse (-32768 to 32767 times of the time base)
5	32-bit		Pulse count (1 to 4,294,967,295)
9	16-bit	2	Initial cycle time (2 to 65535 times of the time base)
11	16-bit		Cycle time increment for each pulse (-32768 to 32767 times of the time base)
13	32-bit		Pulse count (1 to 4,294,967,295)
…	…	…	…

1 All the offsets in this column are relative to the starting position of the profile table.

Notice: the starting position of the profile table must be an odd address in V area, e.g. VB3001.

The cycle time can be increased or decreased automatically according to the specified cycle time increment value for each pulse. A positive increment value makes the cycle time increase, a negative increment value makes the cycle time decrease, and 0 makes the cycle time remain unchanged.

6.13.4.2 Configuring and Controlling the PTO/PWM Operation

Each PTO/PWM generator is provided with some registers in SM area to store its configurations or indicate its status. The characteristics of a PTO/PWM waveform can be changed by modifying the corresponding SM registers and then executing the *PLS* instruction. The following table decribes control registers detaildly.

Q0.0	**Q0.1**	**Control bits**
SM67.0	SM77.0	(PTO/PWM) Whether to update the cycle time: 0 = not update; 1 = update
SM67.1	SM77.1	(PWM) Whether to update pulse width time: 0 = not update; 1 = update
SM67.2	SM77.2	(PTO) Wheter to update the pulse count: 0 = not update; 1 = update
SM67.3	SM77.3	(PTO/PWM) Time base: 0 = 1μs; 1 = 1ms
SM67.4	SM77.4	(PWM) Update method: 0 = asynchronous update; 1 = synchronous update
SM67.5	SM77.5	(PTO) Single or multiple segment operation: 0 = single; 1 = multiple
SM67.6	SM77.6	Select PTO or PWM mode: 0 = PTO; 1 = PWM
SM67.7	SM77.7	(PTO/PWM) Enable: 0 = disable; 1 = enable
Q0.0	**Q0.1**	**Other registers**
SMW68	SMW78	(PTO/PWM) Cycle time value, Range: 2 to 65535
SMW70	SMW80	(PWM) Pulse width value, Range: 0 to 65535
SMD72	SMD82	(PTO) Pulse count value, Range: 1 to 4,294,967,295
SMB166	SMB176	The number of the segments in progress For multi-segment PTO operation only

SMW168	SMW178	The starting location of the profile table (byte offset from V0) For multi-segment PTO operation only

The following table describes the status bits of the PTO/PWM generators.

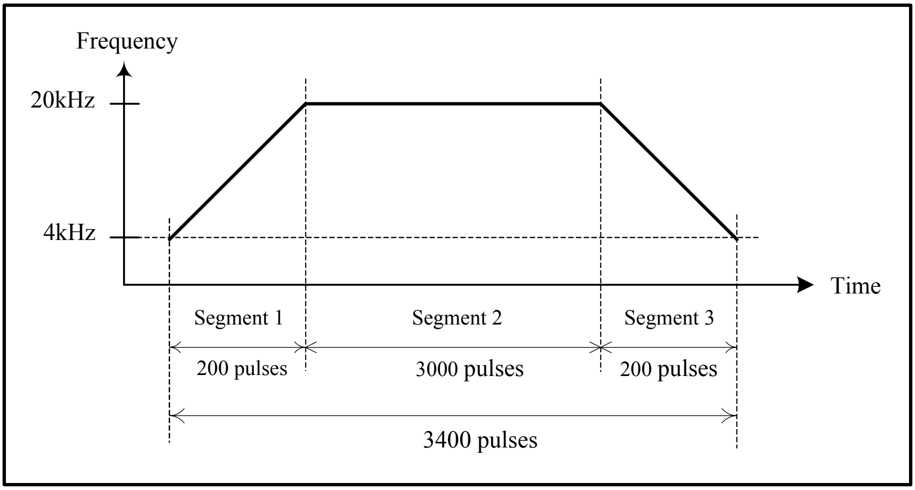
Q0.0	Q0.1	Status Bits
SM66.4	SM76.4	PTO profile terminated due to increment calculation error: 0 = no error; 1 = terminated
SM66.5	SM76.5	PTO profile terminated due to user command: 0 = not terminated; 1 = terminated
SM66.6	SM76.6	PTO pipeline overflow/underflow 0= no; 1 = overflow/underflow
SM66.7	SM76.7	PTO idle 0 = in progress; 1 = iddle

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.

6.13.4.3 Calculating Profile Table

The multi-segment pipelining funciton is helpful for many applications, especially for stepping motor control. For example, you can use multi-segment pipelining funciton to control a stepping motor according to a profile table that includes a simple accelerating, constant-speed, and decelerating sequence, or a more complicated profile table that includes up to 64 segments and each segment corresponds to an accelerating, constant-speed, and decelerating operation.

The following is a specific sample of stepping motor control that illustrates how to calculate the multi-segment profile table values. The profile table includes 3 segments: accelerating the stepping motor (segment 1), operating the motor at a constant speed (segment 2) and then decelerating the motor (segment 3). See diagram 2-13.



Figue 6-9 A Sample Frequency/Time Diagram

For this example: in segment 1, the output frequency accelerates from 4kHz to 20kHz in 200 pulses; in segment 2, the output frequency keeps at 20kHz in 3000 pulses; in segment 3, the output frequency decelerates from 20kHz to 4kHz in 200 pulses. Because the cycle time instead of frequency is used in the profile table, you have to convert the frequency values into the cycle time values. Therefore, the initial and final cycle time is 250µs, and the least cycle time (corresponding to the maximum frequency) is 50µs.

The following formula can be used to calculate the cycle time increment value for a segment:

The cycle time increment value for a segment = (ETsegn - ITseg) / Qseg

Where: ETseg = The final cycle time value for this segment

ITseg = The initial cycle time value for this segment

Qseg = The number of pulses in this segment

Using this formula to calculate the cycle time increment values for the above example:

Segment 1 (acceleration):

Cycle time increment value = -1

Segment 2 (constant speed):

Cycle time increment value = 0

Segment 3 (deceleration):

Cycle time increment value = 1

Assume that the profile table is in the V area, starting at VB701.

The following table lists the generated profile table values.

Byte Offset	Value	Comment	
VB701	3	The number of segments	
VW702	250	Initial cycle time	Segment 1
VW704	-1	Cycle time increment	
VD706	200	The number of pulses	
VW710	50	Initial cycle time	Segment 2
VW712	0	Cycle time increment	
VD714	3000	The number of pulses	
VW718	50	Initial cycle time	Segment 3
VW720	1	Cycle time increment	
VD722	200	The number of pulses	

Smooth transition between the segments is very important, a smooth transition requires such condition that the final cycle time of the previous segment plus the cycle time increment value equals to the initial cycle time of the subsequent segment.

6.13.4.4 PTO Operations

The fallowing takes PTO0 as an example to introduce how to configure and operate the PTO/PWM generator in the user programme.

- **Initializing the PTO (Single-Segment Operation)**

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 PTO0 in the initialization subroutine:

1) Load the desired control status into SMB67:

   For example, SMB67 = B#16#85 indicates

   - Enable the PTO/PWM function
   - Select PTO operation
   - Select 1μs as the time base
   - Allow updating the pulse count value and cycling time value.

2) Load the cycle time value into SMW68.
3) Load the pulse count value to SMD72.
4) (Optional) Attach the PTO0-complete event (event 28) to an interrupt routine to respond in real time to a PTO0-complete event.
5) Execute the *PLS* struction to cause the CPU to configure PTO0 and start it.

➢ **Changing the PTO Cycle Time (Single-Segment Operation)**

Follow these steps to change the PTO cycle time:

1) Load the desired control status into SMB67:

   For example, SMB67 = B#16#81 indicates

   - Enable the PTO/PWM function
   - Select PTO operation
   - Select 1μs as the time base
   - Allow updating the cycle time value.

2) Load the cycle time value into SMW68.
3) Execute the *PLS* struction to cause the CPU to configure PTO0 and start it. After the active PTO in process

is completed, a new PTO waveform with the updated cycle time shall be generated.

➢ Changing the PTO Pulse Count (Single-Segment Operation)

Follow these steps to change the PTO pulse count:

1) Load the desired control status into SMB67:

   For example, SMB67 = B#16#84 indicates

   - Enable the PTO/PWM function
   - Select PTO operation
   - Select 1μs as the time base
   - Allow updating the pulse count value

2) Load the pulse count value into SMD72.
3) Execute the *PLS* struction to cause the CPU to configure PTO0 and start it. After the active PTO in process is completed, a new PTO waveform with the updated pulse count shall be generated.

➢ Changing the PTO Cycle Time and the Pulse Count (Single-Segment Operation)

Follow these steps to change the PTO cycle time value and pulse count value:

1) Load the desired control status into SMB67:

   For example, SMB67 = B#16#85 indicates

   - Enable the PTO/PWM function
   - Select PTO operation
   - Select 1μs as the time base
   - Allow updating the pulse count value and cycle time value.

2) Load the cycle time value into SMW68.
3) Load the pulse count value to SMD72.
4) Execute the *PLS* struction to cause the *CPU* to configure PTO0 and start it. After the active PTO in process is completed, a new PTO waveform with the updated cycle time and pulse count shall be generated.

- **Initializing the PTO (Multiple-Segment Operation)**

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 PTO0 in the initialization subroutine:

1) Load the desired control status into SMB67:

   For example, SMB67 = B#16#A0 indicates

   - Enable the PTO/PWM function
   - Select PTO operation
   - Select multi-segment operation
   - Select 1μs as the time base

2) Load an odd number as the starting position of the profile table into SMW168.
3) Use V area to configure the profile table.
4) (Optional) Attach the PTO0-complete event (event 28) to an interrupt routine to respond in real time to a PTO0-complete event.
5) Execute the *PLS* struction to cause the CPU to configure PTO0 and start it.

6.13.4.5 PWM Operations

The fallowing 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:

1) Load the desired control status into SMB67:

   For example, SMB67 = B#16#D3 indicates

   - Enable the PTO/PWM function
   - Select PWM operation
   - Select 1μs as the time base
   - Allow updating the pulse width value and cycle time value
   - Setlect synchronousv update method

2) Load the cycle time value into SMW68.

3) Load the pulse width value into SMW70.

4) 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.):

1) Load the pulse width value (16-bit) into SMW70.

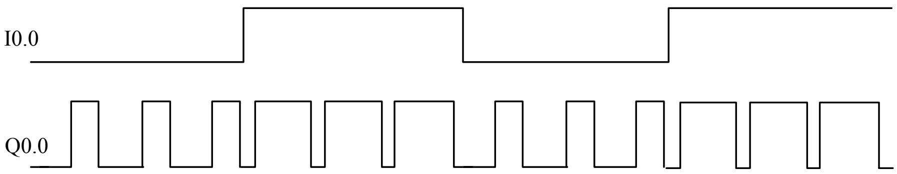
2) Execute the *PLS* struction to cause the CPU to configure PWM0 and start it.

6.13.4.6 Example

➢ **PWM**

PWM1 (output through Q0.1) is used in the example.

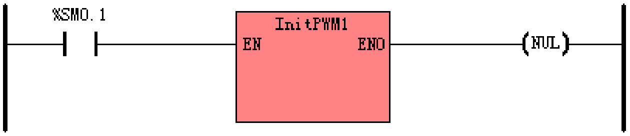
If I0.0 is false, change the pulth width to 40% duty cycle; if I0.0 is true, change the pulth width to 40% duty cycle. The time sequence diagram is shown as follows:



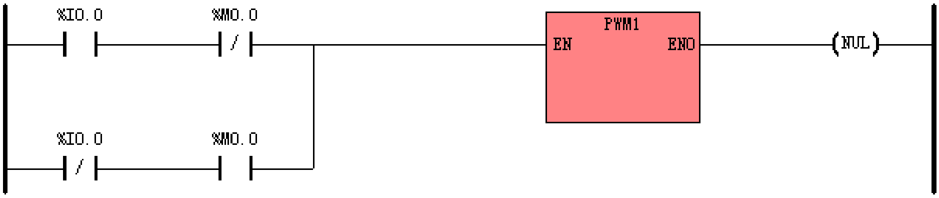
LD

MAIN Program:

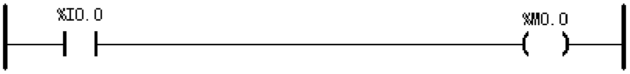
```
(* Network 0 *)
(* Use SM0.1 to call subroutine InitPWM1 to initialize PWM1 *)
```



```
(* Network 1 *)
(* If the status of I0.0 changes,
subroutine PWM1 shall be called to change the pulse width. *)
```

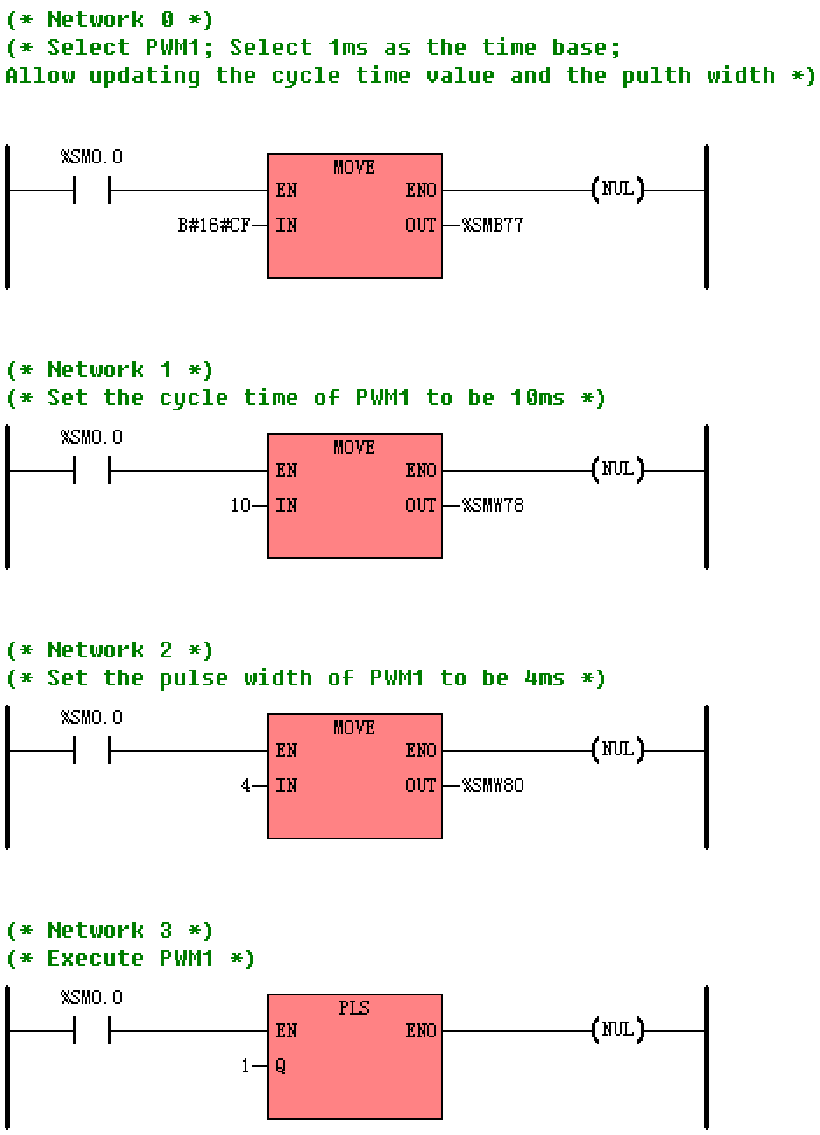


```
(* Network 2 *)
```



LD

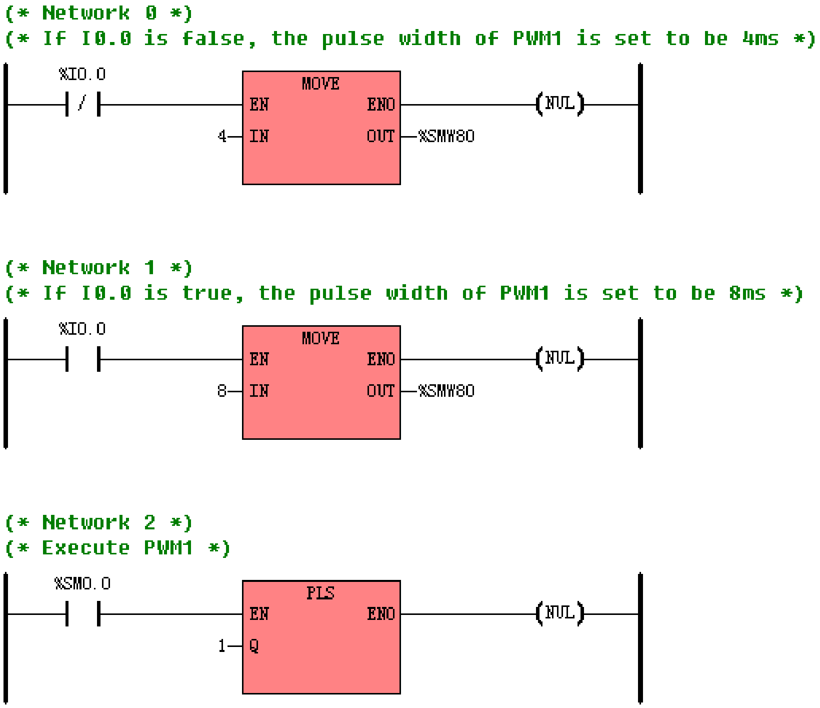
Subroutine *InitPWM1*:



```
(* 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 *)
```

IL

MAIN Program:

```
(* Network 0 *)
(* Use SM0.1 to call subroutine InitPWM1 to initialize PWM1 *)
LD          %SM0.1
CAL         InitPWM1
(* Network 1 *)
(* If the status of I0.0 changes, subroutine PWM1 shall be called to change the pulse width. *)
LD          %I0.0
ANDN        %M0.0
OR(
LDN         %I0.0
AND         %M0.0
)
CAL         PWM1
(* Network 2 *)
LD          %I0.0
ST          %M0.0
```

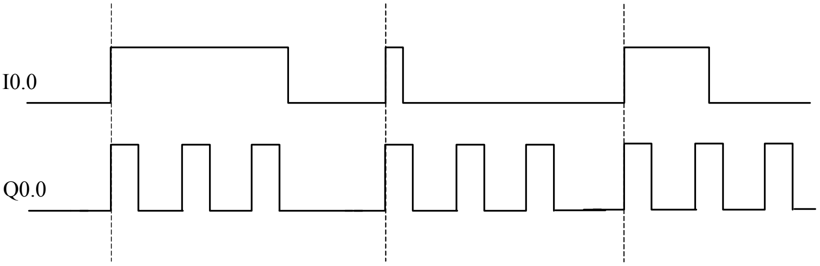
IL	**Subroutin *InitPWM1*:** (* Network 0 *) (*Select PWM1; Select 1ms as the time base; Allow updating the cycle time value and the pulth width*) LD %SM0.0 MOVE B#16#CF, %SMB77 (* Network 1 *) (*Set the cycle time of PWM1 to be 10ms*) LD %SM0.0 MOVE 10, %SMW78 (* Network 2 *) (*Set the pulse width of PWM1 to be 4ms*) LD %SM0.0 MOVE 4, %SMW80 (* Network 3 *) (*Execute PWM1*) LD %SM0.0 PLS 1
	Subroutin *PWM1*: (* Network 0 *) (*If I0.0 is false, the pulse width of PWM1 is set to be 4ms*) LDN %I0.0 MOVE 4, %SMW80 (* Network 1 *) (*If I0.0 is true, the pulse width of PWM1 is set to be 8ms*) LD %I0.0 MOVE 8, %SMW80 (* Network 2 *) (*Execute PWM1*) LD %SM0.0 PLS 1

➢ 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.

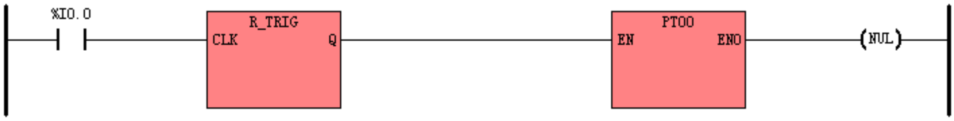
The time sequence diagram is shown as follows:



LD

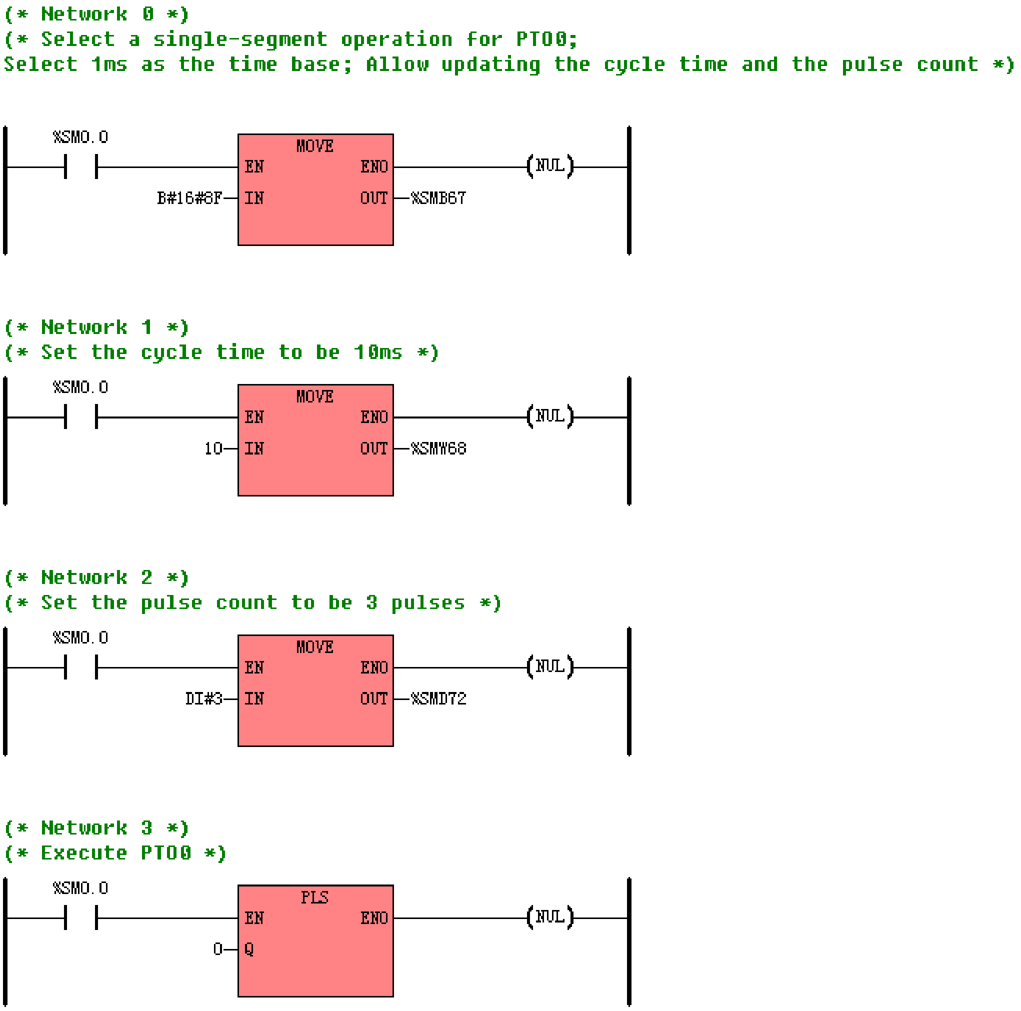
MAIN Program:

```
(* Network 0 *)
(* Start PTO0 on the rising edge of I0.0 *)
```



LD

Subprogram *PTO0*:



```
(* Network 0 *)
(* Select a single-segment operation for PTO0;
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 PTO0 *)
```

IL

MAIN Pogram:

```
(* Network 0 *)
(* Start PTO0 on the rising edge of I0.0 *)
LD          %I0.0
R_TRIG
CAL         PTO0
```

Subprogram PTO0:

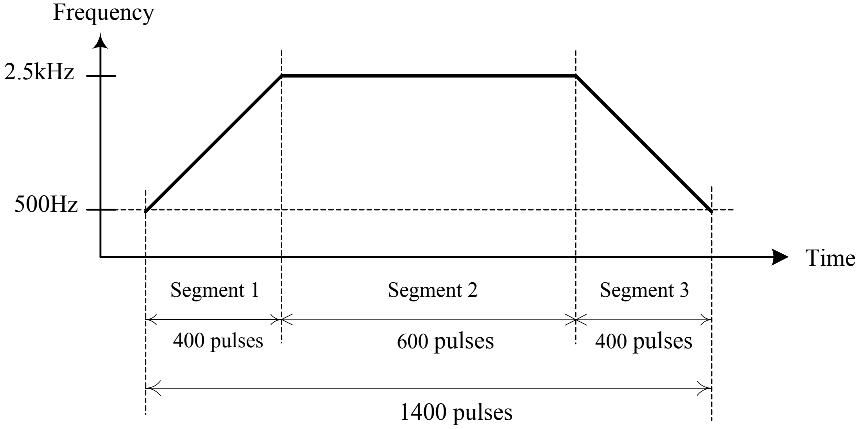
```
(* Network 0 *)
(* Select a single-segment operation for PTO0; *)
(* Select 1ms as the time base; Allow updating the cycle time and the pulse count *)
LD          %SM0.0
MOVE        B#16#8F, %SMB67
(* Network 1 *)
(* Set the cycle time to be 10ms *)
LD          %SM0.0
MOVE        10, %SMW68
(* Network 2 *)
(* Set the pulse count to be 3 pulses *)
LD          %SM0.0
MOVE        DI#3, %SMD72
(* Network 3 *)
(* Execute PTO0 *)
LD          %SM0.0
PLS         0
```

➢ PTO operation (Multi-Segment)

PTO0 (output through Q0.0) is used in the example.

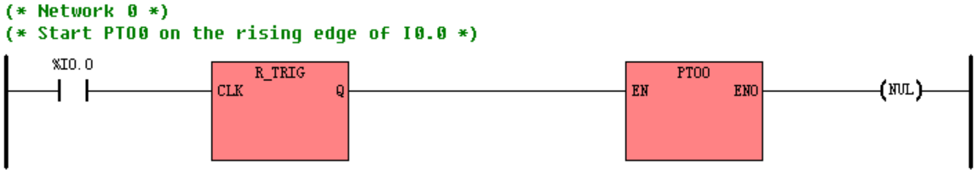
Start PTO0 on the rising edge of I0.0.

Calculate the multi-segment profile table values according to the following chart.



LD

MAIN Program:

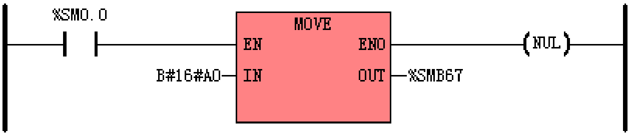


```
(* Network 0 *)
(* Start PTO0 on the rising edge of I0.0 *)
```

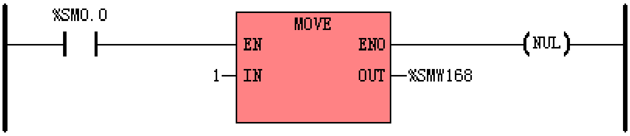
LD

Subroutine *PTO0*:

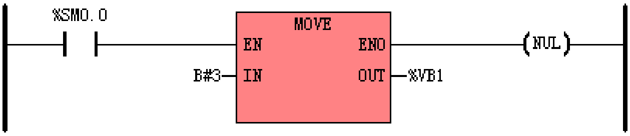
(* Network 0 *)
(* Enable PTO0; Select multi-segment operation; Set the time base to be 1us *)



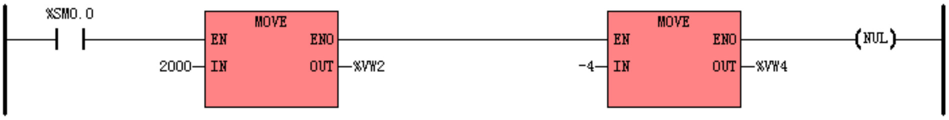
(* Network 1 *)
(* Use VB1 as the staring position of the profile table *)



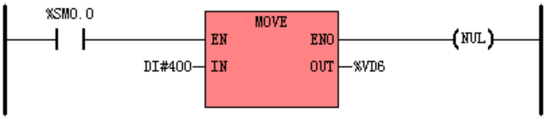
(* Network 2 *)
(* Set the number of segments to be 3 *)

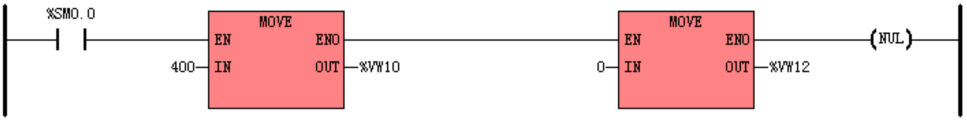
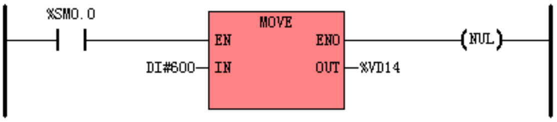
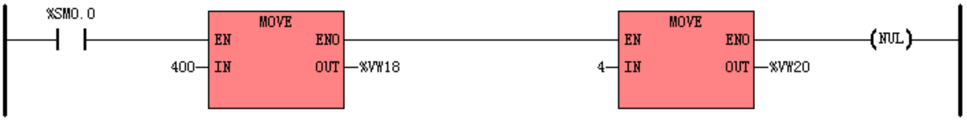
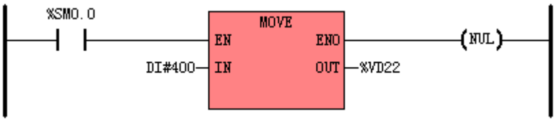
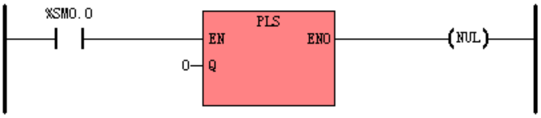


(* Network 3 *)
(* Segment 1: Set the initial cycle time to 2000us, set the cycle time increment to -4us *)



(* Network 4 *)
(* Segment 1: Set the number of pulses to 400 *)



LD	**Subroutine *PTO0*: (Continued)** (* Network 5 *) (* Segment 2: Set the initial cycle time to 400us, set the cycle time increment to 0 *)  %SM0.0 MOVE EN ENO 400 IN OUT %VW10 MOVE EN ENO 0 IN OUT %VW12 NUL (* Network 6 *) (* Segment 2: Set the number of pulses to 600 *)  %SM0.0 MOVE EN ENO DI#600 IN OUT %VD14 NUL (* Network 7 *) (* Segment 3: Set the initial cycle time to 400us, set the cycle time increment to 4us *)  %SM0.0 MOVE EN ENO 400 IN OUT %VW18 MOVE EN ENO 4 IN OUT %VW20 NUL (* Network 8 *) (* Segment 3: Set the number of pulses to 400 *)  %SM0.0 MOVE EN ENO DI#400 IN OUT %VD22 NUL (* Network 9 *) (* Execut PTO0 *) 

IL

MAIN Program:

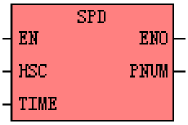
```
(* Network 0 *)
LD        %I0.0
R_TRIG
CAL       PTO0                        (* Start PTO0 on the rising edge of I0.0 *)
```

Subroutine PTO0:

```
(* NETWORK 0 *)
LD        %SM0.0
MOVE      B#16#A0, %SMB67     (* Enable PTO0; Select multi-segment operation; Set the time base to be 1us *)
MOVE      1, %SMW168          (* Use VB1 as the staring position of the profile table *)
MOVE      B#16#03, %VB1       (* Set the number of segments to be 3 *)

(* Segment 1 *)
MOVE      2000, %VW2          (* Set the initial cycle time to 2000us *)
MOVE      -4, %VW4            (* Set the cycle time increment to -4us *)
MOVE      DI#400, %VD6        (* Set the number of pulses to 400 *)

(* Segment 2 *)
MOVE      400, %VW10          (* Set the initial cycle time to 400us *)
MOVE      0, %VW12            (* Set the cycle time increment to 0 *)
MOVE      DI#600, %VD14       (* Set the number of pulses to 600 *)

(* Segment 3 *)
MOVE      400, %VW18          (* Set the initial cycle time to 400us *)
MOVE      4, %VW20            (* Set the cycle time increment to 4us *)
MOVE      DI#400, %VD22       (* Set the number of pulses to 400 *)

PLS       0                   (* Execute PTO0 *)
```

6.13.5 SPD (Speed detection)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	SPD			
IL	SPD	SPD *HSC, TIME, PNUM*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
HSC	Input	INT	Constant（0-5, the number of a HSC）
TIME	Input	WORD	I, Q, M, V, L, SM, Constant
PNUM	Output	DINT	Q, M, V, L, SM

This instruction counts the number of the pulses received at the specified High-speed counter, whos number is *HSC*, in the specified time frame (*TIME*, in ms), and writes the result to the output *PNUM*.

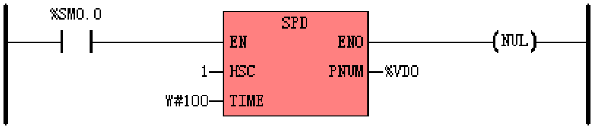
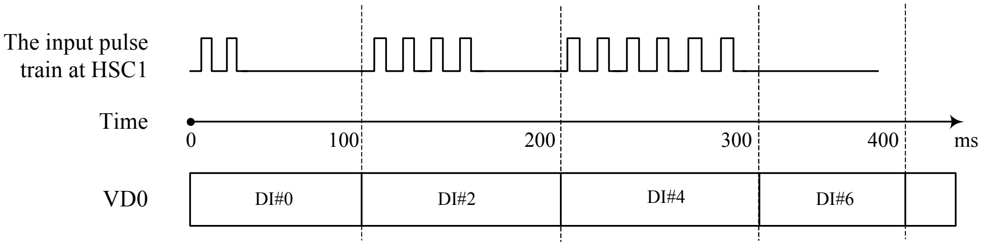
- **LD**

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	 %SM0.0 SPD EN ENO 1 HSC PNUM %VD0 W#100 TIME NUL	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.
IL	LD %SM0.0 SPD 1, W#100, %VD0	
Result	The result is as the following:  The input pulse train at HSC1 Time 0 100 200 300 400 ms VD0 DI#0 DI#2 DI#4 DI#6	

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 2.6.4 Function Block and Function Block Instance for more detailed information.

6.14.1 The resolution of the timer

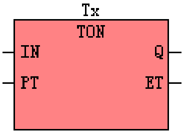
Theer are three resolutions for timers. The timer number determines the resolution as shown in the table.

	CPU304	**CPU306**
Resolution	T0 --- T3: 1ms T4 --- T19: 10ms T20 --- T63: 100ms	T0 --- T3: 1ms T4 --- T19: 10ms T20 --- T127: 100ms
Max timing	32767* Resolution	32767* Resolution

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.

6.14.2 TON (On-delay Timer)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	TON			
IL	TON	TON *Tx*, *PT*	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
Tx	-	Timer instance	T
IN	Input	BOOL	Power flow
PT	Input	INT	I, AI, AQ, M, V, L, SM, constant
Q	Output	BOOL	Power flow
ET	Output	INT	Q, M, V, L, SM, AQ

Tx is an instance of TON fuction block.

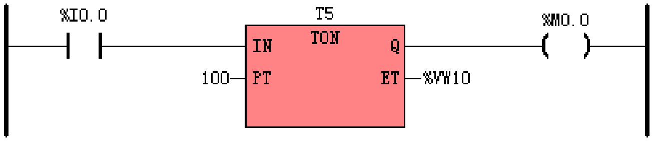
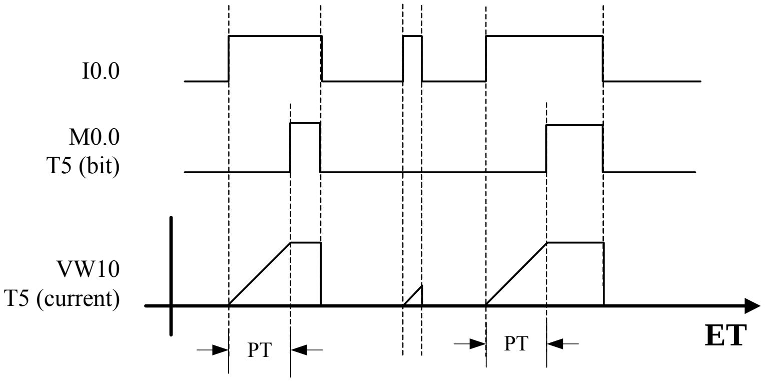
- **LD**

Tx 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 *Tx* are set to be TRUE. If the *IN* input turns to FALSE, *Tx* 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**

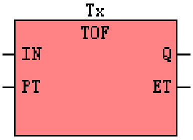
Tx starts to time on the rising edge of CR. When the current value is greater than or equal to the preset value *PT*, the status bit of *Tx* is set to be TRUE. If CR turns to FALSE, *Tx* is reset, and its status bit is set to be FALSE, meanwhile its current value is cleared to 0. After each scan, CR is set to be the status bit value of *Tx*.

- Examples

LD	IL
(* Network 0 *) (* T5 is an instance of TON, and its preset time is 1000ms (100*10ms) *)  %I0.0 T5 TON IN Q 100 PT ET %VW10 %M0.0	(* NETWORK 0 *) LD %I0.0 TON T5, 100 ST %M0.0
Time Sequence Diagram	
 I0.0 M0.0 T5 (bit) VW10 T5 (current) PT PT ET	

6.14.3 TOF (Off-delay Timer)

➢ Description

	Name	Usage	Group	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	TOF			
IL	TOF	TOF *Tx*, *PT*	P	

Operands	Input/Output	Data Type	Acceptable Memory Areas
Tx	-	Timer instance	T
IN	Input	BOOL	Power flow
PT	Input	INT	I, AI, AQ, M, V, L, SM, constant
Q	Output	BOOL	Power flow
ET	Output	INT	Q, M, V, L, SM, AQ

Tx is an instance of TOF fuction block.

- **LD**

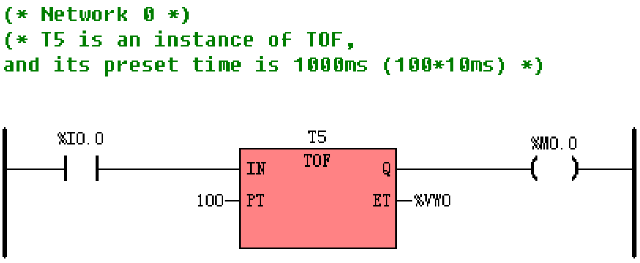
Tx starts to time on the falling edge of the *IN* input. When the elapsed time (i.e. the current value) *ET* is greater than or equal to the preset time *PT*, both the *Q* output and the status bit of *Tx* are set to be FALSE. If the *IN* input turns to TRUE, *Tx* is reset, and both the *Q* output and it status bit are set to be TRUE, meanwhile its current value is cleared to 0.

- **IL**

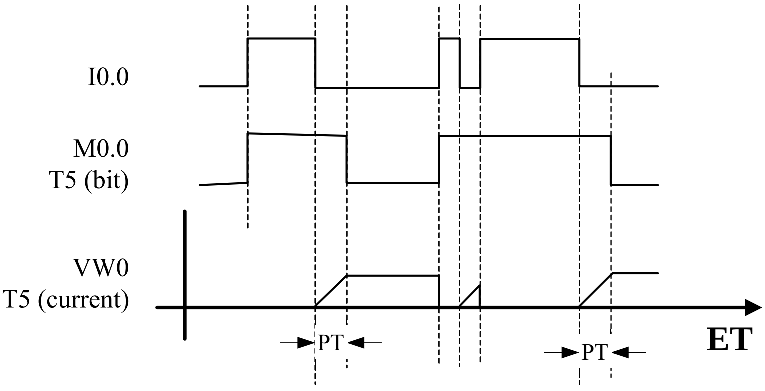
Tx starts to time on the falling edge of CR. When the current value is greater than or equal to the preset value

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

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>(* Network 0 *)
(* T5 is an instance of TOF,
and its preset time is 1000ms (100*10ms) *)

%I0.0
T5
TOF
IN
Q
100
PT
ET
%VW0
%M0.0

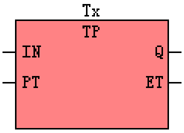
</td><td>(* NETWORK 0 *)
LD %I0.0
TOF T5, 100
ST %M0.0</td></tr>
<tr><th colspan="2">Time Sequence Diagram</th></tr>
<tr><td colspan="2">

I0.0
M0.0
T5 (bit)
VW0
T5 (current)
PT
PT
ET

</td></tr>
</table>

6.14.4 TP (Pulse Timer)

➢ Description

	Name	**Usage**	**Group**	☑ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308
LD	TP			
IL	TP	TP *Tx*, *PT*	P	

Operands	**Input/Output**	**Data Type**	**Acceptable Memory Areas**
Tx	-	Timer instance	T
IN	Input	BOOL	Power flow
PT	Input	INT	I, AI, AQ, M, V, L, SM, constant
Q	Output	BOOL	Power flow
ET	Output	INT	Q, M, V, L, SM, AQ

Tx is an instance of TP fuction block. The *TP* instruction is used to generate a pulse for the preset time.

- **LD**

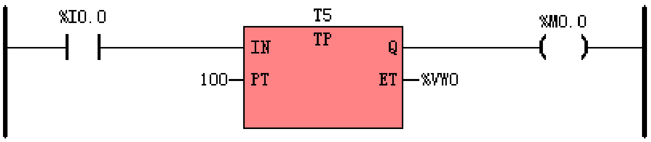
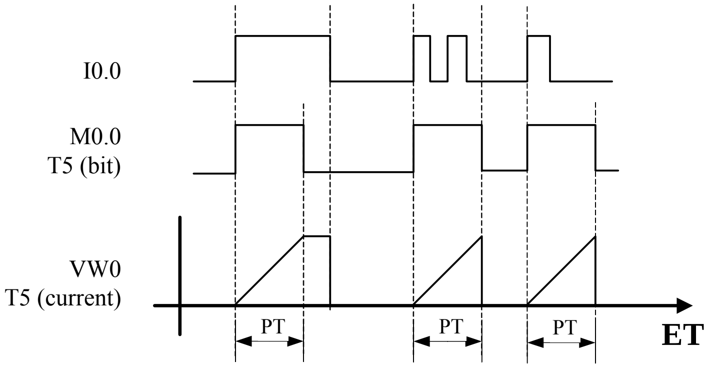
On the rising edge of the *IN* input, *Tx* starts to time, and both the *Q* output and the status bit of *Tx* are set to be TRUE. The *Q* output and the status bit remain TRUE within the preset time *PT*. As soon as the elapsed time (i.e. the current value) *ET* reaches the *PT*, both the *Q* output 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

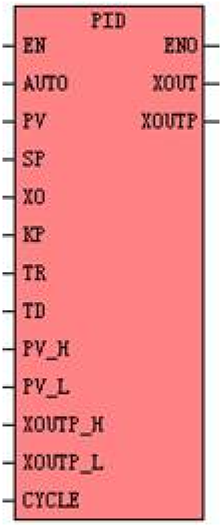
<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>(* Network 0 *)
(* T5 is an instance of TP,
and its preset time is 1000ms (100*10ms) *)
 </td><td>(* NETWORK 0 *)
LD %I0.0
TP T5, 100
ST %M0.0</td></tr>
<tr><th colspan="2">Time Sequence Diagram</th></tr>
<tr><td colspan="2"> </td></tr>
</table>

6.15 PID

PID instruction is provided in Kinco-K3, 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.

6.15.1 PID

- **Description**

	Name	Usage	Group	
LD	PID	 PID EN AUTO PV SP XO KP TR TD PV_H PV_L XOUTP_H XOUTP_L CYCLE ENO XOUT XOUTP		☐ CPU304 ☑ CPU304EX ☑ CPU306 ☑ CPU306EX ☑ CPU308 ☑ CPU406 ☑ CPU408
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
AUTO	INPUT	BOOL	I,Q,V,M,SM,L,T,C	Manual/Auto. 0=Manual, 1=Auto.
PV	INPUT	INT	AI,V,M,L	Process Variable
SP	INPUT	INT	V,M,L	Setpoint
XO	INPUT	REAL	V,L	Manual value, range [0.0, 1.0]
KP	INPUT	REAL	V,L	Proportionality constant
TR	INPUT	REAL	V,L	Reset time, which determines the time response of the integrator. (Unit: s)
TD	INPUT	REAL	V,L	Derivative time, which determines the time response of the derivative unit. (Unit: s)
PV_H	INPUT	INT	V,L	The upper limit value of PV
PV_L	INPUT	INT	V,L	The lower limit value of PV
XOUTP_H	INPUT	INT	V,L	The upper limit value of XOUTP
XOUTP_L	INPUT	INT	V,L	The lower limit value of XOUTP
CYCLE	INPUT	DINT	V,M,L	Sampling period. (Unit: ms)
XOUT	OUTPUT	REAL:	V,L	Manipulated Value, range [0.0, 1.0].
XOUTP	OUTPUT	INT	AQ,V,M,L	Manipulated Value Peripheral. This value is the normalizing result of XOUT.

² **LD**

If EN is 1, this instruction is executed.

² **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-K3 automatically finishs 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. The normalization is as following:

The normalization value of PV = k*PV + b

The normalization value of SP = k*SP + b

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 measurend type is '4-20mA' whose the measured value is '4000-20000'. Now, you can assign the following values to the PID inputs:

<1cmn lang="EN-US">	**Actual Parameter**	**Comment**
PV	AIW0	AIW0 can be set as PV because of their linear relation.
SP	14000	14mA. Because 14mA means the real pressure value 25MPa.
PV_L	4000	The lower limit value of the transformer's output
PV_H	20000	The upper limit value of the transformer's output

Manipulated Values

This PID has two manipulated values: XOUT and XOUTP.

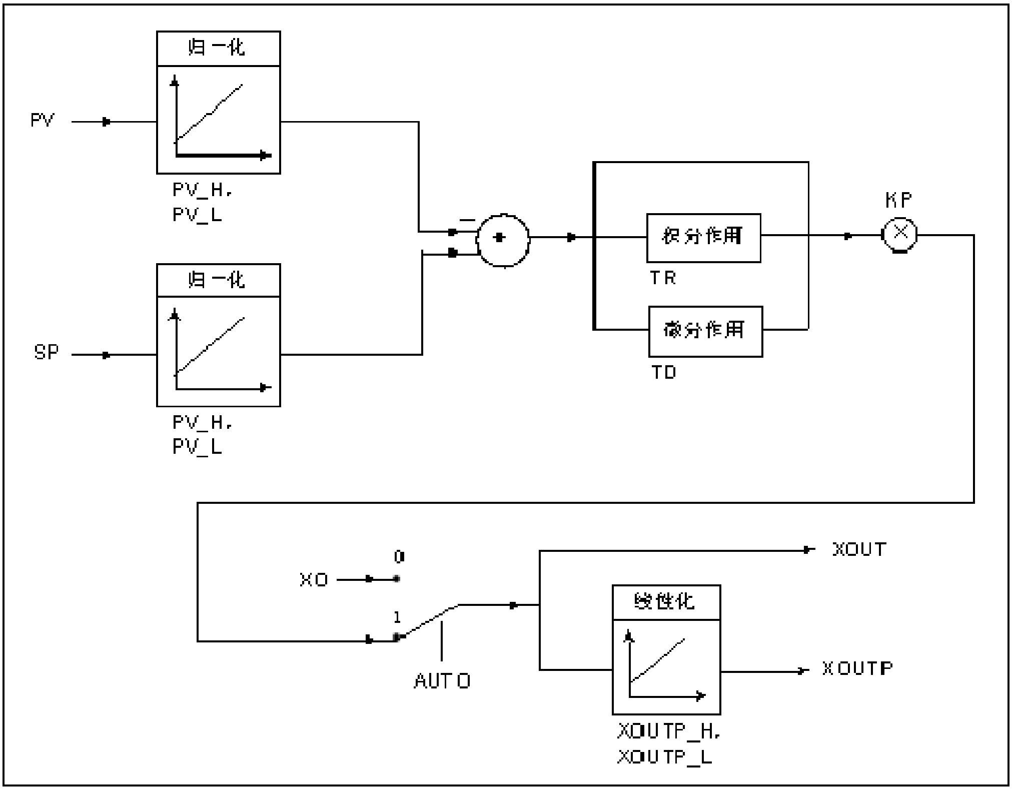
XOUT is a value between 0.0 and 1.0 (that is between 0.0 and 100.0%).

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


归一化
PV
PV_H,
PV_L
归一化
SP
PV_H,
PV_L
—
积分作用
TR
微分作用
TD
KP
XO
0
1
AUTO
XOUT
线性化
XOUTP
XOUTP_H,
XOUTP_L

Example

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

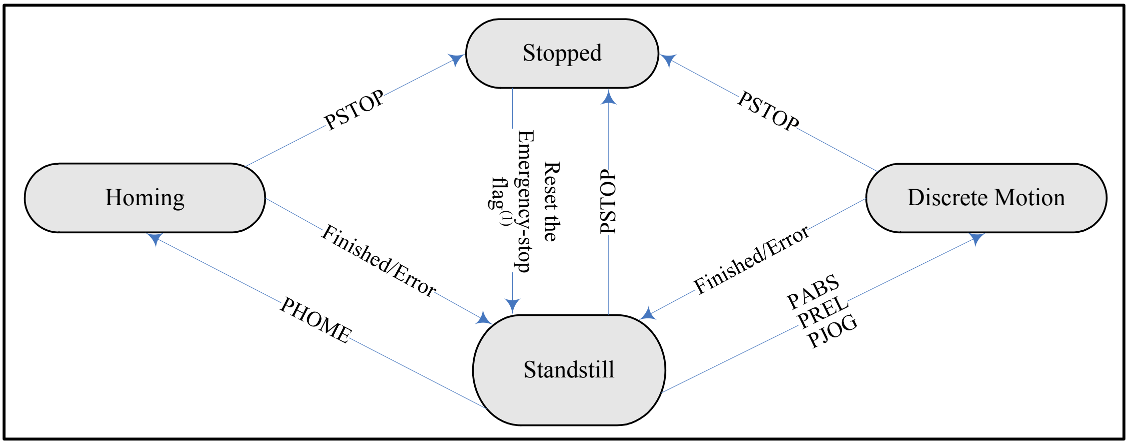
The KINCO-K3 provides 2 high-speed pulse output channels: Q0.0 and Q0.1， and can be used for position control for 2 axes. In 6.13.3 High-speed Pulse Output Instructions, the usage of PTO/PWM and the PLS instruction is described detailedly.

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 can reach 20kHz maximumly.

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 positon 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.



(1) 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-K3 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-K3 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 conrol 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

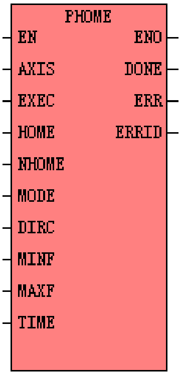
6.16.2.3 The error identification

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 *AXIS* is not 0 or 1.
2	The value of *MINF* is larger than the value of *MAXF.*
3	The value of *MINF* is less than the allowed lowest frequency (20Hz).
4	The value of *TIME* (accelerating / decelerating time) doesn't match the value of *MINF* and *MAXF*.

6.16.3 PHOME (Homing)

➢ Description

	Name	Usage	Group	
LD	PHOME	 PHOME EN AXIS EXEC HOME NHOME MODE DIRC MINF MAXF TIME ENO DONE ERR ERRID		□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
IL	PHOME	PHOME *AXIS*, *EXEC*, *HOME*, *NHOME*, *MODE*, *DIRC*, *MINF*, *MAXF*, *TIME*, *DONE*, *ERR*, *ERRID*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
AXIS	Input	INT	Constant (0 or 1)
EXEC	Input	BOOL	I, Q, V, M, L, SM, RS, SR
HOME	Input	BOOL	I, Q, V, M, L, SM, RS, SR
NHOME	Input	BOOL	I, Q, V, M, L, SM, RS, SR
MODE	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constant
DIRC	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constant
MINF	Input	WORD	I, Q, M, V, L, SM, Constant
MAXF	Input	WORD	I, Q, M, V, L, SM, Constant
TIME	Input	WORD	I, Q, M, V, L, SM, Constant

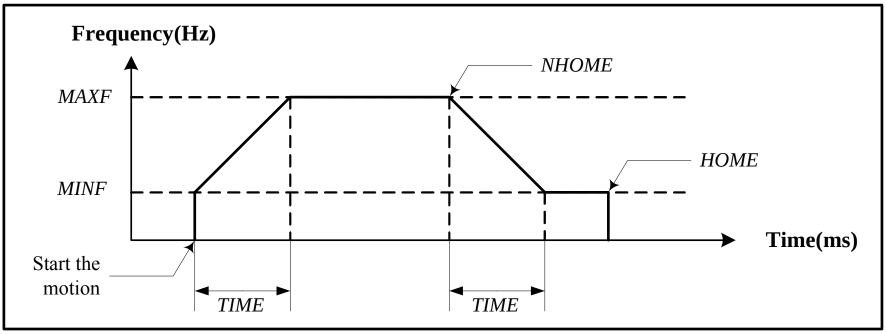
DONE	Output	BOOL	Q, M, V, L, SM
ERR	Output	BOOL	Q, M, V, L, SM
ERRID	Output	BYTE	Q, M, V, L, SM

The following table describes all the operands detailedly.

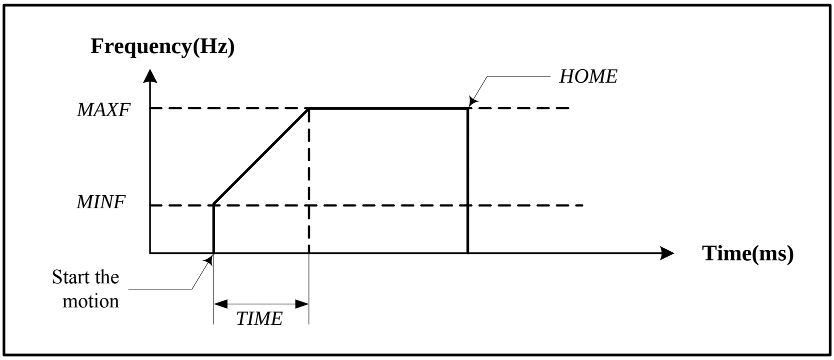
Operands	**Description**
AXIS	The high-speed output channel, 0 means Q0.0, 1 means Q0.1.
EXEC	If *EN* is 1, the *EXEC* starts the 'search home' motion on the rising edge.
HOME	The home signal from the home sensor
NHOME	The near home signal from the near home sensor
MODE	Specifies the homing mode: 0 means that the home signal and the near home signal are all used; 1 means that only the home signal is used.
DIRC	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.
MINF	Specifies the initial speed (i.e., the initial frequency) of the pulse train output. Unit: Hz. Note: the value of *MINF* must be equal to or less than 2KHz.
MAXF	Specifies the highest speed (i.e., the highest frequency) of the pulse train output. Unit: Hz. The available range of *MAXF* is 20Hz ~ 20KHz. *MAXF* must be larger than or equal to *MINF*.
TIME	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 *MINF* to *MAXF*. The deceleration time is the time for the speed decelerating from *MAXF* to *MINF*.
DONE	Indicates that the instruction has finished successfully. 0 = not finished; 1 = finished.
ERR	Indicates that error has occurred during the execution. 0 = no error; 1 = an error has occured.
ERRID	Error identification. If the ERR is 1, the ERRID describes the error's detailed information. Please refer to 6.16.2.3 The 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**

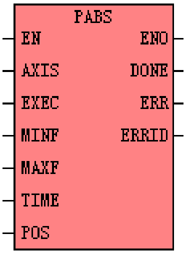
If *EN* is 1, this instruction is executed.

- **IL**

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

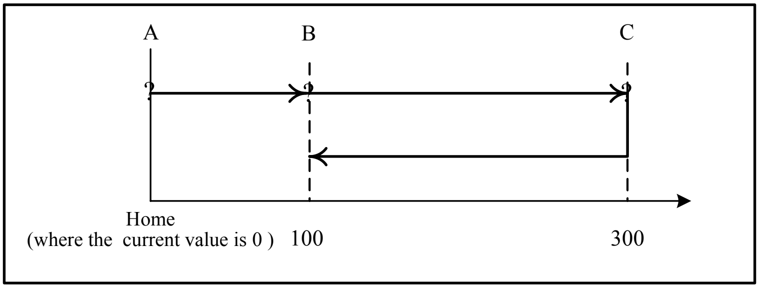
6.16.4 PABS (Moving Absolutely)

➢ Description

	Name	Usage	Group	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	PABS	 PABS EN AXIS EXEC MINF MAXF TIME POS ENO DONE ERR ERRID		
IL	PABS	PABS *AXIS*, *EXEC*, *MINF*, *MAXF*, *TIME*, *POS*, *DONE*, *ERR*, *ERRID*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
AXIS	Input	INT	Constant (0 or 1)
EXEC	Input	BOOL	I, Q, V, M, L, SM, RS, SR
MINF	Input	WORD	I, Q, M, V, L, SM, Constant
MAXF	Input	WORD	I, Q, M, V, L, SM, Constant
TIME	Input	WORD	I, Q, M, V, L, SM, Constant
POS	Input	DINT	I, Q, M, V, L, SM, HC, Constant
DONE	Output	BOOL	Q, M, V, L, SM
ERR	Output	BOOL	Q, M, V, L, SM
ERRID	Output	BYTE	Q, M, V, L, SM

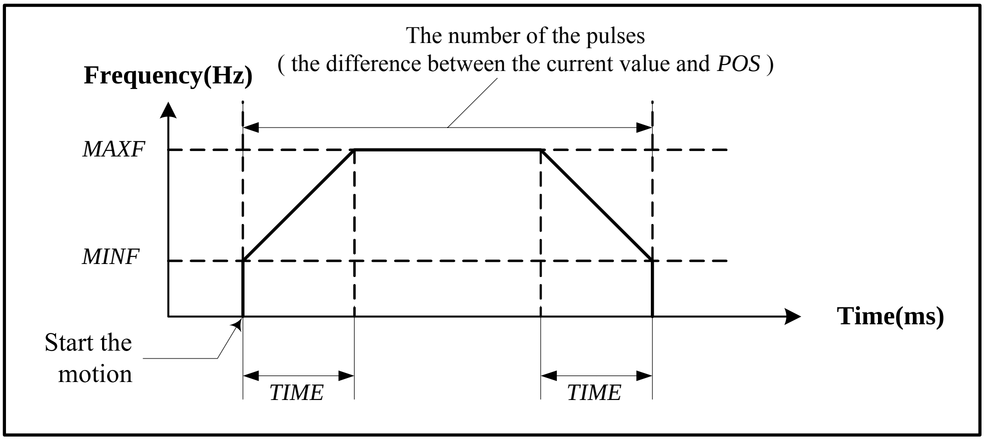
The following table describes all the operands detailedly.

Operands	Description
AXIS	The high-speed output channel, 0 means Q0.0, 1 means Q0.1.
EXEC	If *EN* is 1, the *EXEC* starts the absolute motion on the rising edge.
MINF	Specifies the initial speed (i.e., the initial frequency) of the pulse train output. Unit: Hz. Note: the value of *MINF* must be equal to or less than 2KHz.
MAXF	Specifies the highest speed (i.e., the highest frequency) of the pulse train output. Unit: Hz. The available range of *MAXF* is 20Hz ~ 20KHz. *MAXF* must be larger than or equal to *MINF*.
TIME	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 *MINF* to *MAXF*. The deceleration time is the time for the speed decelerating from *MAXF* to *MINF*.
POS	Specifies the target value. It is represented with the number of pulses between the home positon, where the current value is 0, and the target position. As shown in the following figure, if the object is moved from A to B, the *POS* should be set as '100'; If it is moved from B to C, the *POS* should be set as '300'; If it is moved from C to B, the *POS* should be set as '100'.  A B C Home (where the current value is 0) 100 300
DONE	Indicates that the instruction has finished successfully. 0 = not finished; 1 = finished.
ERR	Indicates that error has occurred during the execution. 0 = no error; 1 = an error has occured.
ERRID	Error identification. If the ERR is 1, the ERRID describes the error's detailed information. Please refer to 6.16.2.3 The error identification .

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**

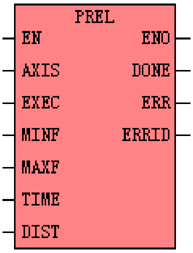
If *EN* is 1, this instruction is executed.

- **IL**

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

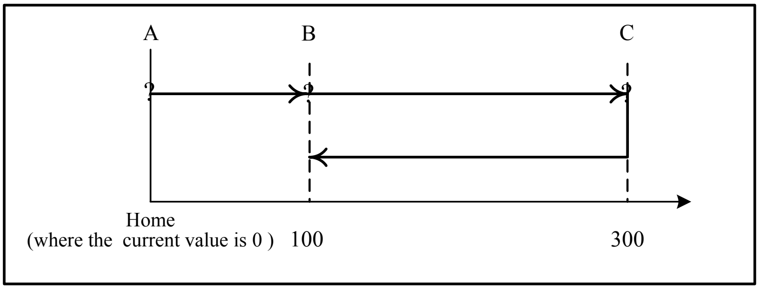
6.16.5 PREL (Moving Relatively)

- Description

	Name	Usage	Group	
LD	PREL	 PREL EN AXIS EXEC MINF MAXF TIME DIST ENO DONE ERR ERRID		☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
IL	PREL	PREL *AXIS*, *EXEC*, *MINF*, *MAXF*, *TIME*, *DIST*, *DONE*, *ERR*, *ERRID*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
AXIS	Input	INT	Constant (0 or 1)
EXEC	Input	BOOL	I, Q, V, M, L, SM, RS, SR
MINF	Input	WORD	I, Q, M, V, L, SM, Constant
MAXF	Input	WORD	I, Q, M, V, L, SM, Constant
TIME	Input	WORD	I, Q, M, V, L, SM, Constant
DIST	Input	DINT	I, Q, M, V, L, SM, HC, Constant
DONE	Output	BOOL	Q, M, V, L, SM
ERR	Output	BOOL	Q, M, V, L, SM
ERRID	Output	BYTE	Q, M, V, L, SM

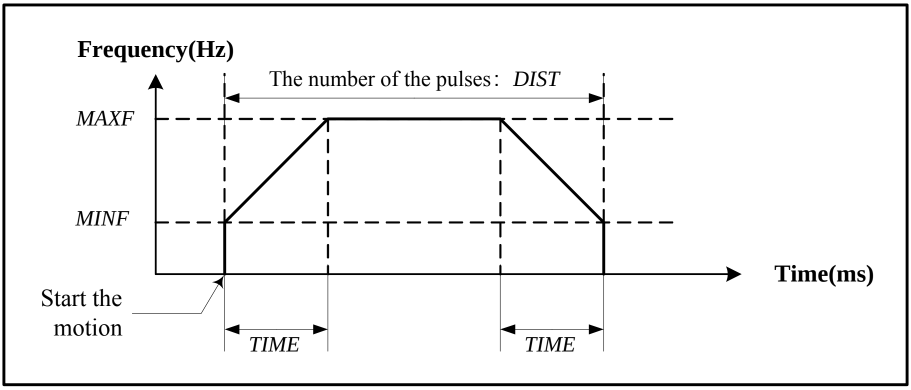
The following table describes all the operands detailedly.

Operands	Description
AXIS	The high-speed output channel, 0 means Q0.0, 1 means Q0.1.
EXEC	If *EN* is 1, the *EXEC* starts the relative motion on the rising edge.
MINF	Specifies the initial speed (i.e., the initial frequency) of the pulse train output. Unit: Hz. Note: the value of *MINF* must be equal to or less than 2KHz.
MAXF	Specifies the highest speed (i.e., the highest frequency) of the pulse train output. Unit: Hz. The available range of *MAXF* is 20Hz ~ 20KHz. *MAXF* must be larger than or equal to *MINF*.
TIME	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 *MINF* to *MAXF*. The deceleration time is the time for the speed decelerating from *MAXF* to *MINF*.
DIST	Specifies the target distance. It is represented with the number of pulses between the current positon and the target position. As shown in the following figure, if the object is moved from A to B, the *DIST* should be set as '100'; If it is moved from B to C, the *DIST* should be set as '200'; If it is moved from C to B, the *DIST* should be set as '-200'.  A B C Home (where the current value is 0) 100 300
DONE	Indicates that the instruction has finished successfully. 0 = not finished; 1 = finished.
ERR	Indicates that error has occurred during the execution. 0 = no error; 1 = an error has occured.
ERRID	Error identification. If the ERR is 1, the ERRID describes the error's detailed information. Please refer to 6.16.2.3 The error identification .

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**

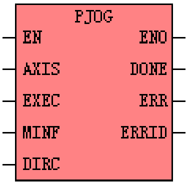
If *EN* is 1, this instruction is executed.

- **IL**

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

6.16.6 PJOG (Jog)

- Description

	Name	Usage	Group	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	PJOG	 PJOG EN ENO AXIS DONE EXEC ERR MINF ERRID DIRC		
IL	PJOG	PJOG *AXIS*, *EXEC*, *MINF*, *DIRC*, *DONE*, *ERR*, *ERRID*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
AXIS	Input	INT	Constant (0 or 1)
EXEC	Input	BOOL	I, Q, V, M, L, SM, RS, SR
MINF	Input	WORD	I, Q, M, V, L, SM, Constant
DIRC	Input	INT	I, Q, M, V, L, SM, AI, AQ, T, C, Constant
DONE	Output	BOOL	Q, M, V, L, SM
ERR	Output	BOOL	Q, M, V, L, SM
ERRID	Output	BYTE	Q, M, V, L, SM

The following table describes all the operands detailedly.

Operands	Description
AXIS	The high-speed output channel, 0 means Q0.0, 1 means Q0.1.
EXEC	If *EN* is 1, the *EXEC* starts the jog motion on the rising edge.
MINF	Specifies the speed (i.e., the initial frequency) of the pulse train output. Unit: Hz.

DIRC	Specifies the the direction of the electric motors: 0 means rotating forwards, and 1 means rotating backwards.
DONE	Indicates that the instruction has finished successfully. 0 = not finished; 1 = finished.
ERR	Indicates that error has occurred during the execution. 0 = no error; 1 = an error has occured.
ERRID	Error identification. If the ERR is 1, the ERRID describes the error's detailed information. Please refer to 6.16.2.3 The 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**

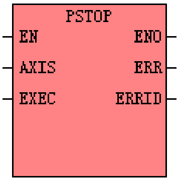
If *EN* is 1, this instruction is executed.

- **IL**

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

6.16.7 PSTOP (Stop)

➢ Description

	Name	Usage	Group	☐ CPU304 ☐ CPU304EX ☐ CPU306 ☑ CPU306EX ☑ CPU308
LD	PSTOP			
IL	PSTOP	PSTOP *AXIS*, *EXEC*, *ERR*, *ERRID*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
AXIS	Input	INT	Constant (0 or 1)
EXEC	Input	BOOL	I, Q, V, M, L, SM, RS, SR
ERR	Output	BOOL	Q, M, V, L, SM
ERRID	Output	BYTE	Q, M, V, L, SM

The following table describes all the operands detailedly.

Operands	Description
AXIS	The high-speed output channel, 0 means Q0.0, 1 means Q0.1.
EXEC	If *EN* is 1, the *EXEC* stops the current motion on the rising edge.
ERR	Indicates that error has occurred during the execution. 0 = no error; 1 = an error has occured.
ERRID	Error identification. If the ERR is 1, the ERRID describes the error's detailed information.

	Please refer to 6.16.2.3 The 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.

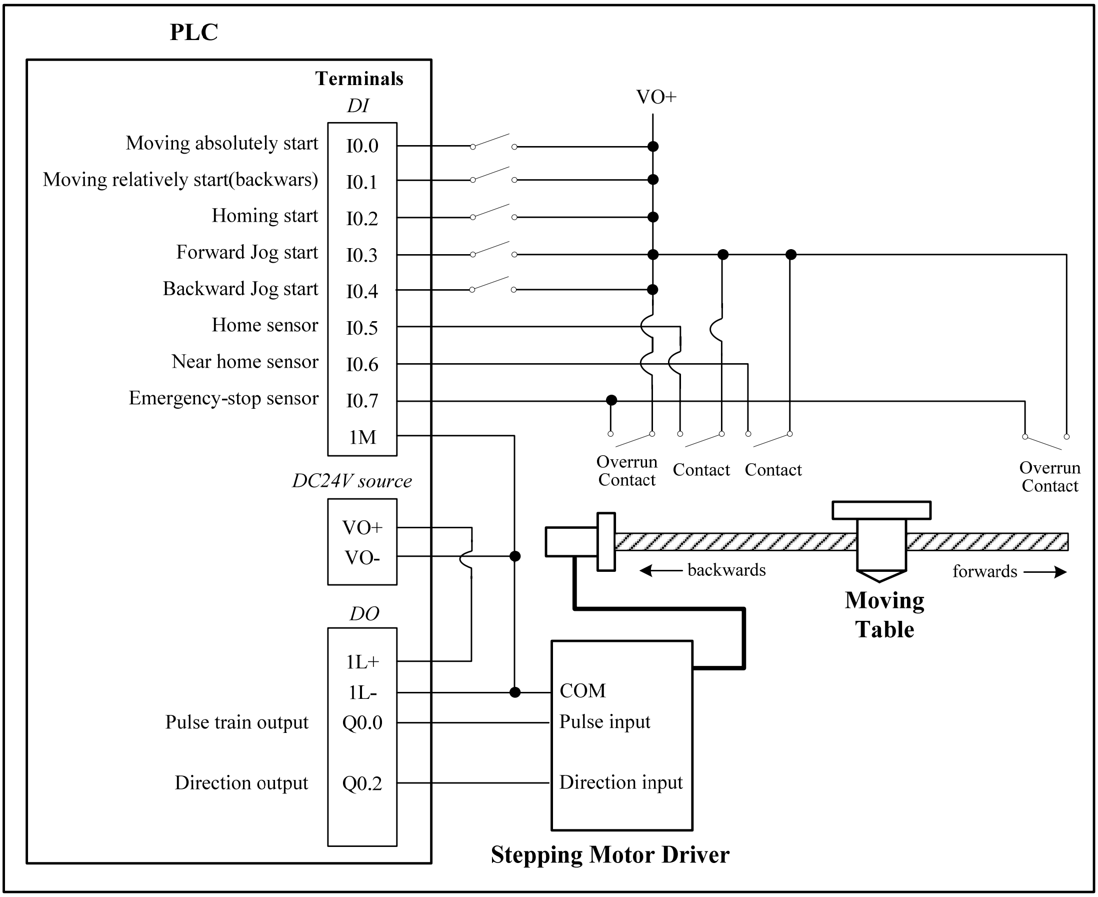
- **IL**

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

6.16.8 Examples

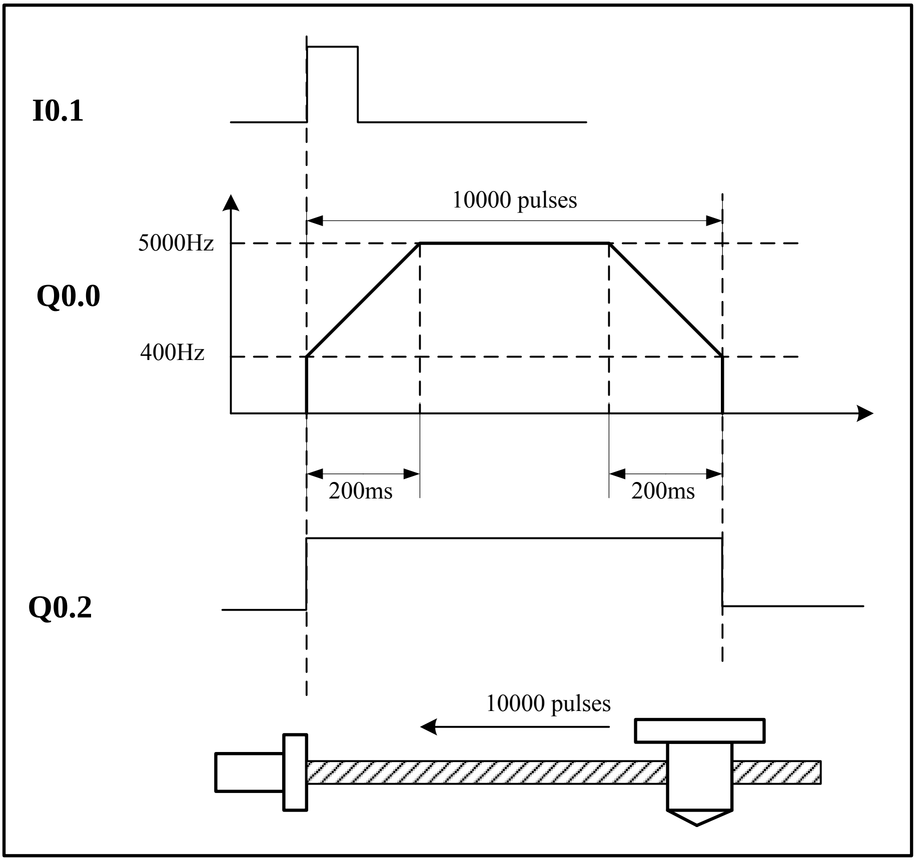
- Wiring

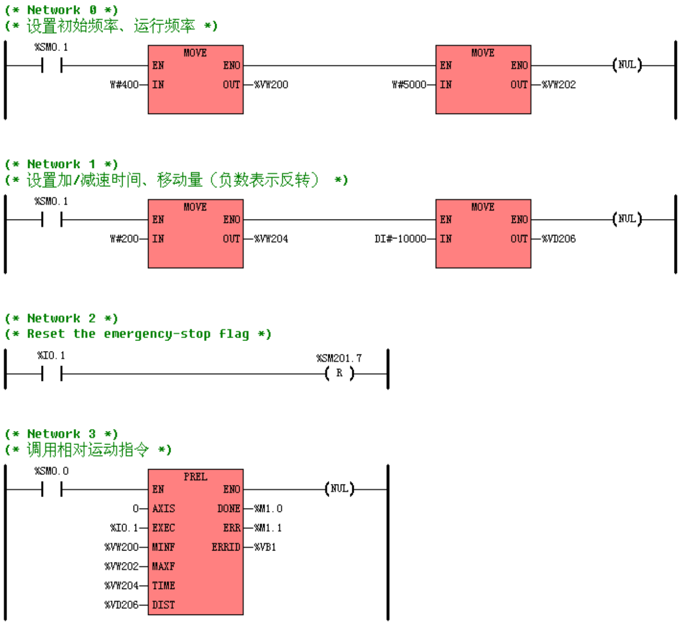
The following system is taken as the example to describe how to use the instructions PREL, PABS, PHOME, PJOG and PSTOP.



- Moving relatively

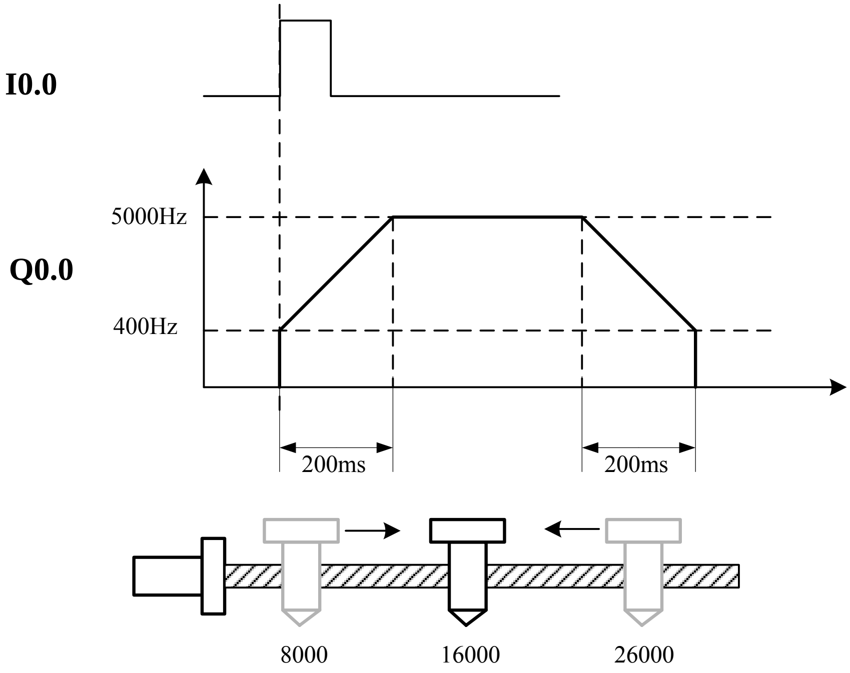
I0.1 is used for starting to move relatively (backwards)。

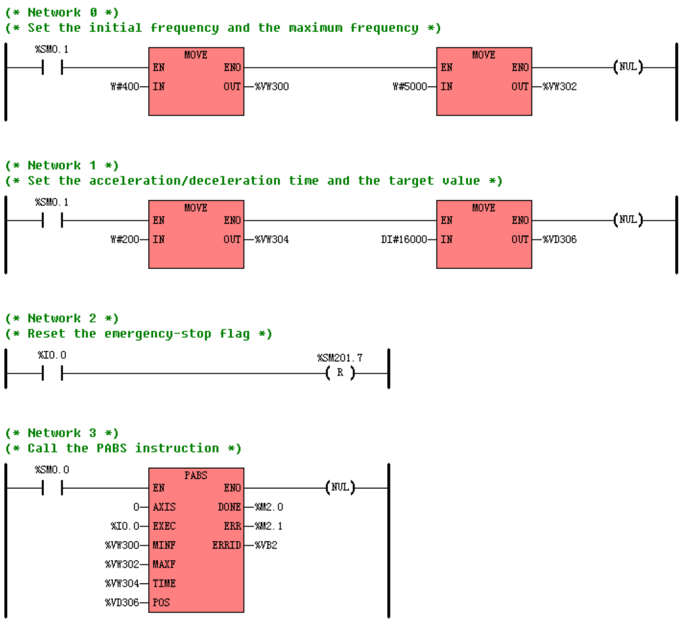
Time Sequence Diagram	 I0.1 10000 pulses 5000Hz Q0.0 400Hz 200ms 200ms Q0.2 10000 pulses

LD	 (* Network 0 *) (* 设置初始频率、运行频率 *) (* Network 1 *) (* 设置加/减速时间、移动量（负数表示反转） *) (* Network 2 *) (* Reset the emergency-stop flag *) (* Network 3 *) (* 调用相对运动指令 *) %SM0.1 MOVE EN ENO W#400 IN OUT %VW200 W#5000 %VW202 NUL W#200 %VW204 DI#-10000 %VD206 %I0.1 %SM201.7 R %SM0.0 PREL 0 AXIS EXEC MINF MAXF TIME DIST DONE ERR ERRID %M1.0 %M1.1 %VB1
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	 I0.0 Q0.0 5000Hz 400Hz 200ms 200ms 8000 16000 26000 The table is required to move to the position ‘16000’.

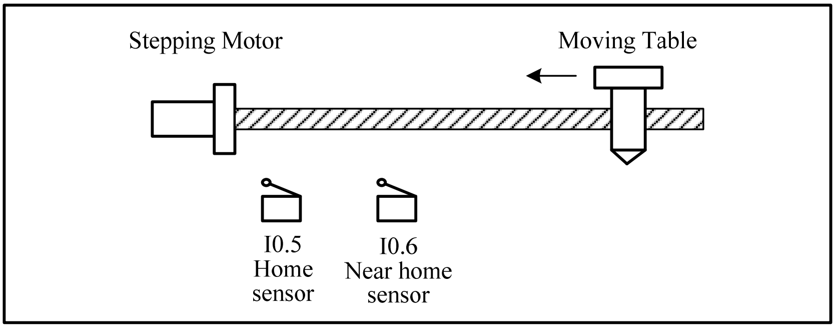
LD	 (* Network 0 *) (* Set the initial frequency and the maximum frequency *) (* Network 1 *) (* Set the acceleration/deceleration time and the target value *) (* Network 2 *) (* Reset the emergency-stop flag *) (* Network 3 *) (* Call the PABS instruction *) %SM0.1 MOVE EN ENO W#400 IN OUT %VW300 MOVE EN ENO W#5000 IN OUT %VW302 NUL %SM0.1 MOVE EN ENO W#200 IN OUT %VW304 MOVE EN ENO DI#16000 IN OUT %VD306 NUL %I0.0 %SM201.7 R %SM0.0 PABS EN ENO 0 AXIS DONE %M2.0 %I0.0 EXEC ERR %M2.1 %VW300 MINF ERRID %VB2 %VW302 MAXF %VW304 TIME %VD306 POS NUL
IL	(* Network 0 *) (*Set the initial frequency and the maximum frequency*) LD %SM0.1 MOVE W#400, %VW300 MOVE W#5000, %VW302 (* Network 1 *) (*Set the acceleration/deceleration time and the target value*) LD %SM0.1 MOVE W#200, %VW304 MOVE DI#16000, %VD306 (* Network 2 *) (*Reset the emergency-stop flag*) LD %I0.0 R %SM201.7 (* Network 3 *) (*Call the PABS instruction*) LD %SM0.0 PABS 0, %I0.0, %VW300, %VW302, %VW304, %VD306, %M2.0, %M2.1, %VB2

- Home

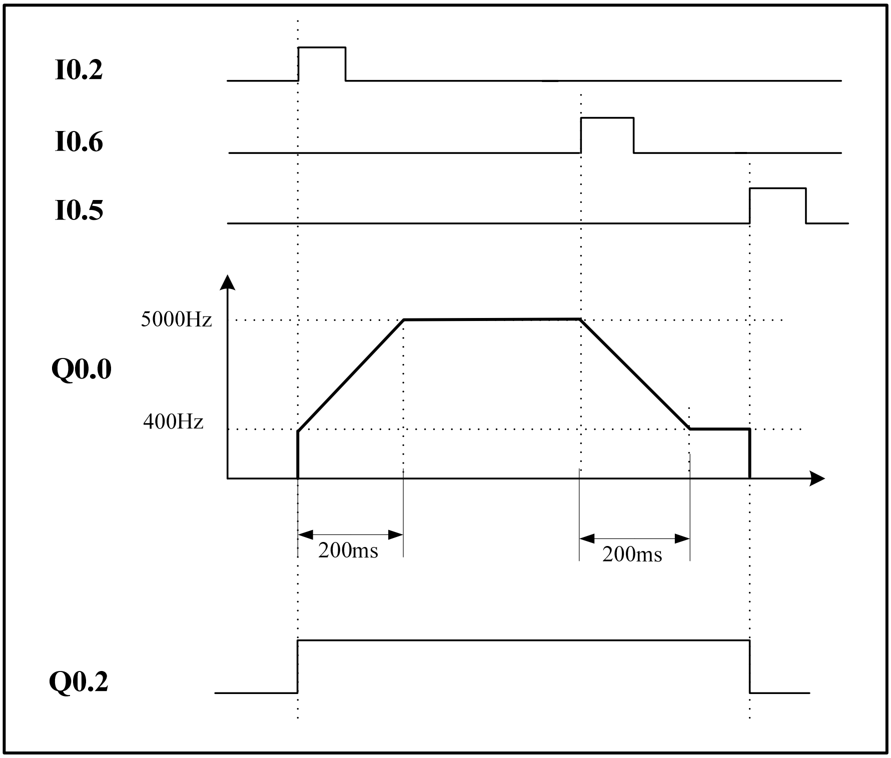
I0.2 is used for starting to return to the home position,

Time Sequence Diagram

Supposing that the moving is in the following initial status:



During the motion, Q0.2 is 1 because of moving backwards。

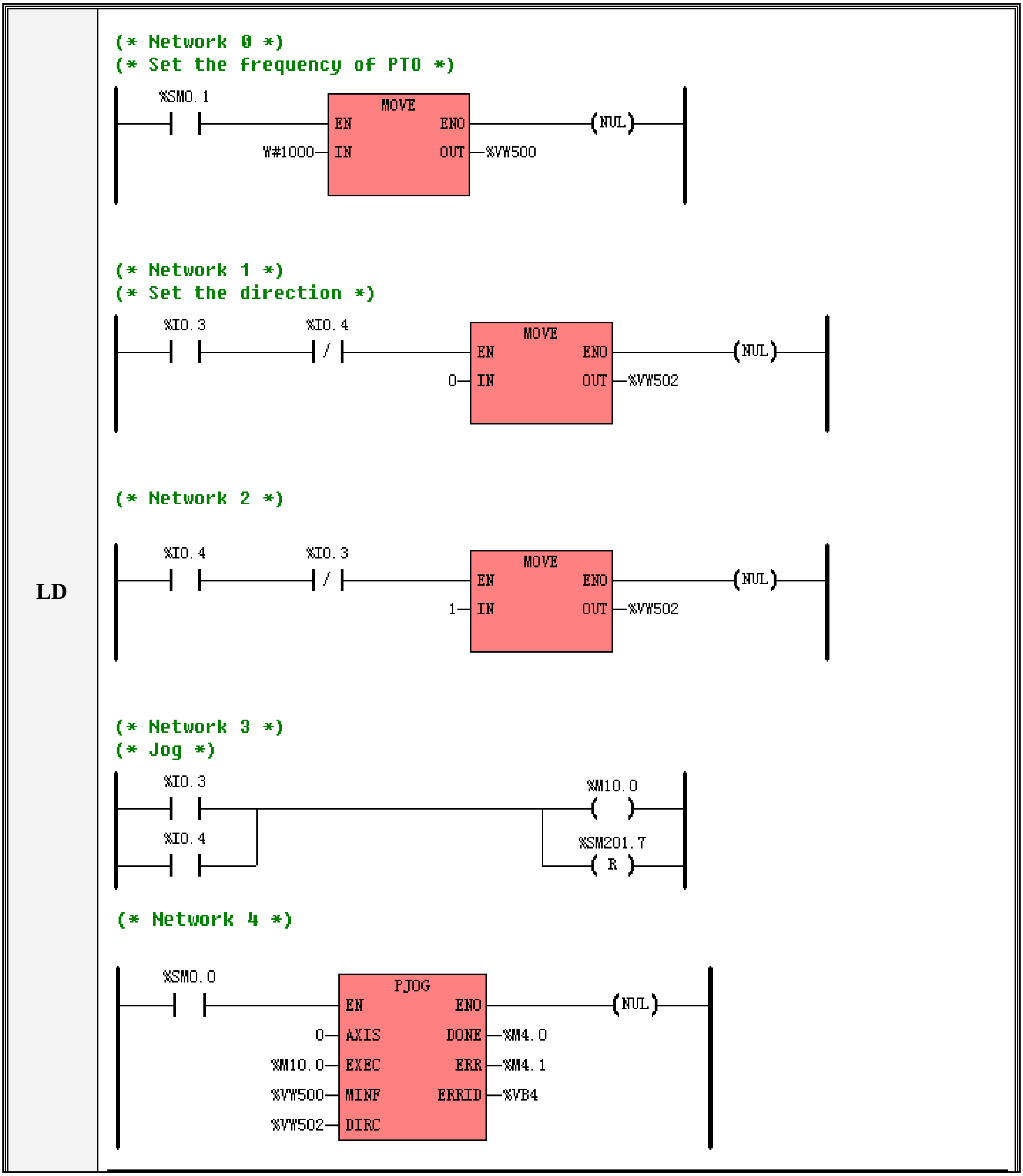


IL	(* Network 0 *) (*use both the home and the near home input; move backwards*) LD %SM0.1 MOVE 0, %VW396 MOVE 1, %VW398 (* Network 1 *) (*set the initial frequency, maximum frequency and acceleration/deceleration time*) LD %SM0.1 MOVE W#400, %VW400 MOVE W#5000, %VW402 MOVE W#200, %VW404 (* Network 2 *) (*Reset the emergency-stop flag*) LD %I0.2 R %SM201.7 (* Network 3 *) LD %SM0.0 PHOME 0, %I0.2, %I0.5, %I0.6, %VW396, %VW398, %VW400, %VW402, %VW404, %M3.0, %M3.1, %VB3

- Jog

I0.3 is used for starting forward jog. I0.4 is used for starting backward jog.

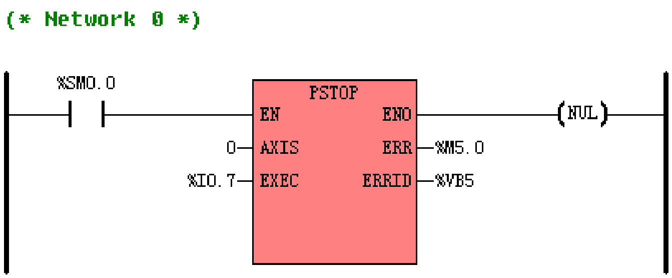
If I0.3 and I0.4 are all 1, then the most recent direction is followed.



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

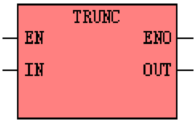
There are 2 overrun contacts at the 2 ends of the feed screw, and they are connected in parallel to I0.7 as the emergency-stop signal

LD	 (* Network 0 *) %SM0.0 PSTOP EN ENO 0 AXIS ERR %M5.0 %I0.7 EXEC ERRID %VB5 NUL
IL	(* Network 0 *) LD %SM0.0 PSTOP 0, %I0.7, %M5.0, %VB5

6.17 Additional Instructions

6.17.1 LINCO (Linear Calculation)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	LINCO			
IL	LINCO	LINCO *IN_L, IN_H, OUT_L, OUT_H, RATIO, IN, DOUT, ROUT*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN_L	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constants
IN_H	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ, Constants
OUT_L	Input	REAL	V, L, Constants
OUT_H	Input	REAL	V, L, Constants
RATIO	Input	REAL	Constants
IN	Input	INT	I, Q, V, M, L, SM, T, C, AI, AQ
DOUT	Output	DINT	Q, M, V, L, SM
ROUT	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:

$$ROUT = RATIO * (k*IN + b)$$

$$DOUT = \mathrm{TRUNC}(ROUT)$$

Therein, $k = \frac{OUT\_H - OUT\_L}{IN\_H - IN\_L}$, $b = OUT\_L - k \times IN\_L$.

- **LD**

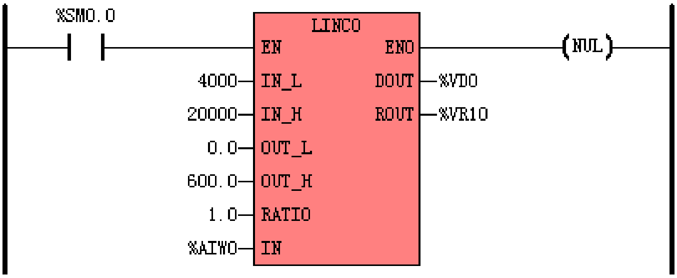
If *EN* is 1, this instruction is executed.

- **IL**

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

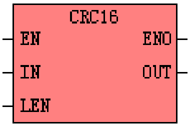
➢ Examples

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-K3. Now the KINCO-K3 needs to calculate the actual temperature value.

LD	 %SM0.0 LINCO EN ENO (NUL) 4000 IN_L DOUT %VD0 20000 IN_H ROUT %VR10 0.0 OUT_L 600.0 OUT_H 1.0 RATIO %AIW0 IN
IL	LD %SM0.0 LINCO 4000, 20000, 0.0, 600.0, 1.0, %AIW0, %VD0, %VR10

6.17.2 CRC16 (16-Bit CRC)

➢ Description

	Name	Usage	Group	□ CPU304 □ CPU304EX □ CPU306 ☑ CPU306EX ☑ CPU308
LD	CRC16			
IL	CRC16	CRC16 *IN, OUT, LEN*	U	

Operands	Input/Output	Data Type	Acceptable Memory Areas
IN	Input	BYTE	I, Q, M, V, L, SM
LEN	Input	BYTE	I, Q, M, V, L, SM, Constant
OUT	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 varialbe after *OUT* is the low byte of the CRC.

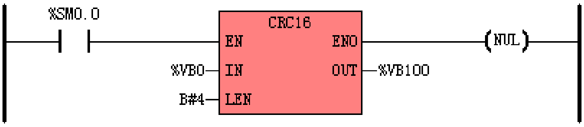
- **LD**

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

- **IL**

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

- Examples

<table>
<tr><td><b>LD</b></td><td> </td><td>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.</td></tr>
<tr><td><b>IL</b></td><td colspan="2">LD %SM0.0
CRC16 %VB0, %VB100, B#4</td></tr>
<tr><td><b>Result</b></td><td colspan="2">The result is as the following:
<table>
<tr><th colspan="4">The data to be checked</th><th colspan="2">16-bit CRC</th></tr>
<tr><td>VB0</td><td>VB1</td><td>VB2</td><td>VB3</td><td>VB100</td><td>VB101</td></tr>
<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>
</table></td></tr>
</table>

7 Appendix A Communicate Using Modbus RTU Protocol

Default, the KINCO-K3 serves as a slave using Modbus RTU Protocol, and can communicate with a Modbus RTU master directly.

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

2. Accessible Memory Ranges of CPU306

(1) In some equipment, modbus registers begin with 1, so 1 should be added to each data in this colume.

- **For CPU304**

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 --- M31.7	DI/DO	64 --- 319
AI	-----	AI	-----
AQ	-----	AO	-----
V	VW0 ---VW2046	AI/AO	16 ---1039

- **For CPU304EX and CPU306**

Area	Range	Type	Corresponding Modbus Registers
I	I0.0 --- I7.7	DI	0 --- 63
Q	Q0.0 --- Q7.7	DO	0 --- 63
M	M0.0 --- M31.7	DI/DO	64 -- 319
AI	AIW0 --- AIW30	AI	0 --- 15
AQ	AQW0 --- AQW30	AO	0 --- 15
V	VW0 ---VW4094	AI/AO	16 -- 2063

- **For CPU306EX and CPU308**

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 -- 575
AI	AIW0 --- AIW62	AI	0 --- 31
AQ	AQW0 --- AQW62	AO	0 --- 31
V	VW0 ---VW4094	AI/AO	100 -- 2147

8 Appendix B Assignments and Functions of SM

After each scan cycle, the firmware of the KINCO-K3 shall update the system data stored in SM (Systme Memory) area. You can read some SM addresses to evaluate the current system status, and you can write to some SM addresses to control some system functions.

1. SMB0

SMB0 (SM0.0 --- SM0.7) are updated by the CPU after each scan cycle. These bits are read-only. Your program can read the status of these bits and make use of them.

SM Bit	Description
SM0.0	Always ON
SM0.1	ON during the first scan cycle only. Usually used for some initializations.
SM0.2	If the data in RAM is lost, this bit is ON during the first scan cycle, and later cleared to FALSE.
SM0.3	Provide a pulse train (50% duty cycle) with a cycle time of 1s.
SM0.4	Provide a pulse train (50% duty cycle) with a cycle time of 2s..
SM0.5	Provide a pulse train (50% duty cycle) with a cycle time of 4s.
SM0.6	Provide a pulse train (50% duty cycle) with a cycle time of 60s.
SM0.7	Reserved

2. SMW22 and SMW24

SMW22 is used to store the cycle time value of Timed interrupt 0 (event 3), range: 1~65535, unit: ms. If SMW22 is set to be 0, Timed interrupt 0 is disenabled. The default value of SMW22 is 0.

SMW24 is used to store the cycle time value of Timed interrupt 1 (event 4), range: 1~65535, unit: ms. If SMW24 is set to be 0, Timed interrupt 1 is disenabled. The default value of SMW24 is 0.

3. SMW26 and SMW28

SMW26 and SMW28 are used to store the numerical values of the two analogue potentiometers; SMW26 is for No. 1 potentiometer and SMW28 for No. 0 potentiometer.

The CPU automatically update the values of S MW26 and SMW28. SMW26 and SMW28 are read-only.

4. SMB31 and SMW32

In the CPU304, CPU304EX and CPU306, these two variables are used for permanent data backup.

Please refer to Appendix C Permanent Data Backup for more details.

9 Appendix C Permanent Data Backup

A value stored in the special range of V area can be written into FRAM under the control of your program for permanent backup. SMB31 and SMW32 are used for the write control.

The value of SMB31 decides the write mode. Notice: If SMB31 has been assigned with multiple values before the execution of command for writing into FRAM, the latest assignment prevails.

1. The memory range for permanent backup

The following table lists the V area ranges that can be saved into FRAM. We call this area as the Permanent Data Area.

	CPU304	**CPU304EX, CPU306, CPU306EX and CPU308**
Length	128 bytes	255 bytes
Range	VB1648~VB1775	VB3648~VB3902

2. How to backup data permanently

2.1 For the CPU306EX and CPU308

The CPU306EX and CPU308 write the data from the Permanent Data Area into FRAM automatically. You just write the data to be stored permanently into the Permanent Data Area. For example:

```
  (*NETWORK    0*)
LD       %SM0.0
MOVE     %AIW0, %VW3648          (* store the value of AIW0 permanently *)
SPD      1, W#1000, %VD4000      (* calculate the frequency of the pulse train from HSC1 *)
                                 (* and store the frequency permanently *)
```

2.2 For the CPU304, CPU304EX and CPU306

When using the CPU304, CPU304EX and CPU306, you can store the data according to the following steps:

(1) Write the data to be to be stored permanently into the Permanent Data Area.

(2) Program using SMB31 and SMW32 to move the data from the Permanent Data Area. into FRAM.

2.2.1 SM31.0, SM31.1 and SM31.7

SM31.1	SM31.0	Description
0	0	Save a BYTE (8-bit) value
0	1	Save a BYTE (8-bit) value
1	0	Save a WORD (16-bit) value
1	1	Save a DWORD (32-bit) value

SM31.7	Description
0	Enable writing into FRAM
1	Disenable writing into FRAM

2.2.2 SMW32

The V area address of the data to be saved is stored in SMW32. This value is an offset from VB0.

2.2.3 Writing to FRAM

The command for writing into FRAM: **MOVE *offset*, %SMW32**

The *offset* is an INT offset from VB0, and represents the V area address of the data to be saved. For example, if writing the value of VB3600 into FRAM, the value of the *offset* should be 3600. Notice: At the end of each scan cycle, the CPU shall execute the write command to write the data to be saved into FRAM.

The following is an example in IL language.

```
(* NETWORK    0 *)
(* Write VB3649, VW3650, VD3652 into FRAM under the control of M0.0*)
 LDN      %M0.0                    (* If M0.0 is 0 *)
MOVE     B#0, SMB31               (* Disenable writing into FRAM *)

(* NETWORK 1 *)
LD       %M0.0                         (* If M0.0 is 1 *)
MOVE     B#2#10000001, SMB31      (* To save 1 byte *)
MOVE     3649, %SMW32             (* Save VB3649 to FRAM*)
MOVE     B#2#10000010, SMB31      (* To save 1 word (2 bytes) *)
MOVE     3650, %SMW32             (* Save VW3650 to FRAM *)
MOVE     B#2#10000011, SMB31      (* To save 1 double-word (4 bytes) *)
MOVE     3652, %SMW32             (* Save VD3652 to FRAM *)
```