Modicon
Ladder Logic Block Library
User Guide
Volume 1

840USE10100 Version 5.0

043505766 79

a brand of Schneider Electric

Telemecanique

Document Set

At a Glance

This manual consists of four volumes.

Volume 1
- General Information and Instruction Descriptions (A - D)

Volume 2
- Instruction Descriptions (E)

Volume 3
- Instruction Descriptions (F - N)

Volume 4
- General Information and Instruction Descriptions (O - X) and Appendix

Table of Contents

Safety Information

Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a Danger or Warning safety label indicates that an electrical hazard exists, which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

DANGER

DANGER indicates an imminently hazardous situation, which, if not avoided, **will result** in death, serious injury, or equipment damage.

WARNING

WARNING indicates a potentially hazardous situation, which, if not avoided, **can result** in death, serious injury, or equipment damage.

CAUTION

CAUTION indicates a potentially hazardous situation, which, if not avoided, **can result** in injury or equipment damage.

PLEASE NOTE

Electrical equipment should be serviced only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material. This document is not intended as an instruction manual for untrained persons.

About the Book

At a Glance

Document Scope

This documentation will help you configure LL 984 instructions to any controller using ProWorx NxT, ProWorx 32 or Modbus Plus. Examples in this book are used with ProWorx 32. For LL 984 using Concept software, see Concept Block Library LL984 (840USE49600).

Validity Note

The data and illustrations found in this book are not binding. We reserve the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be construed as a commitment by Schneider Electric.

Related Documents

Title of Documentation	Reference Number
Concept Block Library LL 984	840USE49600

Product Related Warnings

Schneider Electric assumes no responsibility for any errors that may appear in this document. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When controllers are used for applications with technical safety requirements, please follow the relevant instructions.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this product related warning can result in injury or equipment damage.

User Comments

We welcome your comments about this document. You can reach us by e-mail at TECHCOMM@modicon.com

General Information

I

Introduction

At a Glance

In this part you will find general information about the instruction groups and the use of instructions.

What's in this Part?

This part contains the following chapters:

Ladder Logic Overview

1

At a Glance

Overview

This chapter provides an overview of the Ladder Logic programming language.

What's in this Chapter?

This chapter contains the following topics:

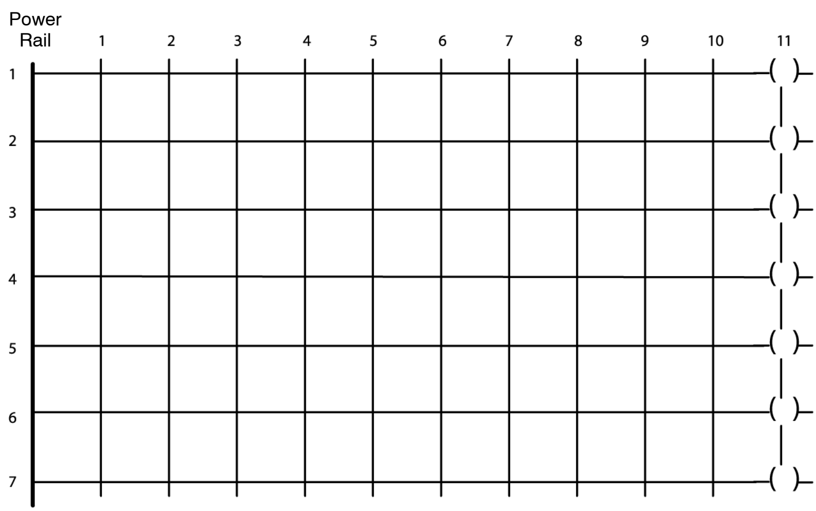
Segments and Networks in Ladder Logic

Overview

Ladder Logic is an easy-to -use graphical programming language that implements relay-equivalent symbology. Its major components are single node elements and multi node instructions. These components are programmed into networks, which are ladder logic constructs of a preset size and shape. A ladder logic program comprises a sequence of networks collected together in one or more segments.

A Ladder Logic Network

A network is a ladder diagram bounded on the left and right by power rails. By convention, the rail on the left is shown and the one on the right is not. Seven rungs (or rows) run from left to right between the two power rails. Each rung is eleven columns wide.

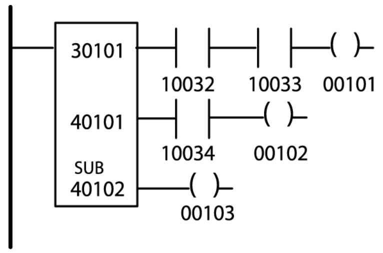


NOTE Only coils can be shown in column 11

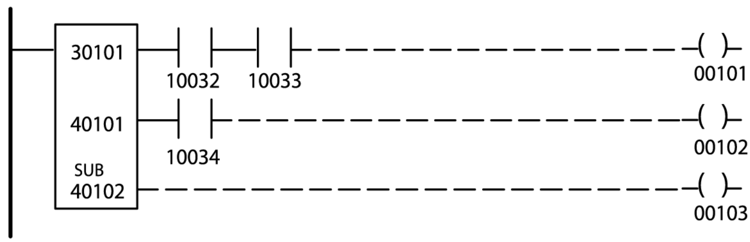
The 77 regions formed by the intersections of rungs and columns are called nodes. Logic elements and instructions can be programmed into these nodes. All 77 nodes in a network may be used to store ladder logic elements and instructions, which are the fundamental building blocks of the logic program. Some rules of placement apply, particularly with respect to coil placement.

A Coil Placement in a Network

When a coil is inserted on a rung of a network, no other logic elements or instructions can be placed to the right of it on that rung. The seven nodes in the 11th column are reserved for displaying coils. Many software panels allow you to select the way you display coils in a network, either in their logic-solve positions or expanded to column 11 where they can all be viewed in parallel.The two examples below show the same logic structure with the coils displayed differently according to user preference. The first example shows the coils displayed in their logic-solve positions and the second example shows the coils displayed in expanded positions.



Coils Displayed in Logic-solve Positions



Coils Displayed in Expanded Positions

Although the coil expansion display shows the coils in the 11th column, they are solved in their real logic-solve position. Coil 00103 is solved immediately after contact 10034 and coil 00102 is solved immediately after contact 10033 in both examples above. Coil 00101 is always the last coil solved in the network.

Ladder Logic Segments

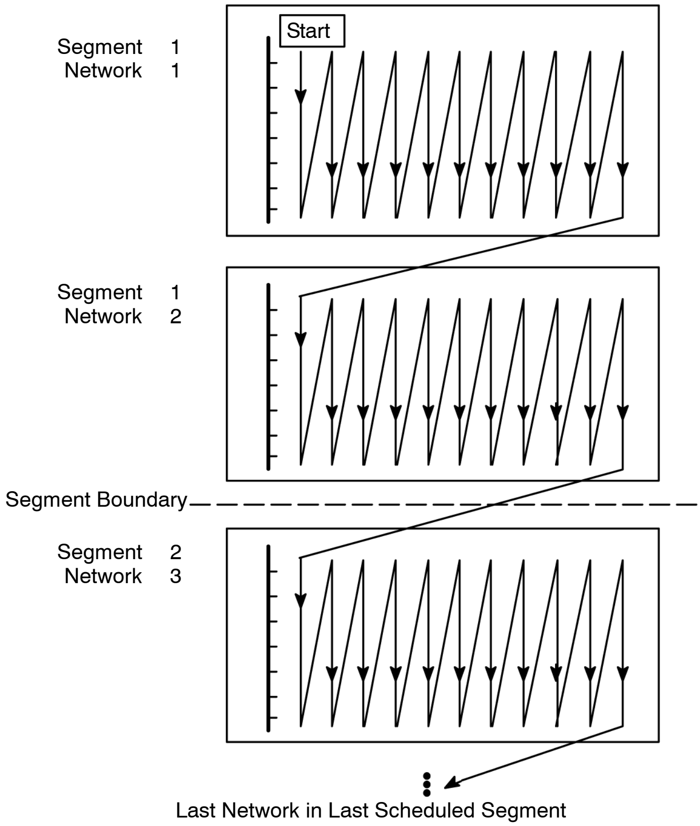
Because the structure of a network is fixed, the logic program generally overlaps into multiple networks. A group of contiguous networks performing a task or subtask in the application program is called a segment. There is no prescribed limit on the number of networks that can be placed in a segment—size is limited only by the amount of UserMemory available and by the maximum amount of PLC scan time(250 ms).For small ladder logic applications, a single segment may be sufficient to store the whole program. For larger applications, such as multi-drop remote I/O applications, several segments may be programmed. As arule in RIO configurations, the number of segments in the program equals the number of I/O drops; you may want to use more segments than drops, but never fewer segments than drops.Segments are numbered 1 ... n, up to a maximum of 32, in the order they are created by the programmer. You may modify the order in which segments are solved with the segment scheduler, an editor available with your panel software that allows you to adjust the order-of-solve table in system memory. Refer to Appendix A for a description of how to improve system performance via the segment scheduler.With some PLCs, you may also create an unscheduled segment that contains one or more ladder logic subroutines, which can be called from the scheduled segments via the JSR function.

How a PLC Solves Ladder Logic

Overview

The PLC scans the ladder logic program sequentially in the following order.

- Segments are scanned according to the way they are scheduled in an order-of-solve table known as the segment scheduler. The segment scheduler can be customized during system configuration, or it can default to a standard scanning sequence (segment 1 followed by segment 2 followed by segment 3, etc.)
- Networks in each segment are scanned contiguously.
- Nodes within each network are scanned top to bottom, left to right.



The PLC begins solving logic in the network at the top of the leftmost column and proceeds down, then moves to the top of the next column and proceeds down, as shown in the illustration. Each node is solved in the order it is encountered in the logic scan. Power flow within the network is down each column from left to right, never from bottom to top and never from right to left.

Ladder Logic Elements and Instructions

Overview

There is a core set of ladder logic elements (contacts, coils, vertical and horizontal shorts) and instructions built into all PLC firmware packages. Additional instructions are available for specific PLC types as either built-in or loadable instructions. This section provides a brief list of the available instructions and their functions; a detailed description of all instruction, including the PLC models they are available on, is provided in later chapters of this book.

Standard Ladder Logic Elements

Symbol	**Definition**	**Nodes Consumed**
┤├	A normally open (N.O.) contact	1
┤/├	A normally closed (N.C.) contact	1
┤↑├	A positive transitional (P.T.) contact	1
┤↓├	A negative transitional (N.T.) contact	1
–( )–	A normal coil	1
–(L)– –(M)–	A memory-retentive or latched coil; the two symbols mean the same thing, and the user may select the preferred version for on-line display.	1
—	A horizontal short	1
│	A vertical short	1

Standard Ladder Logic Instructions for all PLCs

Counter and Timer Instructions

Instruction	Definition	Nodes Consumed
UCTR	Counts up from 0 to a preset value	2
DCTR	Counts down from a preset value to 0	2
T1.0	Timer that increments in seconds	2
T0.1	Timer that increments in tenths of a second	2
T.01	Timer that increments in hundredths of a second	2

Integer Math Instructions

Instruction	Definition	Nodes Consumed
ADD	Adds top node value to middle node value	3
SUB	Subtracts middle node value from top node value	3
MUL	Multiplies top node value by middle node value	3
DIV	Divides top node value by middle node value	3

DX Move Instructions

Instruction	Definition	Nodes Consumed
R→T	Moves register values to a table	3
T→P	Moves specified table values to a register	3
T→T	Moves a specified set of values from one table to another table	3
BLKM	Moves a specified block of data	3
FIN	Specifies first-entry in a FIFO queue	3
FOUT	Specifies first-entry out of a FIFO queue	3
SRCH	Performs a table search	3
STAT	Displays status registers from status table in system memory	1

DX Matrix Instructions

Instruction	Definition	Nodes Consumed
AND	Logically ANDs two matrices	3
OR	Does logical inclusive OR of two matrices	3
XOR	Does logical exclusive OR of two matrices 3	3
COMP	Performs logical complement of values in a matrix	3
CMPR	Logically compares values in two matrices	3
MBIT	Logical bit modify	3
SENS	Logical bit sense	3
BROT	Logical bit rotate	3

Skip-Node Instruction

Instruction	Definition	Nodes Consumed
SKP	Skips a specified number of networks in a ladder logic program	1

Some ladder logic instructions are standard (built in) to some PLCs but unavailable in others. For example, PLCs with the Modbus Plus communication capability built in it are shipped with an MSTR instruction in the firmware while PLCs that cannot operate on Modbus Plus do not support this instruction. Here is a list of these select built-in instruction

Built-in Ladder Logic Instruction for Select PLCs

Bit Manipulation Instructions

Instruction	Definition	Nodes Consumed
NOBT	Uses a register to represent 16 bits as N.O. contacts	2
NCBT	Uses a register to represent 16 bits as N.C. contacts	2
NBIT	Uses an output register to represent 16 bits as normal coils	2
SBIT	Latches a bit in an output register to remain ON	2
RBIT	Clears a bit that has been set via the SBIT instruction	2

Other Math Instructions

Instruction	Definition	Nodes Consumed
AD16	Signed/unsigned 16-bit addition	3
SU16	Signed/unsigned 16-bit subtraction	3
TEST	Compares the magnitudes of the values in the top and middle nodes	3
MU16	Signed/unsigned 16-bit multiplication	3
DV16	Signed/unsigned 16-bit division	3
ITOF	Signed/unsigned integer-to-floating point conversion	3
FTOI	Floating point-to-signed/unsigned integer conversion	3
EMTH	Performs 38 math operations, including floating point math operations and extra integer math operations such as square root	3
BCD	Converts binary values to BCD values and BCD values to binary values	3
Equation Network	Uses an entire ladder logic network as an editing environment where a user can enter equations in a standard syntax	77

Interrupt Instructions

Instruction	Definition	Nodes Consumed
ITMR	Defines an interval timer that generates interrupts into the normal logic scan and initiates an interrupt handling subroutine	2
ID	Interrupt disable	1
IE	Interrupt enable	1
BMDI	Masks timer-generated and local I/O-generated interrupts, performs a block data move, then unmasks the interrupts	3
IMIO	Permits immediate access of specified I/O modules from within ladder logic	2

ASCII Message Instructions

Instruction	Definition	Nodes Consumed
READ	Reads data entered at an ASCII device into the PLC via its RIO link	3
WRIT	Sends a message from the PLC to an ASCII device via its RIO link	3
COMM	Combines both ASCII READ and WRITE capabilities for simple (canned) messages in the Micro PLCs	3

Ladder Logic Subroutine Instructions

Instruction	Definition	Nodes Consumed
JSR	Jumps from scheduled logic scan to a ladder logic subroutine	2
LAB	Labels the entry point of a ladder logic subroutine	1
RET	Returns from the subroutine to scheduled logic	1
CTIF	Used to set up high-speed input terminals on a Micro PLC for scheduled-logic interrupts and/or counter/timer operations	3

Other Special-Purpose Instructions

Instruction	Definition	Nodes Consumed
CKSM	Calculates any of four types of checksum operations (CRC-16, LRC, straight CKSM, and binary add)	3
MSTR	Specifies a function from a menu of networking operations	3
PID2	Performs proportional-integral-derivative calculations for closed-loop control	3
PCFL	Accesses advanced functions from a process control library	3
TBLK	Moves a block of data from a table to another specified block area	3
BLKT	Moves a block of registers to specified locations in a table	3
SCIF	Provides tenor drum sequencer functionality and the ability to do input comparisons within the application program	3
T1MS	A timer that increments in milliseconds	3
IKBR	Performs an indirect block read operation—i.e., copies specified registers to a working block of holding registers	3
IBKW	Performs an indirect block write operation—i.e., copies registers from a working block to individual register locations	3

Other instructions are available for specific PLCs as loabable functions. Loadables support optional software development products that can be purchased for special applications. The loadable instructions may be used only with specific PLC models. Loadable instructions include:

Instruction	**Definition**	**Nodes Consumed**
HSBY	Sets up a 984 hot standby back-up PLC that takes control of the application if the primary PLC goes down	3
CHS	Optional method for setting up a Quantum hot standby back-3	3
CALL	Supports 984 Coprocessor option module applications	3
MBUS PEER	For initiating message transactions on a Modbus II network	3
ESI	Optional instruction in Quantum PLCs that supports the 140 ESI 062 10 Quantum ASCII module	3
FNxx	A three-node template for creating custom loadable instructions via Assembly or C source code	3
DRUM ICMP	Supports sequence control application logic in some PLC models that do not have the built-in SCIF instruction	3
MATH DMATH	Support some square root, logarithm, and double-precision math functions in PLCs that cannot support the Enhanced Math library	3
EARS	Supports an event/alarm recording system by tracking events/alarms and reporting time-stamped messages	3
EUCA	Performs an engineering unit conversion algorithm	3
HLTH	Detects changes in the I/O system and reports problems on an exception-only basis	3

Memory Allocation in a PLC

2

At a Glance

Overview

This chapter discusses memory allocation in a PLC.

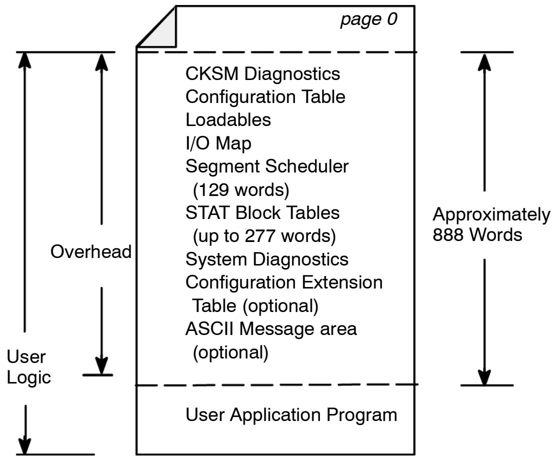
What's in this Chapter?

This chapter contains the following topics:

User Memory

Overview

User memory is the space provided in the PLC for the logic program and for system overhead. User memory sizes vary from 1K ... 64K words, depending on PLC type and model. Each word in user memory is stored on page 0 in the PLC's memory structure; words may be either 16 or 24 bits long, depending on the CPU size.

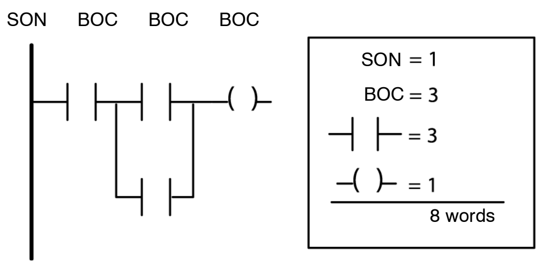


User Logic

The amount of space available for application logic is calculated by subtracting the amount of space consumed by system overhead from the total amount of user logic. System overhead in a relatively conservative system configuration can be expected to consume around 1000 words; system configurations with moderate or large I/O maps will require more overhead.

User Memory

Ladder logic requires one word of either 16-bit or 24-bit memory to uniquely identify each node in an application program. Contacts and coils each occupy one node, and therefore one word. Instructions, which usually comprise two or three nodes, require two or three words, respectively. Other elements that control program scanning three words, respectively. Other elements that control program scanning — start of a network (SON), beginning of a column (BOC), and horizontal shorts — use one word of user logic memory as well.



Note: Vertical shorts do not consume any words of user memory.

System Overhead

System overhead refers to the contents of a set of tables where the system's size, structure, and status are defined. Some overhead tables have a predetermined amount of memory allocated to them. The configuration table, for example, contains 128 words, and the order-of-solve table (the segment scheduler) contains 129 words. Other tables, such as the I/O map (a.ka. traffic cop), can consume a large amount of memory, but its size is not predetermined. Optional pieces of system overhead — e.g., the loadable table, the ASCII message area, the configuration extension table — may or may not consume memory depending on the requirements of your application.

Memory Backup

User memory is stored in CMOS RAM. In the event that power is lost, CMOS RAM is backed up by a long-life (typically 12-month) battery. In many PLC models, the battery is a standard part of the hardware package; in smaller-scale PLCs — e.g., the Micro PLCs — a battery is available as an option.In the case of the Micro PLCs, where the battery is an option, an area in its Flash memory is available for backing up user logic. (Flash is a standard feature on the Micros.)

State RAM Values

Overview

As part of your PLC's configuration process, you specify a certain number of discrete outputs or coils, discrete inputs, input registers, and output holding registers available for application control. These inputs and outputs are placed in a table of 16-bit words in an area of system memory called state RAM.

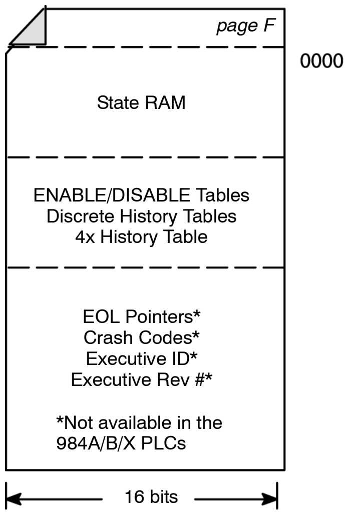
Referencing System for Inputs and Outputs

The system uses a reference numbering system to identify the various types of inputs and outputs. Each reference number has a leading digit that identifies its data type (discrete input, discrete output, register input, register output) followed by a string of digits indicating its unique location in state RAM.

Reference Indicator	Reference Type	Meaning
0x	discrete output or coil	Can be used to drive a real output through an output module or to set one or more internal coils in state RAM. The state of a coil can be used to drive multiple contacts.
1x	discrete input	Can be used to drive contacts in the logic program. Its ON/OFF state is controlled by an input module.
3x	input register	Holds numerical inputs from an external source—for example, a thumbwheel entry, an analog signal, data from a high speed counter. A 3x register can also be used to store 16 contiguous discrete signals, which may be entered into the register in either binary or binary coded decimal (BCD) format.
4x	output holding register	Can be used to store numerical (decimal or binary) information in state RAM or to send the information to an output module.
6x	extended memory register	Stores binary information in extended memory area; available only in PLCs with 24-bit CPUs that support extended memory—the 984B, the E984-785, and the Quantum Automation Series PLCs

Storing a Discreet and Register Data in State RAM

State RAM data is stored in 16-bit words on page F in System Memory. The state RAM table is followed by a discrete history table that stores the state of the bits at the end of the previous scan, and by a table of the current ENABLE/DISABLE status of all the discrete (0x and 1x) values in state RAM.

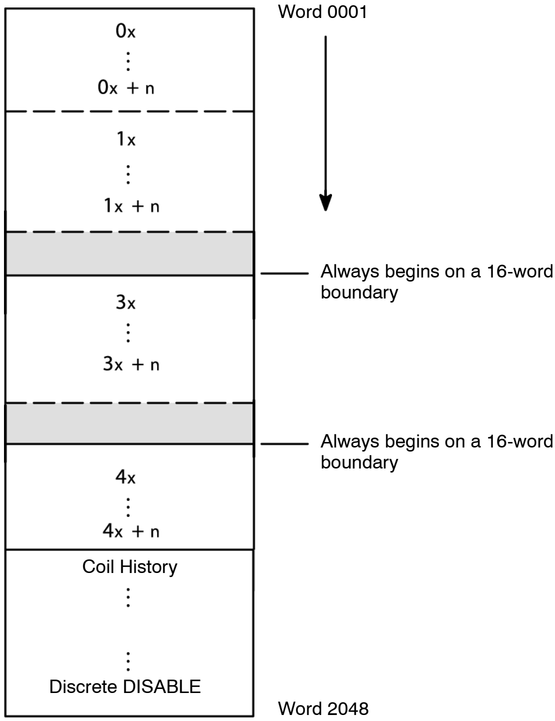


Each 0x or 1x value implemented in user logic is represented by one bit in a word in state RAM, by a bit in a word in the history table, and by a bit in a word in the DISABLE table. In other words, for every discrete word in the state RAM table there is one corresponding word in the history table and one corresponding word in the DISABLE table.Counter input states for the previous scan are represented on page F in an up-counter/down-counter history table. Each counter register is represented by a single bit in a word in the table; a value of 1 indicates that the top input was ON in the last scan, and a value of 0 indicates that the top input was OFF in the last scan.

State RAM Structure

Overview

Words are entered into the state RAM table from the top down in the following order.



Discrete references come before registers, the 0x words first followed by the 1x words. The discrete references are stored in words containing 16 contiguous discrete references.The register values follow the discrete words. Blocks of 3x and 4x register values must each begin at a word that is a multiple of 16. For example, if you allocate five words for eighty 0x references and five words for eighty 1x references, you have used words 0001 ... 0010 in state RAM. Words 0011 ... 0016 are then left empty so that the first 3x reference begins at word 0017.

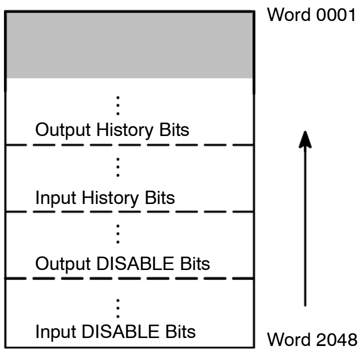
Minimum Required State RAM Values

A minimum configuration consists of the following allocations in state RAM.

Reference Type	Minimum Words for Modsoft	Minimum Words for P190	Minimum Bits for Modsoft	Minimum Bits for discretes
Discrete out (0X)	3	1	48	16
Discrete in (1x)	1	1	16	16
Register in (3x)	1	1		
Register out (4x)	1	1		

History and Disable Bits for Discrete References

For each word allocated to discrete references, two additional words are allocated in the history/disable tables. These tables follow the state RAM table on page F in system memory. They are generated from the bottom up in the following manner.



The Configuration Table

Overview

The configuration table is one of the key pieces of overhead contained in system memory. It comprises 128 consecutive words and provides a means of accessing information defining your control system capabilities and your user logic program. With your programming panel software, you can access the configurator editor, which allows you to specify the configuration parameters — such as those shown on the following page — for your control system.When a PLC's memory is empty — in a state called DIM AWARENESS — you are not able to write a I/O map or a user logic program. Therefore, the first programming task you must undertake with a new PLC is to write a valid configuration table using your configurator editor.

Assigning a Battery Coil

A 0x coil can be set aside in the configuration to reflect the current status of the PLC's battery backup system. If this coil has been set and is queried, it displays a discrete value of either 0, indicating that the battery system is healthy, or 1 indicating the battery system is not healthy.

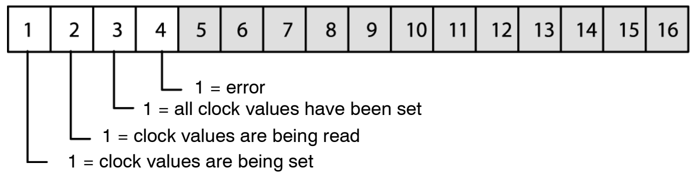
Assigning Timer Register

A 4x register can be set aside in the configuration as a synchronization timer. It stores a count of clock cycles in 10 ms increments. If this register is set and queried, it displays a free-running value that ranges from 0000 to FFFF hex with wrap-around to 0000.

Note: If you are doing explicit address routing in bridge mode on a Modbus Plus network, the location of the explicit address table in the configuration is dependent on the timer register address—i.e., a timer register must be assigned in order to create the explicit address table. The explicit address table can consist of from 0 ... 10 blocks, each block containing five consecutive 4x registers. The address of the first block in the explicit address table begins with the 4x register immediately following the address assigned to the timer register. Therefore, when you assign the timer register, you must choose a 4x register address that has the next 5 ... 50 registers free for this kind of application.

Time of Day Clock

When a 4x holding register assignment is made in the configurator for the time of day (TOD) clock, that register and the next seven consecutive registers (4x ... 4x + 7) are set aside in the configuration to store TOD information. The block of registers is implemented as follows.

<table>
<tr><th>Register</th><th>Definition</th></tr>
<tr><td>4X</td><td>The control register:


</td></tr>
<tr><td>4X+1</td><td>Day of the week (Sunday = 1, Monday = 2, etc.)</td></tr>
<tr><td>4x+2</td><td>Month of the year (Jan. = 1, Feb. = 2, etc.)</td></tr>
<tr><td>4x+3</td><td>Day of the month (1... 31)</td></tr>
<tr><td>4x+4</td><td>Year (00... 99)</td></tr>
<tr><td>4x+5</td><td>Hour in military time (0... 23)</td></tr>
<tr><td>4x+6</td><td>Minute (0... 59)</td></tr>
<tr><td>4x+7</td><td>Second (0... 59) When a 4x holding register assignment is made in the configurator for the time of day (TOD) clock, that register and the next seven consecutive registers (4x... 4x + 7) are set aside in the configuration to store TOD information.</td></tr>
</table>

The block of registers is implemented as follows. For example, if you configured register 40500 for your TOD clock, set the bits appropriately as shown above, then read the clock values at 9:25:30 on Tuesday, July 16, 1991, the register values displayed in decimal format would read:

Register	Definition
400500	0110000000000000
400501	3 (decimal)
400502	7 (decimal)
400503	16 (decimal)
400504	91 (decimal)
400505	9 (decimal)
400506	25 (decimal
400507	30 (decimal)

Data Type	Format	Default Setting	Notes and Exceptions
# of coils	Even multiple of 16	16	
# of discrete inputs	Even multiple of 16	16	
# of register outputs		01	
# of register inputs		01	
# of I/O drops	Up to 32, depending on PLC type	01	Used only when I/O is configured in drops.
# of I/O modules	Up to 1024, depending on the PLC type	00	Not displayed by editor; used by system to cal-culate I/O map words.
# of logic segments	Generally equal to the # of drops	00	Add one additional seg-ment for subroutines.
# of I/O channels	Even number from 02 to 32	02	Used only when I/O is configured in channels.
Memory Size	PLC- dependent	PLC- dependent	

Modbus (RS-232) Port Parameters

Communication Mode	ASCII or RTU	RTU	Notes
Baud Rate	50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, 19200	9600	
Parity	ON/OFF, EVEN/ODD	ON/EVEN	
Stop bit(s)	1or 2	2	
Device address	001...247	001	
Delay time (in ms)	10...20 (representing 10...20 ms)	01 (10 ms)	Modbus port delay times are implemented only in the 984A/B/X PLCs.

ASCII Message Table

# of messages	Up to 9999	00	If your PLC doesn't support remote I/O, it cannot support ASCII devices. (exception: The Micros)
Size of message area	Decimal > 0 < difference between memory size (32K or 64K) and system overhead	00	
#of ASCII ports	Two per drop, up to 32	00	
ASCII port parameters	Baud Parity # of stop bits # of data bits per character Presence of a keyboard	1200 ON/EVEN 01 08 NONE	
ASCII input	A 4x value representing the first of 32 registers for simple ASCII input	NONE	Only a 984B PLC supports simple ASCII input
ASCII output)	A 4x value representing the first of 32 registers for simple ASCII output	NONE	Only a 984A and 984B PLC supports simple ASCII output

Special Functions

Skip Functions Allowed	**YES/NO**	**No**	
Timer Register	A 4x register set aside to hold a number of 10 ms clock cycles	NONE	
TOD Clock	A 4x register the first of eight reserved for time of day values	NONE	
Battery Coil	A 0x reference reflecting the status of battery backup system 00000	00000	Once a battery coil is placed in a Configuration Table, it cannot be removed.

Loadable Instructions

Install Loadable	**PROCEED or CANCEL**	**Various controllers support different kinds of loadable instruction sets. Make sure that your loadables and controller are compatible.**
Delete Loadable(s)	DELETE ALL, DELETE ONE or CANCEL	Various controllers support different kinds of loadable instruction sets. Make sure that your loadables and controller are compatible

Writing Configurator Data to System Memory

Write data as specified	PROCEED or CANCEL	NONE	PROCEED will overwrite any previous Table data.

The I/O Map Table

Overview

Just as a PLC needs to be physically linked to I/O modules in order to become a working control system, the references in user logic need to be linked in the system architecture to the signals received from the input modules and sent to the output modules. The I/O map table provides that link.

Determining the Size of the I/O Map Table

The I/O map directs data flow between the input/output signals and the user logic program; it tells the PLC how to implement inputs in user logic and provides a pathway down which to send signals to the output modules. The I/O map table, which is stored on page 0 in system memory, consumes a large but not predetermined amount of system overhead.Its length is a function of the number of discrete and register I/O points your system has implemented and is defined by the type of I/O modules you specify in the configuration table.
The minimum allowable size of the I/O map table is nine words.

Writing Data to the I/O Map

With your programming panel software, you can access a I/O map editor that allows you to define:

- the number of drops in the remote I/O system
- the number of discretes/registers that may be used for input and output
- the number, type and slot location of the I/O modules in the drop
- the reference numbers that link the discrete/registers to the I/O modules
- drop hold-up time for each I/O drop
- ASCII messaging port addresses (if used) for any drop

Ladder Logic Opcodes

3

At a Glance

Overview

This chapter discusses Ladder Logic opcodes.

What's in this Chapter?

This chapter contains the following topics:

Translating Ladder Logic Elements in the System Memory Database

Overview

A PLC automatically translates symbolic ladder elements and function blocks into database nodes that are stored on page 0 in system memory. A node in ladder logic is a 16- or 24-bit word — an element such as a contact translates into one database node, while an instruction such as an ADD block translates into three database nodes. The database format differs for 16-bit and 24-bit nodes.

16-bit Node Format

x	x	x	x	x	y	y	y	y	y	y	y	y	z	z	z

24-bit Node Format

x	x	x	x	x	x	x	x	y	y	y	y	y	y	y	y	y	y	y	y	y	z	z	z

The five most significant bits in a 16-bit node and the eight most significant bits in a 24-bit node — the x bits — are reserved for opcodes . An opcode defines the type of functional element associated with the node — for example, the code 01000 specifies that the node is a normally open contact, and the code 11010 specifies that the node is the third of three nodes in a multiplication function block.

Translating Logic Elements and Non-DX Functions

When the system is translating standard ladder logic elements and non-DX function blocks, it uses the remaining (y and z) bits as pointers to register or bit locations in State RAM associated with the discretes or registers used in your ladder logic program. With a 16-bit node, 11 bits are available as state RAM pointers, giving you a total addressing capability of 2048 words. The maximum number of configurable registers in most 16-bit machines is 1920, with the balance occupied by up to 128 words (2048 bits) of discrete reference, disable, and history bits. An exception is the 984-680/-685 PLCs, which have an extended registers option that supports 4096 registers in state RAM. With a 24-bit node, 16 bits are available as state RAM pointers. The maximum number of configurable registers in a 24-bit machine is 9999. Opcodes are generally expressed by their hex values.

Opcode	Definition
00	Beginning of a column in a network
01	Beginning of a column in a network
02	Beginning of a column in a network
03	Beginning of a column in a network
04	Start of a network
05	I/O exchange/End-of-Logic
06	Null Element
07	Horizontal short
08	N.O. contact
09	N.C. contact
0A	P.T. contact
0B	N.T. contact
0C	Normal coil
0D	Memory-retentive (latched) coil
0E	Constant quantity skip function
0F	Register quantity skip function
10	Constant value storage
11	Register reference
12	Discrete group reference
13	DCTR instruction
14	UCTR instruction
15	T1.0 instruction
16	T0.1 instruction
17	T.01 instruction
18	ADD instruction

Opcode	Definition
19	SUB instruction
1A	MULT instruction
1B	DIV instruction
31	AD16 instruction
32	SU16 instruction
33	MU16 instruction
34	DV16 instruction
35	TEST instruction
36	ITOF instruction
37	FTOI instruction
5E	PID2 instruction
7F	EMTH instruction
9F	BLKT instruction
BE	LAB instruction
BF	CKSM or MSTR instruction
DE	DMTH or JSR instruction
DF	TBLK instruction
FE	RET instruction

Translating DX Instructions in the System Memory Database

How X and Z bits are used in 16-bit nodes

When you are using a 16-bit CPU, you are left with only four more x-bit combinations 111000, 11101, 11110 and 11111 with which to express opcodes for the DX instructions. To gain the necessary bit values, the system uses the three least significant (Z) bits along with the x bits to express the opcodes.

1	1	1	0	0									z	z	z

z	z	z	
0	0	0	= R→T
0	0	1	= T→R
0	1	0	= T→T
0	1	1	= BLKM
1	0	0	= FIN
1	0	1	= FOUT
1	1	0	= SRCH
1	1	1	= STAT

1	1	1	0	1									z	z	z

z	z	z	
0	0	0	= AND
0	0	1	= OR
0	1	0	= CMPR
0	1	1	= SENS
1	0	0	= MBIT
1	0	1	= COMP
1	1	0	= XOR
1	1	1	= BROT

1	1	1	1	0									z	z	z

z	z	z	
0	0	0	= READ
0	0	1	= WRIT
0	1	0	For Loadable Options
0	1	1	For Loadable Options
1	0	0	For Loadable Options
1	0	1	For Loadable Options
1	1	0	For Loadable Options
1	1	1	For Loadable Options

How X and Z Bits are Used in 24-Bit Nodes

In the 24-bit CPUs, the three most significant x bits are used to incicate the type of DX function. The z bits which simply echo the three most significant x bits, may be ignored in the 24-bit nodes.

x	x	x	1	1	1	0	0	z	z	z
0	0	0	=	R→T				0	0	0
0	0	1	=	T→R				0	0	1
0	1	0	=	T→T				0	1	0
0	1	1	=	BLKM				0	1	1
1	0	0	=	FIN				1	0	0
1	0	1	=	FOUT				1	0	1
1	1	0	=	SRCH				1	1	0
1	1	1	=	STAT				1	1	1

x	x	x	1	1	1	0	1	z	z	z
0	0	0	=	AND				0	0	0
0	0	1	=	OR				0	0	1
0	1	0	=	CMPR				0	1	0
0	1	1	=	SENS				0	1	1
1	0	0	=	MBIT				1	0	0
1	0	1	=	COMP				1	0	1
1	1	0	=	XOR				1	1	0
1	1	1	=	BROT				1	1	1

x	x	x	1	1	1	1	0	z	z	z
0	0	0	=	READ				0	0	0
0	0	1	=	WRIT				0	0	1
0	1	0	}	For Loadable Options				0	1	0
0	1	1	}	For Loadable Options				0	1	1
1	0	0	}	For Loadable Options				1	0	0
1	0	1	}	For Loadable Options				1	0	1
1	1	0	}	For Loadable Options				1	1	0
1	1	1	}	For Loadable Options				1	1	1

Opcodes for Standard DX Instructions

Opcode	Definition
1c	R->T instruction
3C	T->R instruction
5C	T->T instruction
7C	BLKM instruction
9C	FIN Instruction
BC	FOUT Instruction
DC	SRCH Instruction
FC	STAT Instruction
20	DIOH Instruction
1D	AND Instruction
3D	OR Instruction
5D	CMPR Instruction
7D	SENS Instruction
9D	MBIT Instruction
BD	COMP Instruction
DD	XOR Instruction
FD	BROT Instruction
1E	READ Instruction
3E	WRIT Instruction
7E	XMIT Instruction
9E	XMRD Instruction
51	IBKR
52	IBKW

Note: These opcodes are hard-coded in the appropriate system firmware, and they cannot be altered.

How the Y Bits are Utilized for DX Functions

The y bits in a database node holding DX function data contain a binary number that expresses the number of registers being transferred in the function. A 16-bit database node has 8 y bits. A 16-bit CPU is, therefore, machine limited to no more than 255 transfer registers per DX operation.A 24-bit database node has 13 y bits. A 24-bit CPU is, therefore, capable of reaching a theoretical machine limit of 8191 transfer registers per DX operation; practically, however, the greatest number of transfer registers allowed in a 24-bit DX operation is 999.

Opcode Defaults for Loadables

Overview

Various ladder logic instructions are available only in loadable software packages. When instructions are loaded to a controller, they are stored in RAM on page 0 in system memory. They are not resident on the EPROM. The loadable functions have the following opcodes.

Opcode	Definition
FF	HSBY instruction
SF	CALL, FNxx, or EARS instruction
1F	MBUS instruction
3F	PEER instruction
DE	DMTH instruction
BE	MATH or EARS instruction
FE	DRUM instruction
7F	ICMP instruction

How to Handle Opcode Operations

Note: No two instructions with the same opcode can coexist on a PLC.

The easiest way to stay out of trouble is to never employ two loadables with conflicting opcodes in your user logic. If you are using MODSOFT panel software, it allows you to change the opcodes for loadable instructions. The lodutil utility in the Modicon Custom Loadable Software package (SW-AP98-GDA) also allows you to change loadable opcodes.

WARNING

If you modify any loadables so that their opcodes are different from the ones shown in this chapter, you must use caution when porting user logic to or from your controller. The opcode conflicts that can result may hang up the target controller or cause the wrong function blocks to be executed in ladder logic.

Failure to follow this precaution can result in death, serious injury, or equipment damage.

Instructions

4

Parameter Assignment of Instuctions

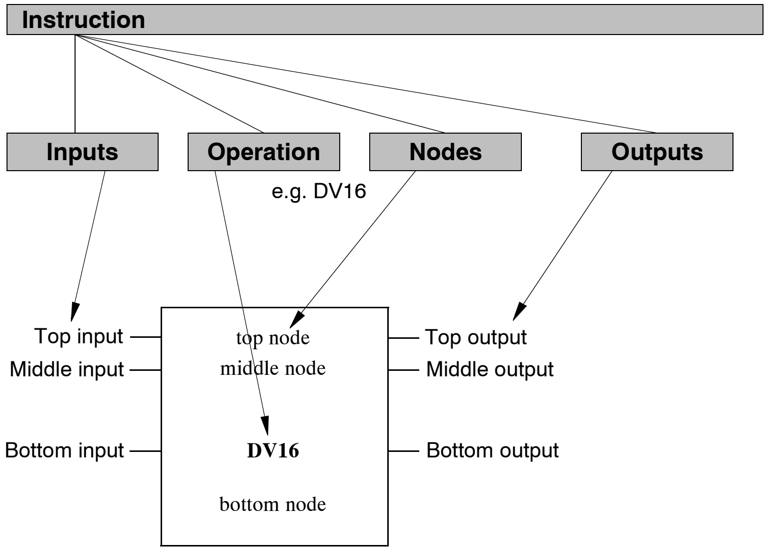
General

Programming for electrical controls involves a user who implements Operational Coded instructions in the form of visual objects organized in a recognizable ladder form. The program objects designed, at the user level, is converted to computer usable OP codes during the download process. the Op codes are decoded in the CPU and acted upon by the controllers firmware functions to implement the desired control.

Each instruction is composed of an operation, nodes required for the operation and in- and outputs.

Parameter Assignment

Parameter assignment with the instruction DV16 as an example:



Operation

The operation determines which functionality is to be executed by the instruction, e.g. shift register, conversion operations.

Nodes, In- and Outputs

The nodes and in- and outputs determines what the operation will be executed with.

Instruction Groups

5

At a Glance

Introduction

In this chapter you will find an overview of the instruction groups.

What's in this Chapter?

This chapter contains the following topics:

Instruction Groups

General

All instructions are attached to one of the following groups.

- ASCII Functions (See *ASCII Functions, p. 43*)
- Counters/Timers (See *Counters and Timers Instructions, p. 44*)
- Fast I/O Instructions (See *Fast I/O Instructions, p. 45*)
- Loadable DX (See *Loadable DX, p. 46*)
- Math (See *Math Instructions, p. 47*)
- Matrix (See *Matrix Instructions, p. 49*)
- Miscellaneous (See *Miscellaneous, p. 50*)
- Move (See *Move Instructions, p. 51*)
- Skips/Specials (See *Skips/Specials, p. 52*)
- Special (See *Special Instructions, p. 53*)
- Coils, Contacts and Interconnects (See *Coils, Contacts and Interconnects, p. 54*)

ASCII Functions

ASCII Functions

This group provides the following instructions.

Instruction	Meaning	Available at PLC family			
		Quantum	Compact	Momentum	Atrium
READ	Read ASCII messages	yes	no	no	no
WRIT	Write ASCII messages	yes	no	no	no

PLCs that support ASCII messaging use instructions called READ and WRIT to handle the sending of messages to display devices and the receiving of messages from input devices. These instructions provide the routines necessary for communication between the ASCII message table in the PLC's system memory and an interface module at the Remote I/O drops.
For further information, see *Formatting Messages for ASCII READ/WRIT Operations, p. 91*.

Counters and Timers Instructions

Counters and Timers Instructions

The table shows the counters and timers instructions.

Instruction	Meaning	Available at PLC family			
		Quantum	**Compact**	**Momentum**	**Atrium**
UCTR	Counts up from 0 to a preset value	yes	yes	yes	yes
DCTR	Counts down from a preset value to 0	yes	yes	yes	yes
T1.0	Timer that increments in seconds	yes	yes	yes	yes
T0.1	Timer that increments in tenths of a second	yes	yes	yes	yes
T.01	Timer that increments in hundredths of a second	yes	yes	yes	yes
T1MS	Timer that increments in one millisecond	yes (See note.)	yes	yes	yes

Note: The T1MS instruction is available only on the B984-102, the Micro 311, 411, 512, and 612, and the Quantum 424 02.

Fast I/O Instructions

Fast I/O Instructions

The following instructions are designed for a variety of functions known generally as fast I/O updating.

Instruction	Meaning	Available at PLC family			
		Quantum	**Compact**	**Momentum**	**Atrium**
BMDI	Block move with interrupts disabled	yes	yes	no	yes
ID	Disable interrupt	yes	yes	no	yes
IE	Enable interrupt	yes	yes	no	yes
IMIO	Immediate I/O instruction	yes	yes	no	yes
IMOD	Interrupt module instruction	yes	no	no	yes
ITMR	Interval timer interrupt	no	yes	no	yes

For more information, see *Interrupt Handling, p. 105*.

Note: The fast I/O instructions are only available after configuring a CPU without extension.

Loadable DX

Loadable DX

This group provides the following instructions.

Instruction	Meaning	Available at PLC family			
		Quantum	Compact	Momentum	Atrium
CHS	Hot standby (Quantum)	yes	no	no	no
DRUM	DRUM sequenzer	yes	yes	no	yes
ESI	Support of the ESI module 140 ESI 062 10	yes	no	no	no
EUCA	Engineering unit conversion and alarms	yes	yes	no	yes
HLTH	History and status matrices	yes	yes	no	yes
ICMP	Input comparison	yes	yes	no	yes
MAP3	MAP 3 Transaction	no	no	no	no
MBUS	MBUS Transaction	no	no	no	no
MRTM	Multi-register transfer module	yes	yes	no	yes
NOL	Transfer to/from the NOL Module	yes	no	no	no
PEER	PEER Transaction	no	no	no	no
XMIT	RS 232 Master Mode	yes	yes	yes	no

For more information, see *Installation of DX Loadables, p. 109*.

Math Instructions

Math Instructions

Two groups of instructions that support basic math operations are available. The first group comprises four integer-based instructions: ADD, SUB, MUL and DIV.

The second group contains five comparable instructions, AD16, SU16, TEST, MU16 and DV16, that support signed and unsigned 16-bit math calculations and comparisons.

Three additional instructions, ITOF, FTOI and BCD, are provided to convert the formats of numerical values (from integer to floating point, floating point to integer, binary to BCD and BCD to binary). Conversion operations are usful in expanded math.

Integer Based Instructions

This part of the group provides the following instructions.

Instruction	Meaning	Available at PLC family			
		Quantum	Compact	Momentum	Atrium
ADD	Addition	yes	yes	yes	yes
DIV	Division	yes	yes	yes	yes
MUL	Multiplication	yes	yes	yes	yes
SUB	Subtraction	yes	yes	yes	yes

Comparable Instructions

This part of the group provides the following instructions.

Instruction	Meaning	Available at PLC family			
		Quantum	Compact	Momentum	Atrium
AD16	Add 16 bit	yes	yes	yes	yes
DV16	Divide 16 bit	yes	yes	yes	yes
MU16	Multiply 16 bit	yes	yes	yes	yes
SU16	Subtract 16 bit	yes	yes	yes	yes
TEST	Test of 2 values	yes	yes	yes	yes

Format Conversion

This part of the group provides the following instructions.

Instruction	Meaning	Available at PLC family			
		Quantum	Compact	Momentum	Atrium
BCD	Conversion from binary to binary code or binary code to binary	yes	yes	yes	yes
FTOI	Conversion from floating point to integer	yes	yes	yes	yes
ITOF	Conversion from integer to floating point	yes	yes	yes	yes

Matrix Instructions

Matrix Instructions

A matrix is a sequence of data bits formed by consecutive 16-bit words or registers derived from tables. DX matrix functions operate on bit patterns within tables.

Just as with move instructions, the minimum table length is 1 and the maximum table length depends on the type of instruction you use and on the size of the CPU (24-bit) in your PLC.

Groups of 16 discretes can also be placed in tables. The reference number used is the first discrete in the group, and the other 15 are implied. The number of the first discrete must be of the first of 16 type 000001, 100001, 000017, 100017, 000033, 100033, ... , etc..

This group provides the following instructions.

Instruction	Meaning	Available at PLC family			
		Quantum	Compact	Momentum	Atrium
AND	Logical AND	yes	yes	yes	yes
BROT	Bit rotate	yes	yes	yes	yes
CMPR	Compare register	yes	yes	yes	yes
COMP	Complement a matrix	yes	yes	yes	yes
MBIT	Modify bit	yes	yes	yes	yes
NBIT	Bit control	yes	yes	no	yes
NCBT	Normally open bit	yes	yes	no	yes
NOBT	Normally closed bit	yes	yes	no	yes
OR	Logical OR	yes	yes	yes	yes
RBIT	Reset bit	yes	yes	no	yes
SBIT	Set bit	yes	yes	no	yes
SENS	Sense	yes	yes	yes	yes
XOR	Exclusive OR	yes	yes	yes	yes

Miscellaneous

Miscellaneous

This group provides the following instructions.

Instruction	Meaning	Available at PLC family			
		Quantum	Compact	Momentum	Atrium
CKSM	Check sum	yes	yes	yes	yes
DLOG	Data Logging for PCMCIA Read/Write Support	no	yes	no	no
EMTH	Extended Math Functions	yes	yes	yes	yes
LOAD	Load flash	yes (CPU 434 12/ 534 14 only)	yes	yes (CCC 960 x0/ 980 x0 only)	no
MSTR	Master	yes	yes	yes	yes
SAVE	Save flash	yes (CPU 434 12/ 534 14 only)	yes	yes (CCC 960 x0/ 980 x0 only)	no
SCIF	Sequential control interfaces	yes	yes	no	yes
XMRD	Extended memory read	yes	no	no	yes
XMWT	Extended memory write	yes	no	no	yes

Move Instructions

Move Instructions

This group provides the following instructions.

Instruction	Meaning	Available at PLC family			
		Quantum	Compact	Momentum	Atrium
BLKM	Block move	yes	yes	yes	yes
BLKT	Table to block move	yes	yes	yes	yes
FIN	First in	yes	yes	yes	yes
FOUT	First out	yes	yes	yes	yes
IBKR	Indirect block read	yes	yes	no	yes
IBKW	Indirect block write	yes	yes	no	yes
R → T	Register to tabel move	yes	yes	yes	yes
SRCH	Search table	yes	yes	yes	yes
T → R	Table to register move	yes	yes	yes	yes
T → T	Table to table move	yes	yes	yes	yes
TBLK	Table to block move	yes	yes	yes	yes

Skips/Specials

Skips/Specials

This group provides the following instructions.

Instruction	Meaning	Available at PLC family			
		Quantum	Compact	Momentum	Atrium
JSR	Jump to subroutine	yes	yes	yes	yes
LAB	Label for a subroutine	yes	yes	yes	yes
RET	Return from a subroutine	yes	yes	yes	yes
SKPC	Skip (constant)	yes	yes	yes	yes
SKPR	Skip (register)	yes	yes	yes	yes

The SKP instruction is a standard instruction in all PLCs. It should be used with caution.

DANGER

Inputs and outputs that normally effect control may be unintentionally skipped (or not skipped).

SKP is a dangerous instruction that should be used carefully. If inputs and outputs that normally effect control are unintentionally skipped (or not skipped), the result can create hazardous conditions for personnel and application equipment.

Failure to follow this precaution will result in death, serious injury, or equipment damage.

Special Instructions

Special Instructions

These instructions are used in special situations to measure statistical events on the overall logic system or create special loop control situations.
This group provides the following instructions.

Instruction	Meaning	Available at PLC family			
		Quantum	**Compact**	**Momentum**	**Atrium**
DIOH	Distributed I/O health	yes	no	no	yes
PCFL	Process control function library	yes	yes	no	yes
PID2	Proportional integral derivative	yes	yes	yes	yes
STAT	Status	yes	yes	yes	yes

Coils, Contacts and Interconnects

Coils, Contacts and Interconnects

Coils, contacts, and interconnects are available at all PLC families.

- normal coil
- memory-retentive, or latched, coil
- normally open (N.O.) contact
- normally closed (N.C.) contact
- positive transitional (P.T.) contact
- negative transitional (N.T.) contact
- horizontal short
- vertical short

Equation Networks

6

At a Glance

Overview

Equation network is a departure from standard ladder logic. Instead of using a two- or three-high function block configuration, this instruction takes a ladder logic network and uses it as an editor where you can compose a complex equation using algebraic notation. It allows you to use standard math operators such as +, -, *, /, as well as conditional and logical expressions. It also lets you specify variables and constants as necessary and group expressions in nested layers of parentheses. The power of an equation network is its ability to deal with complexity in a clear and efficient way. An equation composed in a single equation network might require many networks of standard ladder logic to produce the same result. An equation network can also be read and understood by other users without the need for detailed annotation, as is often required when standard ladder logic is used for complex calculations.

What's in this Chapter?

This chapter contains the following topics:

Equation Network Structure

Overview

An equation network provides an easy way to program complex math functions, with values stored in register locations. Equations in an equation network are presented in a regular, left-to-right format, technically known as "infix" notation. You program equation networks and set its enable contact and output coil(s) in the Equation Network Editor.
Equation networks were introduced in Quantum Rev. 2 controllers; not all controllers support equation networks. The easiest way to see if your controller supports equation networks is by trying to create a new one—if your controller doesn't support it, the equation network option on the right-click Insert menu won't be available.

Creating an Equation Network

In the Network Navigation panel:

Step	Action
1	Select the network where you want to insert the equation network.
2	From the right click menu in the logic editor select **Insert** → **Equation Network**. An equation network occupies a whole network, regardless of the contents of the equation network.

Using the Equation Network

In the Properties panel:

Step	Action
1	Select an input type from the **Input Type** drop-down list.
2	Enter the input offset in the **Input Offset** property
3	Set the register address for the output coils. You can enter either the direct address (in X:Y numeric format) or a symbolic address. You can also insert addresses from the Symbols list panel, Used Register Address table and the Descriptor Summary.See below for coil descriptions.
4	Enter an equation into the network by selecting the ellipsis box in the **Equation** property or double-clicking anywhere in the Equation Editor Network. The Equation Editor dialog appears.

Coil Descriptions

Coil descriptions:

Coil	Description
Solved OK	Solved OK is set when the equation is being solved without errors.
< Coil	Result<0 is set when the equation result is less than zero.
= Coil	Result=0 is set when the equation result is equal to zero.
> Coil	Result>0 is set when the equation result is greater than zero.
Error Coil	Error is set when errors have occurred while solving the equation. While online, if the Error coil receives power, an error message will appear under the coil describing the error ((See *Error Coil Messages, p. 57*)).

Note: If you don't want to use a particular output coil, leave the address for that coil blank (or erase one already typed in). That coil will not be included in the equation network.

Error Coil Messages

Error Message	Meaning
Invalid Op.	An internal error generated by the math coprocessor.
Overflow	A value is too large to be represented in its specified data type.
Underflow	A number is too small to be represented in FP format (for floating point data only)
Divide by 0	The variable, constant, or result of a function directly to the right of a / operator has a value of zero.
Invalid operation with boolean data	Occurs when a boolean value is entered in an argument to a function.

Setting up an Enable Contact

An equation network's enable contact, when set, activates the equation network. If an enable contact passes current, the equation network will be solved. You change settings for the enable contact in the Enable Editor display.

To select a type for the enable contact, select the symbol of the enable contact that corresponds with your chosen type. An enable contact can be a normally-open contact, normally-closed contact, horizontal short, or a horizontal open.

To select a register address for the enable contact, in the Enable Contact address field, type the direct address (in X:Y numeric format) or symbolic address for the enable contact coil. This field is only available if the enable contact type is a normally-open or normally closed contact.

Equation Network Content

The content of the equation network is in the form:
`result := algebraic expression`
where the

result is	a variable contained in 1 or 2 4x registers. It may be a signed or unsigned 16-bit short integer, a signed or unsigned 32-bit long integer, or a floating point number.
algebraic expression is	a syntactically correct construction of variable and/or constant data, standard algebraic operators, and/or functions. Parentheses can be used to define the order in which the expression is evaluated and indicate arguments to functions within the expression.

Equation Network Size

An equation network can contain a maximum of 81 words, which are used according to the following rules:

Each...	Consumes....
enabling input	1 word
normally open or normally closed contact	1 word
horizontal short used as input	no words
output coil	1 word
16-bit register and/or discrete reference	1 word
operator in the equation window	1 word
function in the equation window	1 word
short integer	1 word
floating point or long constant	2 words
open/closed parenthetical pair	2 words

Mathematical Equations in Equation Networks

Equation Format

Equation elements appear in specific formats. Operations and functions each have their own format. Also, for each value, you must specify what kind of value it is (register address, constant or symbol) and its data type (signed integer, unsigned integer, etc.).

Equation Values and Data Types

Each value can refer to a constant, register address or symbol. The Equation Network Editor determines which data type the value is, based on the following format.

Format	Meaning	Example
Default (no # sign or single quotes	Register address	40001
Prefixed by #	Constant	#123
Enclosed in single quotes	Symbol	‘HEIGHT’

The actual data type of a value is determined by its suffix, as shown in the following table:

Suffix	Data Type	Applies to
B	Boolean (binary)	Constants, 1x, or 0x
U	16-bit unsigned short integer	Constants, 3x, or 4x
S	Signed short integer	Constants, 3x, or 4x
L	32-bit signed long integer	Constants, 3x, or 4x
UL	32-bit unsigned long integer	Constants, 3x, or 4x
F	32-bit floating point number	Constants, 3x, or 4x

Typically, you’d first indicate the register address where the calculated result is to be stored, followed by an equal sign (the "assignment operator"), followed by the calculation itself. For example:

```
40001 = 40002U + COS(40003UL) * #+1.35E-4F / ’HEIGHT’L
```

- 40002U is an address of a 16-bit unsigned integer.
- COS(40003UL) calculates the cosine of a long (32-bit) unsigned integer value stored at address 40003.
- #+1.35E-4F is the floating point value of 0.000145, given in exponential notation.
- ’HEIGHT’L is a symbol of the name HEIGHT, representing the address of a long (32-bit) signed integer.
- 40001 = indicates that the result of the calculation is to be stored in register address 40001 as a 16-bit signed integer.

Everything to the right of the assignment operator also constitutes an expression. An expression is any part of an equation that can be evaluated to a single value. This can be a single constant or register address, or a complete mathematical operation. For example, #35 is an expression, as are LOG(#10) and 40002U + COS(40003UL). Complex expressions can contain other expressions within them, as in #3 * (40002U + COS(40003UL)). For the most part, any operator or function can be performed on any expression, no matter how complex.

Note: It is good programming practice to enclose all expressions in parentheses, even when they’re not actually needed. This makes the equation easier to read and ensures that operations in an equation are solved in the correct order.

Variable Data

Variable data within an equation network can be in 0x and 1x discrete references and in 3x and 4x registers.

Data Type	Variable Type	Words Consumed	Registers Consumed
Boolean	0x or 1x	One	N/A
Unsigned 16-bit variable	3x or 4x	One	One
Signed 16-bit variable	3x or 4x	One	One
Unsigned long (32-bit) variable	3x or 4x	One	Two
Signed long (32-bit) variable	3x or 4x	One	Two
Floating point variable	3x or 4x	One	Two

Note: When contiguous 3x or 4x registers are used for 32-bit long integers, the value still consumes only one word in the equation network.

Note: When 3x or 4x registers are used for a floating point number, the value requires one word for complete definition.

Entering Variable Data in an Equation Network

When entering a 0x or 1x references as a discrete variable in an equation network, the reference is assumed to be boolean, and you do not need to append the suffix B to the reference. Thus, the entires 000010 and 000010B are equivalent.
No other suffixes are legal with a 0x or 1x reference.
When you enter a 3x or 4x register in an equation network, the following rules apply:

If you enter a register . .	then . .
without a suffix,	it is assumed to represent a signed 16-bit integer variable. You do not need to append the suffix S to the reference. Thus the entries 400023 and 400023S are equivalent.
with the suffix U (e.g., 300004U),	you indicate that a single register containing an unsigned 16-bit integer variable is used.
with the suffix L,	you indicate that two contiguous registers containing a signed 32-bit long integer variable are used (e.g., 400012L implies that register 400013 is also used).
with the suffix UL,	you indicate that two contiguous registers containing an unsigned 32-bit long integer variable are used (e.g., 300006UL imples that register 300007 is also used).
with the suffix F,	you indicate that two contiguous registers containing a floating point variable are used (e.g., 400101F implies that register 400102 is also used).

Note: You cannot append a 3x or 4x register with the suffix B.

Constant Data

Constants can also be used to specify data in an equation network. Long (32-bit) constants and floating point constants always require two words. The least significant byte (LSB) is always in the first of the two words. Both words must have the same data type.

Data Type	Words Consumed	Valid Range of Values
Boolean	One	0, 1
Signed 16-bit constant	One	-32,768 ... +32,767
Unsigned 16-bit constant	One	0 ... 65,535
Signed long (32-bit) constant	Two	-2 x 109 ... +2 x 109
Unsigned long (32-bit) constant	Two	0 ... 4,294,967,295
Floating point constant	Two	8.43 x 1037 ≤ $\lvert x \rvert$ ≤ 3.402 x 1038

Entering Constant Data in an Equation Network

A constant is prefaced with a # sign and appended with a data type suffix data type suffix (See *Equation Values and Data Types, p. 60*). All constant values are in decimal format. Hexadecimal values are not allowed in Modsoft.
If you enter a constant in an equation network without a suffix, it is assumed to a signed short integer. For example, the entries #-3574 and #-3574S are equivalent.
A boolean constant must have the suffix B. The only two valid boolean constants are #0B and #1B. No other values are legal boolean constants.

Exponential Notation

Floating point numbers are normally specified in exponential notation, as in:
`+1.34E-4`
This represents 1.35 times 10 to the -4th power, or 1.35 times 0.0001. Thus, we would shift the decimal place four places to the left to get 0.000135. The "-4" part is called the exponent (note the preceding "E") and can be a positive or negative number.
In the Equation Network Editor, you must also indicate:

- That these numbers are constants; and
- Their data types. For example, integers or floating point numbers.

The default data type is unsigned 16-bit integer. So, since the above value is a fraction (and therefore must be a floating point number), it would have to appear as `#+1.35E-4F`.
With no data type suffix, numbers in exponential notation are assumed to be integers. For example, #+1.35E+2 represents the unsigned 16-bit integer value 135. Exponential notation is particularly useful for very large integers.

Mathematical Operations in Equation Networks

Mathematical Operations

The following table lists the mathematical operations you can include in your equation:

<table>
<tr><th>Type</th><th>Operator</th><th>Result</th></tr>
<tr><td>Assignment operator
The assignment operator = is used to assign a storage place for the results of the equation. All equations will use the assignment operator. The format is:

<code>ADDRESS = EXPRESSION</code>

Where ADDRESS is a valid register address and EXPRESSION is a valid value or expression assigned to the address.</td><td>=</td><td>Assignment</td></tr>
<tr><td rowspan="2">Unary operator
"Unary" means "single", so unary operators are used on only one value. The unary operator is placed just before the value or expression to which it is applied. For example, -(30002) returns -1 times the number stored at address 30002.</td><td>-</td><td>Negation. The result is -1 times the value.</td></tr>
<tr><td>~</td><td>Ones complement. This works on the binary representation of a value: all 1s are changed to 0s and vice versa.</td></tr>
<tr><td>Exponentiation operator
Takes values to a specified power. 40001**3 returns the (integer) value stored at 40001, taken to the third power.</td><td>**</td><td>Exponentiation</td></tr>
<tr><td rowspan="4">Arithmetic operator
These require two values, one before and one after the operator. These values can be any valid expression. For example, #4 * 40003 results in four multiplied by the value stored at address 40003.</td><td>*</td><td>Multiplication</td></tr>
<tr><td>/</td><td>Division</td></tr>
<tr><td>+</td><td>Addition</td></tr>
<tr><td>-</td><td>Subtraction</td></tr>
</table>

<table>
<tr><th>Type</th><th>Operator</th><th>Result</th></tr>
<tr><td rowspan="5">Bitwise operator
Bitwise operators work on binary (base 2) representations of values.

• In the case of AND, OR and XOR, the computer applies the operator to each digit in the two values: 010 XOR 011 (2 XOR 3 in decimal numbers) results in 001 (1 in decimal).
• In the case of shifting operators, the computer shifts all digits in the binary representation of the number the given number of places to the left or right. Digits on one side of the number are lost, and zeros fill in the blanks on the other side. For example, for 8-bit numbers, 77 << 2 means 01001101 shifted left two digits. The binary result is 00110100, or 52 decimal.</td><td>&</td><td>AND. The single bit result of an AND operation is only true (1) if both bits are set to 1.</td></tr>
<tr><td>|</td><td>OR. The single bit result of an OR operation is true (1) if either bit is set to 1. The result is false (0) only if both bits are set to 0.</td></tr>
<tr><td>^</td><td>XOR. Short for "Exclusive OR". The single bit result of an XOR operation is false (0) if both bits are the same, true (1) otherwise.</td></tr>
<tr><td><<</td><td>Left Shift. The result of 40001<<#2 is the binary representation of the number stored at 40001 shifted left two (#2) places. Zeros are added on the right to fill in the gap.</td></tr>
<tr><td>>></td><td>Right Shift. The result of 40001>>#2 is the binary representation of the number stored at 40001 shifted right two (#2) places. Zeros are added on the left to fill in the gap.</td></tr>
<tr><td rowspan="6">Relational operator
These operators describe a comparison between two values or expressions. The result is always true (1) or false (0). For example, #35 <= #42 evaluates to 1 (true). Relational operators are used in Conditional expressions.</td><td><</td><td>Less than.</td></tr>
<tr><td><=</td><td>Less than or equal to.</td></tr>
<tr><td>=</td><td>Equal to.</td></tr>
<tr><td><></td><td>Not equal.</td></tr>
<tr><td>=></td><td>Greater than or equal to.</td></tr>
<tr><td>></td><td>Greater than.</td></tr>
<tr><td>Conditional operator
See below for details.</td><td>?:</td><td>Used in conditional expression.</td></tr>
<tr><td>Parentheses
Used to set precedence in solving equations. To make sure certain operations are solved before others, enclose those operations in parentheses.</td><td>()</td><td></td></tr>
</table>

How an Equation Network Resolves an Equation

An equation network calculates its result in one of two ways, depending on the operator types used in the expression.

Single Expression

Evaluate a single expression and execute it by copying the derived value to the result register.

Conditional Expression

Evaluate the validity of the first of three arguments in a conditional expression and execute it by copying the value from either the second or third argument in the conditional expression to the result register.

If the expression being evaluated contains only some combination of unary, exponentiation, mathematical, and/or logical bitwise operators, it is treated as a single argument and is solved via single expression. For example, in the equation

```
400001 := (#16 ** #2 - #5) * #7
```

the square of 16 (256(minus 5 (251) is multiplied by 7, and the result (1,757) is copied to register 400001.

If you use one or more of the six relational operators shown in the previous table, you create the first of three arguments that comprise a conditional expression. The conditional operators must be used to create then/else arguments in the expression, and conditional expression is used to execute the result. For example, in the equation

```
400001 := 400002 >= #100 ? 300001 : 300002
```

the value in register 400002 is evaluated to see if it is greater than or equal to 100. This is the first argument in the conditional expression. If the value is greater than or equal to 100, the second argument is executed and the value in register 300001 is copied to register 400001. It is less than 100, the third argument is executed and the value in register 300002 is copied to register 400001.

Operator Precedence

In a string of data types and operators, the order or precedence in the expression determines the order in which operations will be evaluated. Review the following examples:

#	Equation	Comments
1	400001 := 300001F ** 300002F * 300003 + 300004 & 300005 > 300006 ? 300007 : 300008	The operations in the first argument of the conditional expression are evaluated from left to right in the order they appear. First, the value in register 300001 is raised to the power of the value in register 300002, then multiplied by the value in register 300003. That result is added to the value in register 300004, then logically ANDed with the value in register 300005 and compared with the value in register 300006. If the > comparison is true, the second argument in the conditional expression is executed, and the value in register 300007 is copied to register 400001. If the > comparison is false, the third argument in the conditional expression is executed and the value in register 300008 is copied to register 400001.
2	400001 := 300002U > 300003U & 300004U + 300005F * 300006F ** 300007U ? 300008 : 300009	Operator precedence forces the opposite effect on the first argument of the conditional expression. Here the first operation to be evaluated is the exponentiation of the value in register 300006 by the value in register 300007, followed by multiplication by the value in register 300005, then adition with the value in register 300004, then logically ANDing the result with the value in register 300003, and finally comparing teh result with the value in register 300002. If the > comparison is true, the second argument in the conditional expression is executed, and the value in register 300008 is copied to register 400001. If the > comparison is false, the third argument in the conditional expression is executed, and the value in register 300009 is copied to register 400001. When operators of equal precedence appear in an expression, they are generally evaluated in the order form from left to right and top to bottom in the equation network.

Using Parentheses in an Equation Network Expression

You can alter the order in which an expression is evaluated by enclosing portions of the expression in parentheses. Parenthetical portions of the expressions are evaluated before portions outside the parentheses. Notice how the following expressions are evaluated with and without parentheses:

<table>
<tr><th>#</th><th>Equation</th><th>Comments</th></tr>
<tr><td>3</td><td>400001 := 300001U < 300002U | 300004U & 300001U + 300003U ? 300004 : 300005</td><td>This expression is evaluated by the precedence 300001U < ((300002U | 300004U) & (300001U + 300003U)) ? 300005 : 300006 where the sum of the values in registers 300001 and 300003 is ANDed with the logical OR of the values in registers 300002 and 300004.</td></tr>
<tr><td>4</td><td>400001 := 300001U < (300002U | 300004U & 300001U) + 300003U ? 300004 : 300005</td><td>This expression is evaluted by ORing the values in registers 300002 and 300004, then ANDing the result with the value in register 300001, and finally adding the value in register 300003.</td></tr>
</table>

Nested Parentheses

When multiple levels of parenthetical data are nested in an expression, the most deeply nested parenthetical data is evaluated first. An equation network permits up to 10 nested levels of parentheses in an expression.

For example, the order in which the second expression in (See *Operator Precedence, p. 67*) is evaluated can be seen more clearly when parentheses are used.

300002U > (300003U & (300004U + (300005U * (300006F ** 300007F)))) ? 300008 : 300009

Entering Parentheses in an Equation Network

Equation network will echo back to you the expression as you enter it. It does not prevent you from entering additional levels of parentheses even when they may not be necessary to make the expression syntactically correct. For example, in the expression

((((300004U + 300005U)))) / 300006U

equation network maintains the four nested level of parentheses in the expression even when only one set of parentheses may be needed.

> **Note:** The expression must have an equal and balanced number of open and closed parentheses in order to compile properly. If it does not, a compiler error will be generated and the equation network will not function.

Each pair of open and closed parentheses consumes two words in the equation network.

Mathematical Functions in Equation Networks

Mathematical Functions

The following table lists the pre-defined math functions you can include in your equation. Each of these functions takes one argument enclosed in brackets following the function name. The argument can be any valid value or expression. For example, COS(#35+40001) returns the cosine of 35 plus the number stored at address 40001. In this table, X refers to a function's argument (as in "COS(X)").

Function	Description
ABS(S)	Absolute value of X (i.e. negative numbers become positive).
ARCCOS(X)	Arc cosine of X radians.
ARCSIN(X)	Arc sine of X radians.
ARCTAN(X)	Arc tangent of X radians.
COS(X)	Cosine of X radians.
COSD(X)	Cosine of X degrees.
EXP(X)	Calculates e (approximately 2.7182818) to the Xth power.
FIX(X)	Converts floating point number X to an integer.
FLOAT(X)	Converts integer X to a floating point number.
LN(X)	Natural (base e) logarithm of X.
LOG(X)	Common (base 10) logarithm of X.
SIN(X)	Sine of X radians.
SIND(X)	Sine of X degrees.
SQRT(X)	Square root of X.
TAN(X)	Tangent of X radians.
TAND(X)	Tangent of X degrees.

Entering Functions in an Equation Network

A function must be entered with its argument in the following form in the equation network expression:

function name (argument)

where the function name is one of those listed in the table above and the argument is entered in parentheses immediately after the function name. The argument may be entered as:

- one or more unary operations
- one or more exponential operations
- one or more multiplication/division operations
- oneo or more addition/subtraction operations
- one or more logical operations
- one or more relational operations
- any legal combination of the above operations

For example, if you want to calculate the absolute value of the sine of the number in FP register 400025 and place the result in FP register 400015, enter the following in the equation network:

400015F := ABS (SIN (400025F))

See *Mathematical Operations in Equation Networks, p. 64* for more details about these operations.

Limits on the Argument to a Function

The argument to a function in an equation network is resolved to a floating ponit (FP) number. The FP value must be in the following range, depending on the type of function.

Function	Argument	Range
ABS	FP value	-3.402823 x 1038 ... +3.402823 x 1038
ARCCOS	FP value	-1.00000 ... +1.00000
ARCSIN	FP value	-1.00000 ... +1.00000
ARCTAN	FP value	-3.402823 x 1038 ... +3.402823 x 1038
COS	FP value	-3.402823 x 1038 ... +3.402823 x 1038
COSD	FP value	-3.224671 x 104 ... +3.224671 x 104
EXP	FP value	-87.33655 ... +88.72284
FIX	FP value	-2.147484 x 109 ... +2.147484 x 109
FLOAT	FP value	-3.402823 x 1038 ... +3.402823 x 1038
LN	FP value	0 ... 3.402823 x 1038
LOG	FP value	0 ... 3.402823 x 1038
SIN	FP value	-3.402823 x 1038 ... +3.402823 x 1038
SIND	FP value	-1.724705 x 104 ... +1.724705 x 104
SQRT	FP value	0 ... 3.402823 x 1038
TAN	FP value	-3.402823 x 1038 ... +3.402823 x 1038, not p/ 2 x n (wher n is an integer value)
TAND	FP value	-1.351511 x 104 ... +1.351511 x 104, not 90 x n (where n is an integer value)

Data Conversions in an Equation Network

Mixed Data Types in an Equation Network

In an equation network, some combinations of operators will convert the value of an operand from 1 data type to another. The following set of rules applies to mixed data types in an equation network:

- All 16-bit signed and unsigned numbers are automatically promoted to 32 bits before an operation.
- In an operation between signed and unsigned numbers, the unsigned number is assumed to be signed without checking for overflow.
- An operation involving a boolean and any other data type uses the other data type and assigns a value of 1 or 0 to the boolean.
- An operation between floating point numbers and signed or unsigned numbers automatically promotes the long integer to floating point and assumes assigned number without checking for overflow.
- An operation involving a bitwise logical AND, OR, or XOR does not check data types and automatically assumes unsigned numbers.
- A bitwise logical AND, OR, or XOR operation with a boolean argument results in a 0 (false) or a 0xFFFFFFFF (true).
- The unary NOT ONE’s complement operation does not operate on floating point numbers and treats signed numbers as if they were unsigned.
- In a shift forward or shift back operation, the number by which the argument is being shifted is always treated as a positive integer between 0 ... 32. If the value of the by number > 32, it is automatically ANDed with 0x1f to make it < 32.
- Signed numbers are shifted arithmetically, and unsigned numbers are shifted logically.
- A floating point number that is shifted becomes useless, since its data type remains unchanged.
- Attempting to shift a boolean argument produces an error.
- The unary negation of an unsigned number produces that number’s 2’s complement.
- The unary negation of a signed or floating point number changes the sign of the number.
- The unary negation of a boolean operator results in a change of true/false state of the boolean.
- An absolute value operation does not change the data type of the result.
- Attempting to find the absolute value of a boolean argument produces an error.
- A floating point result is always produced by an EXP, LN, LOG, SQRT, SIN, COS, TAN, SIND, COSD, TAND, ARCSIN, ARCCOS, or ARCTAN function. If the original argument was not a floating point number, it will be promoted to one, assuming a signed number without checking for overflow. The exception is an original boolean argument, which will produce an error with any of these functions.
- A boolean + boolean operation is an OR operation.
- A boolean - boolean operation is an XOR operation.

- Boolean * boolean, boolean / boolean, and boolean ** boolean operations are AND operations.
- A boolean assignment (=) to a signed or unsigned number produces a signed or unsigned 0 or 1.
- A boolean assignment (=) to a floating point number produces a floating point 0.0 or 1.0.
- A long/short signed/unsigned number assignment (=) to a short unsigned number produces a result in the range 0 ... 65,535. Overflow is set if the result is > 65,535.
- A long/short signed/unsigned number assignment (=) to a short signed number produces a result in the range -32,768 ... 32,767. Overflow is set if the result is > 32,767 or < -32,768.
- A floating point number assignment (=) to a long/short signed/unsigned number will be truncated.
- A floating point number assignment (=) to a short unsigned number produces a result in the range 0 ... 65,535. Overflow is set if the result is > 65,535.
- A floating point number assignment (=) to a short signed number produces a result in the range -32,768 ... 32,767. Overflow is set if the result is > 32,767 or < -32,768.
- A floating point number assignment (=) to a long unsigned number produces a result in the range 0 ... 4,294,967,295. Overflow is set if the result is > 4,294,967,295.
- A floating point number assignment (=) to a long signed number produces a result in the range -2,147,483,648 ... 2,147,483,647. Overflow is set if the result is >2,147,483,647 or < -2,147,483,648.

Roundoff Differences in PLCs without a Math Coprocessor

Overview

Equation networks can be executed by Quantum PLCs like the 140 CPU 424 02 and 140 CPU 213 04, which have on-board math coprocessors, as well as by the 140 CPU 113 02 and 03 PLCs, which do not have math coprocessors. Quantum PLCs without math coprocessors use a 32-bit processing mechanism within the PLC itself to handle floating point calculations, and they can produce results that are less accurate than those produced by the 80-bit math coprocessor.

Differences in accuracy can be noticed starting in the sixth position to the right of the decimal point. For example, the 140 CPU 424 02 and 213 04 will calculate the equation

401010F = SIN(#45)

and produces the result 0.8509035, whereas the 140 CPU 113 02/03 will handle the same equation and produce the result 0.8509022.

For applications that require accuracy beyond the 5th decimal position, a Quantum PLC with a math coprocessor is recommended. Generally, if your application does not require this kind of accuracy, a PLC without a math coprocessor may be acceptable.

Another potential consideration is the effect of less accurate calculation on a truncated result. For example, a PLC with a math coprocessor will calculate the tangent of 225 degrees

401015F = TAND(#225)

as 1, whereas a PLC without a math coprocessor will produce the result 0.999991. If we were to assign the TAND operation to a non-floating point register, equation network will truncate the result so that

401040 = TAND(#225)

will produce a result of 1 when the math coprocessor is used but a result of 0 when the coprocessor is not used.

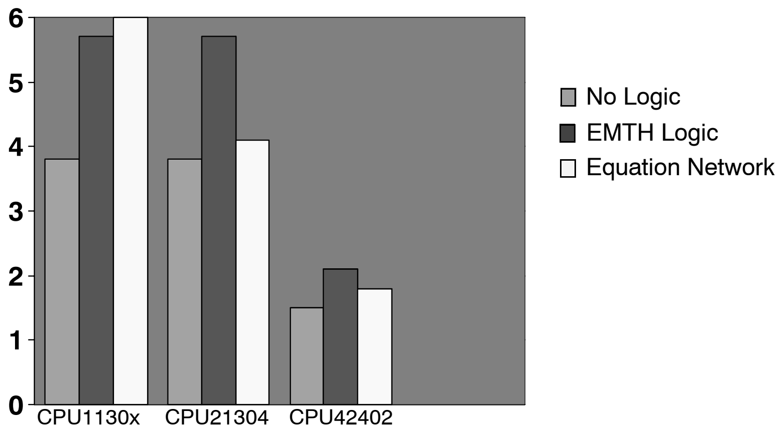
Benchmark Performance

Benchmark Performance

Benchmark tests were performed on 3 Quantum PLCs (CPU113, CPU213, and CPU424), solving the same equation with an equation network operation and EMTH ladder logic operations. The equation was
A = ((B*C) + D - E / SINE F)
where A, B, C, D, E, and F are either constants or registers.

> **Note:** This equation was the only ladder logic loaded to the Quantum PLCs for the benchmark tests.

The graph below shows the scan times for the 3 PLCs. Notice that EMTH performance on the CPU113 and CPU213 is identical; this is because EMTH does not utilize the math coprocessor available on the CPU213. Equation network performance, which does not use a math coprocessor when it is available, improves by 15% in the CPU213 over the CPU113.



> **Note:** The equation network approach provides a more accurate result than the interpolated math implemented in EMTH operations.

> **Note:** Equation network operations yield even better performance versus EMTH operations as the equations become more complex.

Closed Loop Control / Analog Values

7

At a Glance

Introduction

In this chapter you will find general information about configuring closed loop control and using analog values.

What's in this Chapter?

This chapter contains the following topics:

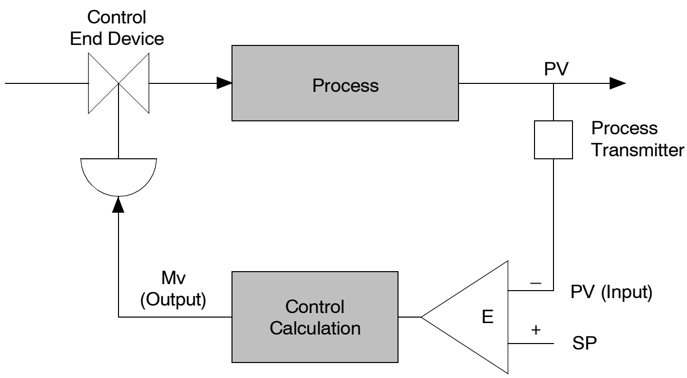
Closed Loop Control / Analog Values

General

An analog closed loop control system is one in which the deviation from an ideal process condition is measured, analyzed and adjusted in an attempt to obtain and maintain zero error in the process condition. Provided with the Enhanced Instruction Set is a proportional-integral-derivative function block called PID2, which allows you to establish closed loop (or negative feedback) control in ladder logic.

Definition of Set Point and Process Variable

The desired (zero error) control point, which you will define in the PID2 block, is called the set point (SP). The conditional measurement taken against SP is called the process variable (PV). The difference between the SP and the PV is the deviation or error (E). E is fed into a control calculation that produces a manipulated variable (Mv) used to adjust the process so that PV = SP (and, therefore, E = 0).



PCFL Subfunctions

General

The PCFL instruction gives you access to a library of process control functions utilizing analog values.
PCFL operations fall into three major categories.
- Advanced Calculations
- Signal Processing
- Regulatory Control

Advanced Calculations

Advanced calculations are used for general mathematical purposes and are not limited to process control applications. With advanced calculations, you can create custom signal processing algorithms, derive states of the controlled process, derive statistical measures of the process, etc.
Simple math routines have already been offered in the EMTH instruction. The calculation capability included in PCFL is a textual equation calculator for writing custom equations instead of programming a series of math operations one by one.

Signal Processing

Signal processing functions are used to manipulate process and derived process signals. They can do this in a variety of ways; they linearize, filter, delay and otherwise modify a signal. This category would include functions such as an Analog Input/Output, Limiters, Lead/Lag and Ramp generators.

Regulatory Control

Regulatory functions perform closed loop control in a variety of applications. Typically, this is a PID (proportional integral derivative) negative feedback control loop. The PID functions in PCFL offer varying degrees of functionality. Function PID has the same general functionality as the PID2 instruction but uses floating point math and represents some options differently. PID is beneficial in cases where PID2 is not suitable because of numerical concerns such as round-off.

Explanation of Formula Elements

Meaning of formula elements in the following formulas:

Formula elements	Meaning
Y	Manipulated variable output
YP	Proportional part of the calculation
YI	Integral part of the calculation
YD	Derivative part of the calculation
Bias	Constant added to input
BT	Bumpless transfer register
SP	Set point
KP	Proportional gain
Dt	Time since last solve
TI	Integral time constant
TD	Derivative time constant
TD1	Derivative time lag
XD	Error term, deviation
XD_1	Previous error term
X	Process input
X_1	Previous process input

General Equations

The following general equations are valid.

Equation	Condition /Requirement
$Y = YP + YI + YD + BIAS$	Integral bit ON
$Y = YP + YD + BIAS + BT$	Integral bit OFF
$Y_{high} \le Y \le Y_{low}$	High/low limits
with	
$YP, YI, YD = f(XD)$	
$XD = SP - X \pm (GRZ \times (1 - KGRZ))$	Gain reduction
$XD = SP - X$	Gain reduction zone not used

Proportional Calculations

The following equations are valid.

Equation	Condition /Requirement
$YP = KP \times XD$	Proportional bit ON
$YP = 0$	

Integral Calculation

The following equations are valid.

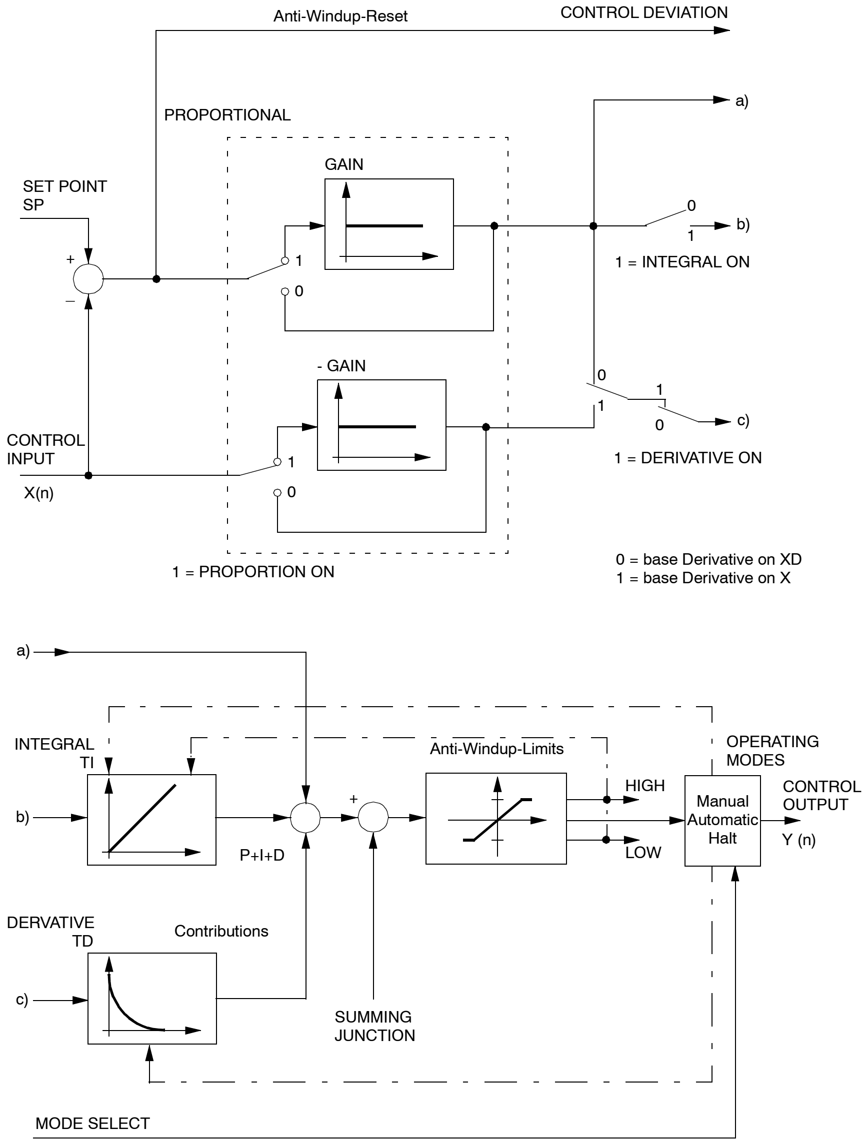
Equation	Condition /Requirement
$YI = YI + KP \times \frac{\Delta t}{TI} \times \frac{XD\_1 + XD}{2}$	Integral bit ON
$YI = 0$	

Derivative Calculation

The following equations are valid.

Equation	Condition /Requirement
$DXD = X\_1 - X$	Base derivative or PV
$DXD = XD - X\_1$	
$YD = \frac{(TD1 \times YD) + (TD \times KP \times DXD)}{\Delta t + TD1}$	Derivative bit ON
$YD = 0$	

Structure Diagram

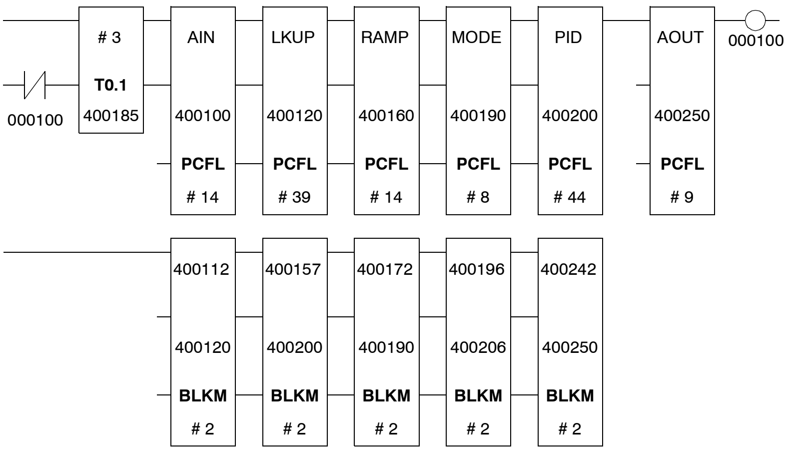


A PID Example

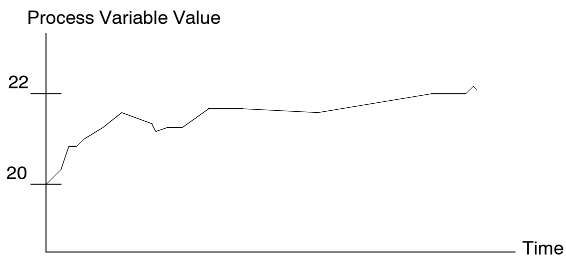
Description

This example illustrates how a typical PID loop could be configured using PCFL function PID. The calculation begins with the AIN function, which takes raw input simulated to cause the output to run between approximately 20 and 22 when the engineering unit scale is set to 0 ... 100.

LL984 Ladder Diagram

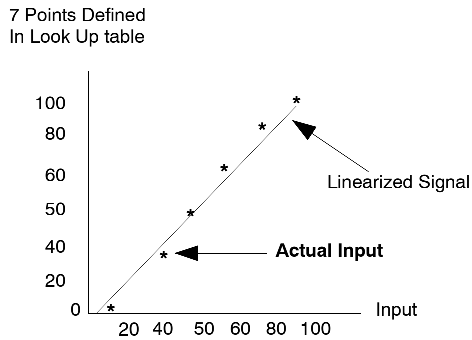


The process variable over time should look something like this.



Main PID Ladder Logic

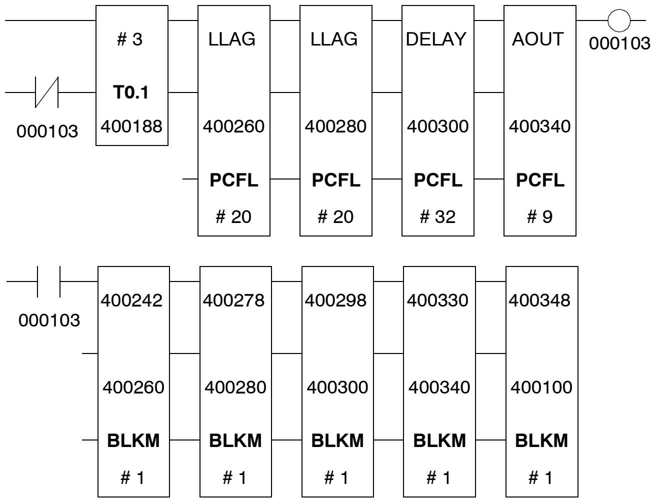
The AIN output is block moved to the LKUP function, which is used to scale the input signal. We do this because the input sensor is not likely to produce highly linear readings; the result is an ideal linear signal.



The look-up table output is block moved to the PID function. RAMP is used to control the rise (or fall) of the set point for the PID controller with regard to the rate of ramp and the solution interval. In this example, the set point is established in another logic section to simulate a remote setting. The MODE function is placed after the RAMP so that we can switch between the RAMP-generated set point or a manual value.

Simulated Process

The PID function is actually controlling the process simulated by this logic [value in 400100: **878**(Dec)].



The process simulator is comprised of two LLAG functions that act as a filter and input to a DELAY queue that is also a PCFL function block. This arrangement is the equivalent of a second-order process with dead time.

The solution intervals for the LLAG filters do not affect the process dynamics and were chosen to give fast updates. The solution interval for the DELAY queue is set at 1000 ms with a delay of 5 intervals,i.e. 5 s. The LLAG filters each have lead terms of 4 s and lag terms of 10 s. The gain for each is 1.0.

In process control terms the transfer function can be expressed as:

$$Gp(S) = \frac{(4S+1)(4S+1)e^{-5S}}{(10S+1)(10S+1)}$$

The AOUT function is used only to convert the simulated process output control value into a range of 0 ... 4 095, which simulates a field device. This integer signal is used as the process input in the first network.

PID Parameters

The PID controller is tuned to control this process at 20.0, using the Ziegler-Nichols tuning method. The resulting controller gain is 2.16, equivalent to a proportional band of 46.3%.

The integral time is set at 12.5 s/repeat (4.8 repeats/ min). The derivative time is initially 3 s, then reduced to 0.3 s to de-emphasize the derivative effect.
An AOUT function is used after the PID. It conditions the PID control output by scaling the signal back to an integer for use as the control value.

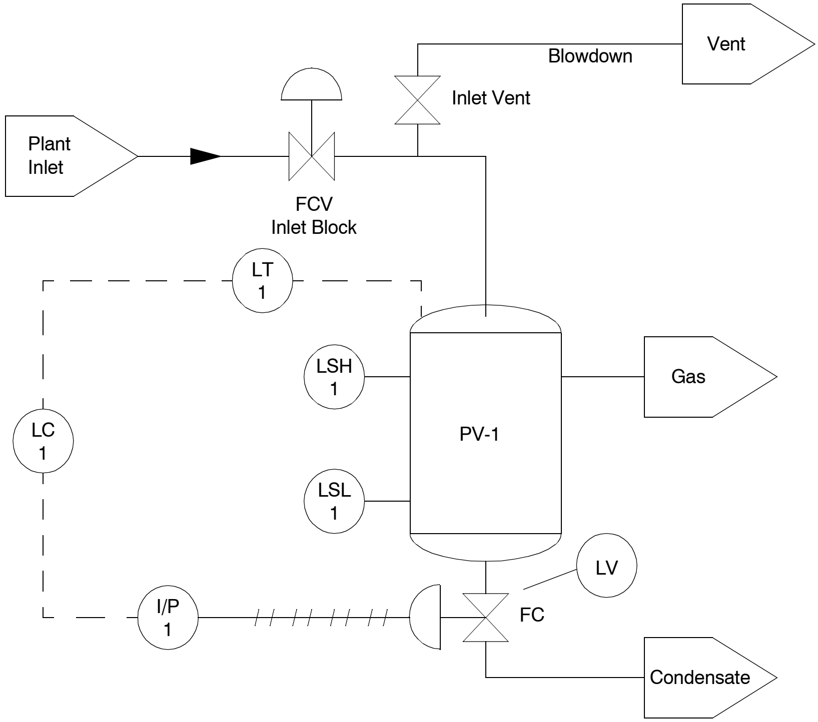
The entire control loop is preceded by a 0.1 s timer. The target solution interval for the entire loop is 1 s, and the full solve is 1 s. However, the nontime-dependent functions that are used (AIN, LKUP, MODE, and AOUT) do not need to be solved every scan. To reduce the scan time impact, these functions are scheduled to solve less frequently. The example has a loop solve every 3 s, reducing the average scan time dramatically.

Note: It is still important to be aware of the maximum scan impact. When programming other loops, you will not want all of the loops to solve on the same scan.

PID2 Level Control Example

Description

Here is a simplified P&I diagram for an inlet separator in a gas processing plant. There is a two-phase inlet stream: liquid and gas.



LT-1 4 ... 20 mA level transmitter

I/P-1 4 ... 20 mA current to pneumatic converter

LV-1 control valve, fail CLOSED

LSH-1 high level switch, normally closed

LSL-1 low level switch, normally open

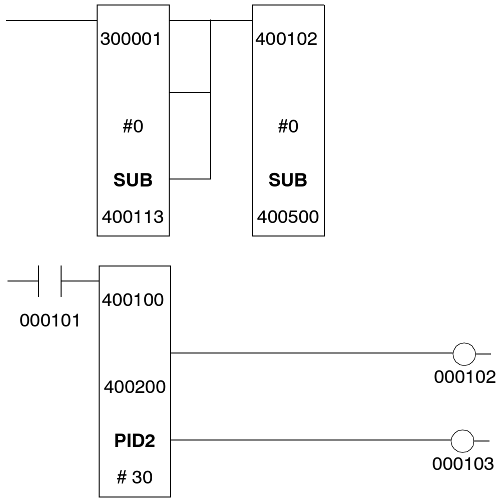
LC-1 level controller

I/P-1 Mv to control the flow into tank T-1

The liquid is dumped from the tank to maintain a constant level. The control objective is to maintain a constant level in the separator. The phases must be separated before processing; separation is the role of the inlet separator, PV-1. If the level controller, LC-1, fails to perform its job, the inlet separator could fill, causing liquids to get into the gas stream; this could severely damage devices such as gas compressors.

Ladder Logic Diagram

The level is controlled by device LC-1, a Quantum controller connected to an analog input module; I/P-1 is connected to an analog output module. We can implement the control loop with the following 984 ladder logic.



The first SUB block is used to move the analog input from LT-1 to the PID2 analog input register, 40113. The second SUB block is used to move the PID2 output Mv to the I/O mapped output I/P-1. Coil 00101 is used to change the loop from AUTO to MANUAL mode, if desired. For AUTO mode, it should be ON.

Register Content

Specify the set point in mm for input scaling (E.U.). The full input range will be 0 ... 4000 mm (for 0 ... 4095 raw analog). Specify the register content of the top node in the PID2 block as follows.

Register	Content Numeric	Content Meaning	Comments
400100		Scaled PV (mm)	PID2 writes this
400101	2000	Scaled SP (mm)	Set to 2000 mm (half full) initially
400102	0000	Loop output (0 ... 4095	PID2 writes this; keep it set to 0 to be safe
400103	3500	Alarm High Set Point (mm)	If the level rises above 3500 mm, coil 000102 goes ON
400104	1000	Alarm Low Set Point (mm)	If the level drops below 1000 mm, coil 000103 goes ON
400105	0100	PB (%)	The actual value depends on the process dynamics
400106	0500	Integral constant (5.00 repeats/min)	The actual value depends on the process dynamics
400107	0000	Rate time constant (per min)	Setting this to 0 turns off the derivative mode
400108	0000	Bias (0 ... 4095)	This is set to 0, since we have an integral term
400109	4095	High windup limit (0 ... 4095)	Normally set to the maximum
400110	0000	Low windup limit (0 ... 4095)	Normally set to the minimum
400111	4000	High engineering range (mm)	The scaled value of the process variable when the raw input is at 4095
400112	0000	Low engineering range (mm)	The scaled value of the process variable when the raw input is at 0
400113		Raw analog measure (0 ... 4095)	A copy of the input from the analog input module register (300001) copied by the first SUB
400114	0000	Offset to loop counter register	Zero disables this feature. Normally, this is not used
400115	0000	Max loops solved per scan	See register 400114

Register	Content Numeric	Content Meaning	Comments
400116	0102	Pointer to reset feedback	If you leave this as zero, the PID2 function automatically supplies a pointer to the loop output register. If the actual output (400500) could be changed from the value supplied by PID2, then this register should be set to 500 (400500) to calculate the integral properly
400117	4095	Output clamp high (0 ... 4095)	Normally set to maximum
400118	0000	Output clamp low (0 ... 4095)	Normally set to minimum
400119	0015	Rate Gain Limit Constant (2 ... 30)	Normally set to about 15. The actual value depends on how noisy the input signal is. Since we are not using derivative mode, this has no effect on PID2
400120	0000	Pointer to track input	Used only if the PRELOAD feature is used. If the PRELOAD is not used, this is normally zero

The values in the registers in the 400200 destination block are all set by the PID2 block.

Formatting Messages for ASCII READ/WRIT Operations

8

At a Glance

Introduction

In this chapter you will find general information about formatting messages for ASCII READ/WRIT operations.

What's in this Chapter?

This chapter contains the following topics:

Formatting Messages for ASCII READ/WRIT Operations

General

The ASCII messages used in the READ and WRIT instructions can be created via your panel software using the format specifiers described below. Format specifiers are character symbols that indicate:

- The ASCII characters used in the message
- Register content displayed in ASCII character format
- Register content displayed in hexadecimal format
- Register content displayed in integer format
- Subroutine calls to execute other message formats

Overview Format Specifiers

The following format specifiers can be used.

Specifier	Meaning
/	ASCII return (CR) and linefeed (LF)
" "	Enclosure for octal control code
‘ ´	Enclosure for ASCII text characters
X	Space indicator
( )	Repeat contents of the parentheses
I	Integer
L	Leading zeros
A	Alphanumeric
O	Octal
B	Binary
H	Hexadecimal

Format Specifiers

Format Specifier /

ASCII return (CR) and linefeed (LF)

Field width	None (defaults to 1)
Prefix	None (defaults to 1)
Input format	Outputs CR, LF; no ASCII characters accepted
Output format	Outputs CR, LF

Format Specifier " "

Enclosure for octal control code

Field width	Three digits enclosed in double quotes
Prefix	None
Input format	Accepts three octal control characters
Output format	Outputs three octal control characters

Format Specifier ‘ ´

Enclosure for ASCII text characters

Field width	1 ... 128 characters
Prefix	None (defaults to 1)
Input format	Inputs number of upper and/or lower case printable characters specified by the field width
Output format	Outputs number of upper and/or lower case printable characters specified by the field width

Format Specifier X

Space indicator, e.g., `14X` indicates 14 spaces left open from the point where the specifier occurs.

Field width	None (defaults to 1)
Prefix	1 ... 99 spaces
Input format	Inputs specified number of spaces
Output format	Outputs specified number of spaces

Format Specifier `( )`

Repeat contents of the parentheses, e.g., `2 (4X, I5)` says repeat 4X, I5 two times

Field width	None
Prefix	1 ... 255
Input format	Repeat format specifiers in parentheses the number of times specified by the prefix
Output format	Repeat format specifiers in parentheses the number of times specified by the prefix

Format Specifier `I`

Integer, e.g., `I5` specifies five integer characters

Field width	1 ... 8 characters
Prefix	1 ... 99
Input format	Accepts ASCII characters 0 ... 9. If the field width is not satisfied, the most significant characters in the field are padded with zeros
Output format	Outputs ASCII characters 0 ... 9. If the field width is not satisfied, the most significant characters in the field are padded with zeros. The overflow field consists of asterisks.

Format Specifier `L`

Leading zeros, e.g., `L5` specifies five leading zeros

Field width	1 ... 8 characters
Prefix	1 ... 99
Input format	Accepts ASCII characters 0 ... 9. If the field width is not satisfied, the most significant characters in the field are padded with zeros
Output format	Outputs ASCII characters 0 ... 9. If the field width is not satisfied, the most significant characters in the field are padded with zeros. The overflow field consists of asterisks.

Format Specifier `A`

Alphanumeric, e.g., `A27` specifies 27 alphanumeric characters, no suffix allowed

Field width	None (defaults to 1)
Prefix	1 ... 99
Input format	Accepts any 8-bit character except reserved delimiters such as CR, LF, ESC, BKSPC, DEL.
Output format	Outputs any 8-bit character

Format Specifier O

Octal, e.g., `O2` specifies two octal characters

Field width	1 ... 6 characters
Prefix	1 ... 99
Input format	Accepts ASCII characters 0 ... 7. If the field width is not satisfied, the most significant characters are padded with zeros.
Output format	Outputs ASCII characters 0 ... 7. If the field width is not satisfied, the most significant characters are padded with zeros. No overflow indicators.

Format Specifier B

Binary, e.g., `B4` specifies four binary characters

Field width	1 ... 16 characters
Prefix	1 ... 99
Input format	Accepts ASCII characters 0 and 1. If the field width is not satisfied, the most significant characters are padded with zeros.
Output format	Outputs ASCII characters 0 and 1. If the field width is not satisfied, the most significant characters are padded with zeros. No overflow indicators.

Format Specifier H

Hexadecimal, e.g., `H2` specifies two hex characters

Field width	1 ... 4 characters
Prefix	1 ... 99
Input format	Accepts ASCII characters 0 ... 9 and A ... F. If the field width is not satisfied, the most significant characters are padded with zeros.
Output format	Outputs ASCII characters 0 ... 9 and A ... F. If the field width is not satisfied, the most significant characters are padded with zeros. No overflow indicators.

Special Set-up Considerations for Control/Monitor Signals Format

General

To control and monitor the signals used in the messaging communication, specify code 1002 in the first register of the control block (the register displayed in the top node). Via this format, you can control the RTS and CTS lines on the port used for messaging.

Note: In this format, only the local port can be used for messaging, i.e., a parent PLC cannot monitor or control the signals on a child port. Therefore, the port number specified in the fifth implied node of the control block must always be 1.

The first three registers in the data block (the displayed register and the first and second implied registers in the middle node) have predetermined content.

Register	Content
Displayed	Stores the control mask word
First implied	Stores the control data word
Second implied	Stores the status word

These three data block registers are required for this format, and therefore the allowable range for the length value (specified in the bottom node) is 3 ... 255.

Control Mask Word

Usage of word:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Bit	Function
1	1 = port can be taken 0 = port cannot be taken
2 - 15	Not used
16	1 = control RTS 0 = do not control RTS

Control Data Word

Usage of word:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Bit	Function
1	1 = take port 0 = return port
2 - 15	Not used
16	1 = activate RTS 0 = deactivate RTS

Status Word

Usage of word:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Bit	Function
1	1 = port taken
2	1 = port ACTIVE as Modbus slave
3 - 13	Not used
14	1 = DSR ON
15	1 = CTS ON
16	1 = RTS ON

Coils, Contacts and Interconnects

9

At a Glance

Introduction

In this chapter you will find information about Coils, Contacts, and Interconnects, also called Shorts. Details of all the elements in the Ladder Logic Instruction Set appear in an alphabetical listing.

What's in this Chapter?

This chapter contains the following topics:

Coils

Definition of Coils

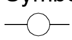
A coil is a discrete output that is turned ON and OFF by power flow in the logic program. A single coil is tied to a 0x reference in the PLC's state RAM. Because output values are updated in state RAM by the PLC, a coil may be used internally in the logic program or externally via the I/O map to a discrete output unit in the control system. When a coil is ON, it either passes power to a discrete output circuit or changes the state of an internal relay contact in state RAM.

There are two types of coils.
- A normal coil
- A memory-retentive, or latched, coil

Normal Coil

A normal coil is a discrete output shown as a 0x reference.
A normal coil is ON or OFF, depending on power flow in the program.
A ladder logic network can contain up to seven coils, no more than one per row.
When a coil is placed in a row, no other logic elements or instruction nodes can appear to the right of the coil's logic-solve position in the row. Coils are the only ladder logic elements that can be inserted in column 11 of a network.
To define a discrete reference for the coil, select it in the editor and click to open a dialog box called `Coil`.

Symbol



WARNING

Forcing of Coils

When a discrete input (1x) is disabled, signals from its associated input field device have no control over its ON/OFF state. When a discrete output (0x) is disabled, the PLC's logic scan has no control over the ON/OFF state of the output. When a discrete input or output has been disabled, you can change its current ON/OFF state with the Force command.
There is an important exception when you disable coils. Data move and data matrix instructions that use coils in their destination node recognize the current ON/OFF state of all coils in that node, whether they are disabled or not. If you are expecting a disabled coil to remain disabled in such an instruction, you may cause unexpected or undesirable effects in your application.
When a coil or relay contact has been disabled, you can change its state using the Force ON or Force OFF command. If a coil or relay is enabled, it cannot be forced.

Failure to follow this precaution can result in death, serious injury, or equipment damage.

Retentive Coil

If a retentive (latched) coil is energized when the PLC loses power, the coil will come back up in the same state for one scan when the PLC's power is restored.

To define a discrete reference for the coil, select it in the editor and click to open a dialog box called `Retentative coil (latch)`.

Symbol

—(L)—

Contacts

Definition of Contacts

Contacts are used to pass or inhibit power flow in a ladder logic program. They are discrete, i.e., each consumes one I/O point in ladder logic. A single contact can be tied to a 0x or 1x reference number in the PLC's state RAM, in which case each contact consumes one node in a ladder network.

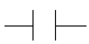
Four kinds of contacts are available.

- normally open (N.O.) contacts
- normally closed (N.C.) contacts
- positive transitional (P.T.) contacts
- negative transitional (N.T.) contacts

Contact Normally Open

A normally open (NO) contact passes power when it is ON.
To define a discrete reference for the NO contact, select it in the editor and click to open a dialog called `Normally open contact`.

Symbol

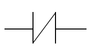


Contact Normally Closed

A normally closed (NC) contact passes power when it is OFF.

To define a discrete reference for the NC contact, double ckick on it in the ladder node to open a dialog called `Normally closed contact`.

Symbol



Contact Pos Trans

A positive transitional (PT) contact passes power for only one scan as it transitions from OFF to ON.

To define a discrete reference for the PT contact, select it in the editor and click to open a dialog called `Positive transition contact`.

Symbol

Contact Neg Trans

A negative transitional (NT) contact passes power for only one scan as it transitions from ON to OFF.

To define a discrete reference for the NT contact, select it in the editor and click to open a dialog called `Contact negative transition` .

Symbol

Interconnects (Shorts)

Definition of Interconnects (Shorts)

Shorts are simply straight-line connections between contacts and/or instructions in a ladder logic network. Shorts may be inserted horizontally or vertically in a network.

Two kinds of shorts are available.
- horizontal short
- vertical short

Horizontal Short

A short is a straight-line connection between contacts and/or nodes in an instruction through which power flow can be controlled.

A horizontal short is used to extend logic out across a row in a network without breaking the power flow. Each horizontal short consumes one node in the network, and uses a word of memory in the PLC.

Symbol

—

Vertical Short

A vertical short connects contacts or nodes in an instruction positioned one above the other in a column. Vertical shorts can also connect inputs or outputs in an instruction to create either-or conditions. When two contacts are connected by a vertical short, power is passed when one or both contacts receive power.

The vertical short is unique in two ways.
- It can coexist in a network node with another element or nodal value.
- It does not consume any PLC memory.

Symbol

|

Interrupt Handling

10

Interrupt Handling

Interrupt-related Performance

The interrupt-related instructions operate with minimum processing overhead. The performance of interrupt-related instructions is especially critical. Using a interval timer interrupt (ITMR) instruction adds about 6% to the scan time of the scheduled ladder logic, this increase does not include the time required to execute the interrupt handler subroutine associated with the interrupt.

Interrupt Latency Time

The following table shows the minimum and maximum interrupt latency times you can expect.

ITMR overhead	No work to do	60 ms/ms
Response time	Minimum	98 ms
	Maximum during logic solve and Modbus command reception	400 ms
Total overhead (not counting normal logic solve time)		155 ms

These latency times assume only one interrupt at a time.

Interrupt Priorities

The PLC uses the following rules to choose which interrupt handler to execute in the event that multiple interrupts are received simultaneously.

- An interrupt generated by an interrupt module has a higher priority than an interrupt generated by a timer.
- Interrupts from modules in lower slots of the local backplane have priority over interrupts from modules in the higher slots.

If the PLC is executing an interrupt handler subroutine when a higher priority interrupt is received, the current interrupt handler is completed before the new interrupt handler is begun.

Instructions that Cannot Be Used in an Interrupt Handler

The following (nonreenterant) ladder logic instructions cannot be used inside an interrupt handler subroutine.

- MSTR
- READ / WRIT
- PCFL / EMTH
- T1.0, T0.1, T.01, and T1MS timers (will not set error bit 2, timer results invalid)
- equation networks
- user loadables (will not set error bit 2)

If any of these instructions are placed in an interrupt handler, the subroutine will be aborted, the error output on the ITMR or IMOD instruction that generated the interrupt will go ON, and bit 2 in the status register will be set.

Interrupt with BMDI/ID/IE

Three interrupt mask/unmask control instructions are available to help protect data in both the normal (scheduled) ladder logic and the (unscheduled) interrupt handling subroutine logic. These are the Interrupt Disable (ID) instruction, the Interrupt Enable (IE) instruction, and the Block Move with Interrupts Disabled (BMDI) instruction.

An interrupt that is executed in the timeframe after an ID instruction has been solved and before the next IE instruction has been solved is buffered. The execution of a buffered interrupt takes place at the time the IE instruction is solved. If two or more interrupts of the same type occur between the ID ... IE solve, the mask interrupt overrun error bit is set, and the subroutine initiated by the interrupts is executed only one time

The BMDI instruction can be used to mask both a timer-generated and local I/O-generated interrupts, perform a single block data move, then unmask the interrupts. It allows for the exchange of a block of data either within the subroutine or at one or more places in the scheduled logic program.

BMDI instructions can be used to reduce the time between the disable and enable of interrupts. For example, BMDI instructions can be used to protect the data used by the interrupt handler when the data is updated or read by Modbus, Modbus Plus, Peer Cop or Distributed I/O (DIO).

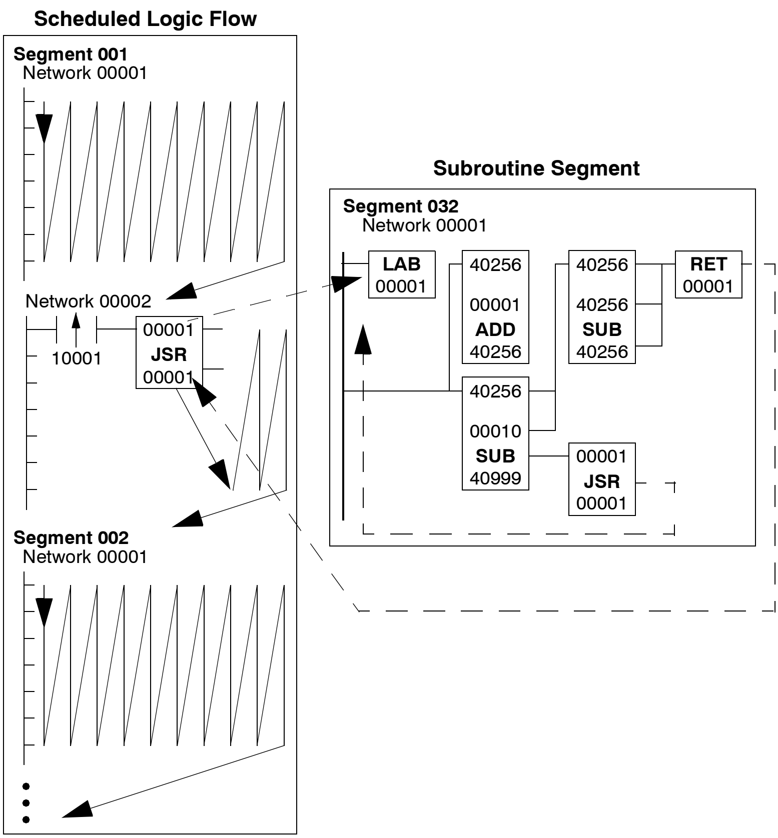
Subroutine Handling

11

Subroutine Handling

JSR / LAB Method

The example below shows a series of three user logic networks, the last of which is used for an up-counting subroutine. Segment 32 has been removed from the order-of-solve table in the segment scheduler.



When input 100001 to the JSR block in network 2 of segment 1 transitions from OFF to ON, the logic scan jumps to subroutine #1 in network 1 of segment 32.

The subroutine will internally loop on itself ten times, counted by the ADD block. The first nine loops end with the JSR block in the subroutine (network 1 of segment 32) sending the scan back to the LAB block. Upon completion of the tenth loop, the RET block sends the logic scan back to the scheduled logic at the JSR node in network 2 of segment 1.

Installation of DX Loadables

12

Installation of DX Loadables

How to install the DX Loadables

The DX loadable instructions are only available if you have installed them. With the installation of the Concept software, DX loadables are located on your hard disk. Now you have to unpack and install the loadables you want to use as follows.

Step	Action
1	With the menu command `Project → Configurator` you open the configurator.
2	With `Configure → Loadables...` you open the dialog box `Loadables`.
3	Press the command button `Unpack...` to open the standard Windows dialog box `Unpack Loadable File` where the multifile loadables (DX loadables) can be selected. Select the loadable file you need, click the button `OK` and it is inserted into the list box `Available:`.
4	Now press the command button `Install=>` to install the loadable selected in the list box `Available:`. The installed loadable will be displayed in the list box `Installed:`.
5	Press the command button `Edit...` to open the dialog box `Loadable Instruction Configuration`. Change the opcode if necessary or accept the default. You can assign an opcode to the loadable in the list box `Opcode` in order to enable user program access through this code. An opcode that is already assigned to a loadable, will be identified by a *. Click the button `OK`.
6	Click the button `OK` in the dialog box `Loadables`. Configuration loadables count is adjusted. The installed loadable is available for programming at the menu `Objects → List Instructions → DX Loadable`.

Instruction Descriptions (A to D)

II

At a Glance

Introduction

In this part instruction descriptions are arranged alphabetically from A to D.

What's in this Part?

This part contains the following chapters:

1X3X - Input Simulation

13

At A Glance

Introduction

This chapter describes the instruction 1X3X.

What's in this Chapter?

This chapter contains the following topics:

Short Description: 1X3X - Input Simulation

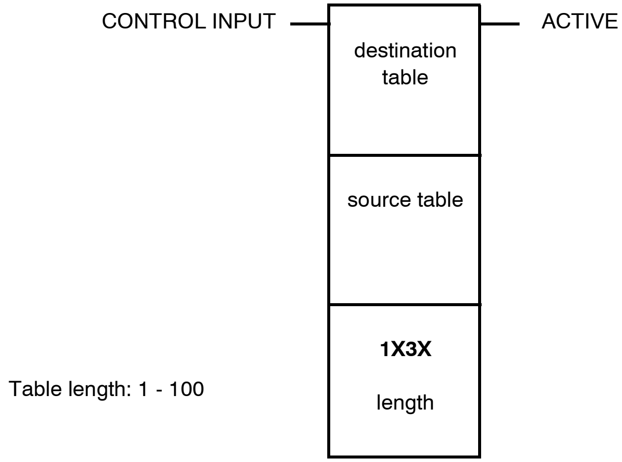
Function Description

The Input Simulation instruction provides a simple method to simulate 1xxxx and 3xxx input data values. This block is similar to a Block Move, the BLKM instruction. When the Control Input receives power, the source table is copied to the destination (input) table.

Representation: 1X3X - Input Simulation

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	
destination table (top node)	1x, 3x	INT	
source table (middle node	4x	INT	Contains source to be moved to destination
length (bottom node		INT	(Length: NNN if 3X) Length: 16* if 4x
Top output	0x	None	Passes power when top input receives power.

AD16: Ad 16 Bit

14

At a Glance

Introduction

This chapter describes the instruction AD16.

What's in this Chapter?

This chapter contains the following topics:

Short Description

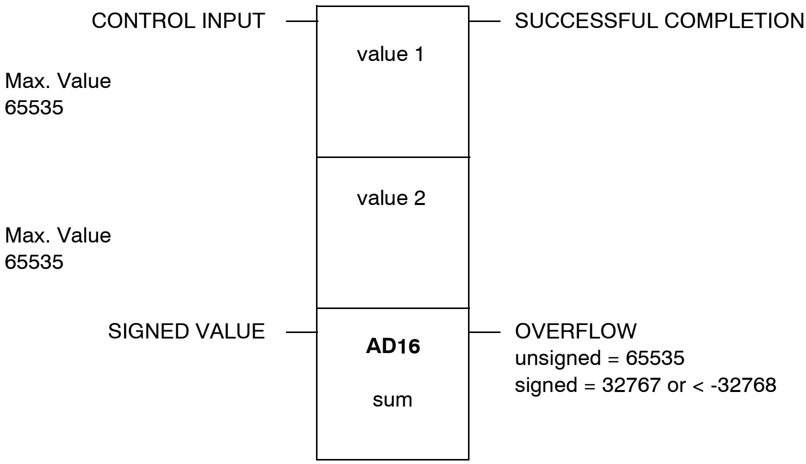
Function Description

The AD16 instruction performs signed or unsigned 16-bit addition on value 1 (its top node) and value 2 (its middle node), then posts the sum in a 4x holding register in the bottom node.

Representation: AD16 - 16-bit Addition

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = add value 1 and value 2
Bottom input	0x, 1x	None	ON = signed operation OFF = unsigned operation
value 1 (top node)	3x, 4x	INT, UINT	Addend, can be displayed explicitly as an integer (range 1 ... 65 535) or stored in a register
value 2 (middle node)	3x, 4x	INT, UINT	Addend, can be displayed explicitly as an integer (range 1 ... 65 535) or stored in a register
sum (bottom node)	4x	INT, UINT	Sum of 16 bit addition
Top output	0x	None	ON = successful completion of the operation
Bottom output	0x	None	ON = overflow in the sum:

ADD: Addition

15

At a Glance

Introduction

This chapter describes the instruction ADD.

What's in this Chapter?

This chapter contains the following topics:

Short Description

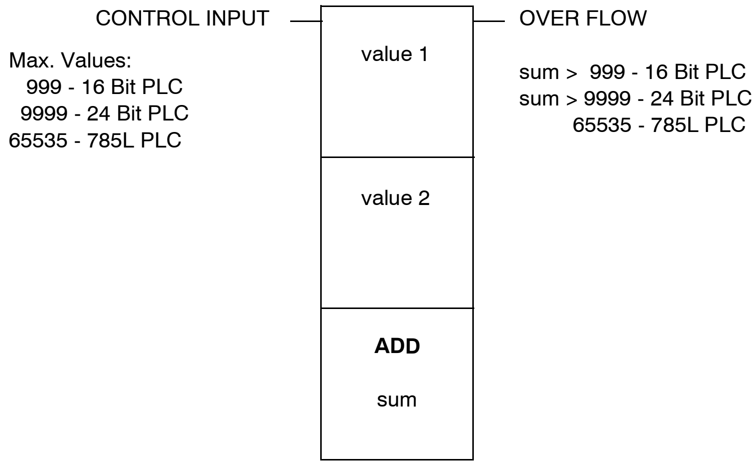
Function Description

The ADD instruction adds unsigned value 1 (its top node) to unsigned value 2 (its middle node) and stores the sum in a holding register in the bottom node.

Representation: ADD - Single Precision Add

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = add value 1 and value 2
value 1 (top node)	3x, 4x	INT, UINT	sum > 999 - 16 Bit PLC sum > 9999 - 24 Bit PLC 65535 - 785L PLC
value 2 (middle node)	3x, 4x	INT, UINT	sum > 999 - 16 Bit PLC sum > 9999 - 24 Bit PLC 65535 - 785L PLC
sum (bottom node)	4x	INT, UINT	Sum
Top output	0x	None	ON = overflow in the sum sum > 999 in 16-bit PLC sum > 9999 in 24-bit PLC 65535 in 785L PLC

AND: Logical And

16

At a Glance

Introduction

This chapter describes the instruction AND.

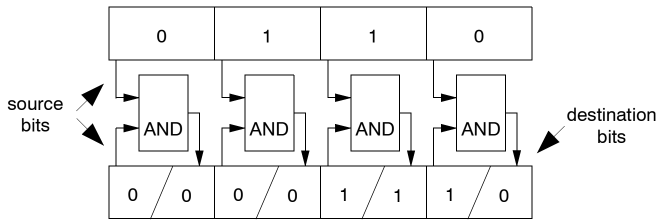
What's in this Chapter?

This chapter contains the following topics:

Short Description

Function Description

The AND instruction performs a Boolean AND operation on the bit patterns in the source and destination matrices.
The ANDed bit pattern is then posted in the destination matrix, overwriting its previous contents.



WARNING

Overriding of any disabled coils within the destination matrix without enabling them.

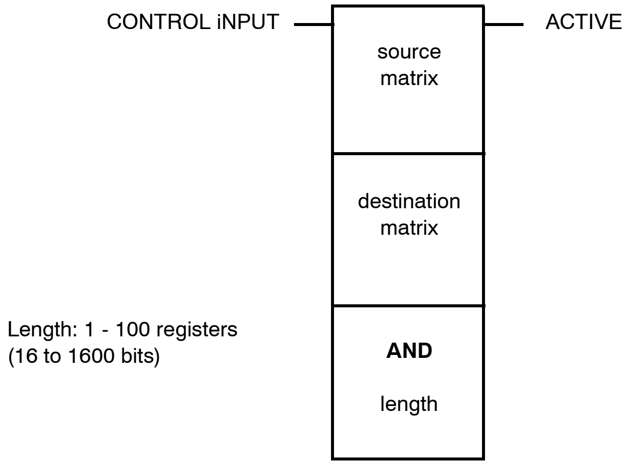
AND will override any disabled coils within the destination matrix without enabling them.This can cause personal injury if a coil has disabled an operation for maintenance or repair because the coil's state can be changed by the AND operation.

Failure to follow this precaution can result in death, serious injury, or equipment damage.

Representation: AND - Logical And

Symbol

Representation of the instruction



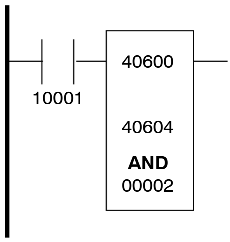
Parameter Description

Description of the instruction’s parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	Initiates AND
source matrix (top node)	0x, 1x, 3x, 4x	BOOL, WORD	First reference in the source matrix
destination matrix (middle node)	0x, 4x	BOOL, WORD	First reference in the destination matrix
length (bottom node)		INT, UINT	Matrix length; range 1 ... 100.
Top output	0x	None	Echoes state of the top input

An AND Example

When contact 10001 passes power, the *source matrix* formed by the bit pattern in registers 40600 and 40601 is ANDed with the *destination matrix* formed by the bit pattern in registers 40604 and 40605. The ANDed bits are then copied into registers 40604 and 40605, overwriting the previous bit pattern in the destination matrix.



source matrix
40600 = 1111111100000000   40601 = 1111111100000000

Original *destination matrix*
40604 = 1111111111111111   40605 = 0000000000000000

ANDed *destination matrix*
40604 = 1111111100000000   40605 = 0000000000000000

Note: If you want to retain the original destination bit pattern of registers 40604 and 40605, copy the information into another table using the BLKM instruction before performing the AND operation.

Parameter Description

Matrix Length (Bottom Node)	The integer entered in the bottom node specifies the matrix length, i.e. the number of registers or 16-bit words in the two matrices. The matrix length can be in the range 1 ... 100. A length of 2 indicates that 32 bits in each matrix will be ANDed.

BCD: Binary to Binary Code

17

At a Glance

Introduction

This chapter describes the instruction BCD.

What's in this Chapter?

This chapter contains the following topics:

Short Description

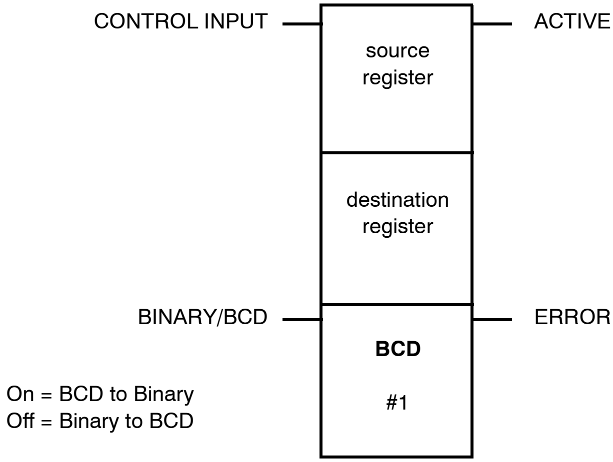
Function Description

The BCD instruction can be used to convert a binary value to a binary coded decimal (BCD) value or a BCD value to a binary value. The type of conversion to be performed is controlled by the state of the bottom input.

Representation: BCD - Binary Coded Decimal Conversion

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = enable conversion
Bottom input	0x, 1x	None	ON = BCD → binary conversion OFF = binary → BCD conversion
source register (top node)	3x, 4x	INT, UINT	Source register where the numerical value to be converted is stored
destination register (middle node)	4x	INT, UINT	Destination register where the converted numerical value is posted
#1 (bottom node)		INT, UINT	Constant value, can not be changed
Top output	0x	None	Echoes the state of the top input
Bottom output	0x	None	ON = error in the conversion operation

BLKM: Block Move

18

At a Glance

Introduction

This chapter describes the instruction BLKM.

What's in this Chapter?

This chapter contains the following topics:

Short Description

Function Description

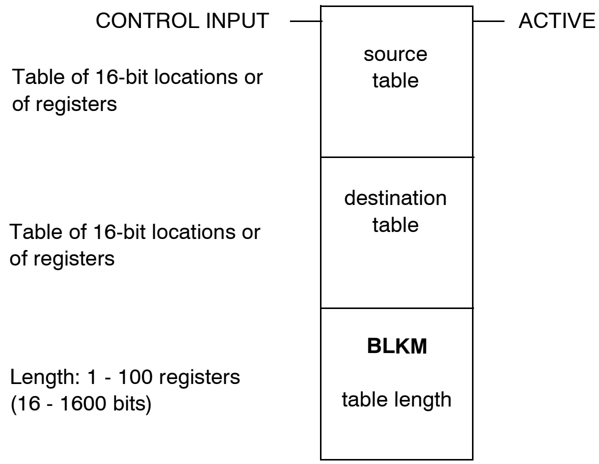
The BLKM (block move) instruction copies the entire contents of a source table to a destination table in one scan.

	WARNING
	Overriding of any disabled coils within a destination table without enabling them. BLKM will override any disabled coils within a destination table without enabling them. This can cause injury if a coil has been disabled for repair or maintenance because the coil's state can change as a result of the BLKM instruction. **Failure to follow this precaution can result in death, serious injury, or equipment damage.**

Representation: BLKM - Block Move

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = initiates block move
source table (top node)	0x, 1x, 3x, 4x	ANY_BIT	Source table that will have its contents copied in the block move
destination table (middle node)	0x, 4x	ANY_BIT	Destination table where the contents of the source table will be copied in the block move
table length (bottom node)		INT, UINT	Table size (number of registers or 16-bit words) for both the source and destination tables; they are of equal length. Range: 1 ... 100.
Top output	0x	None	Echoes the state of the top input

BLKT: Block to Table

19

At a Glance

Introduction

This chapter describes the instruction BLKT.

What's in this Chapter?

This chapter contains the following topics:

Short Description

Function Description

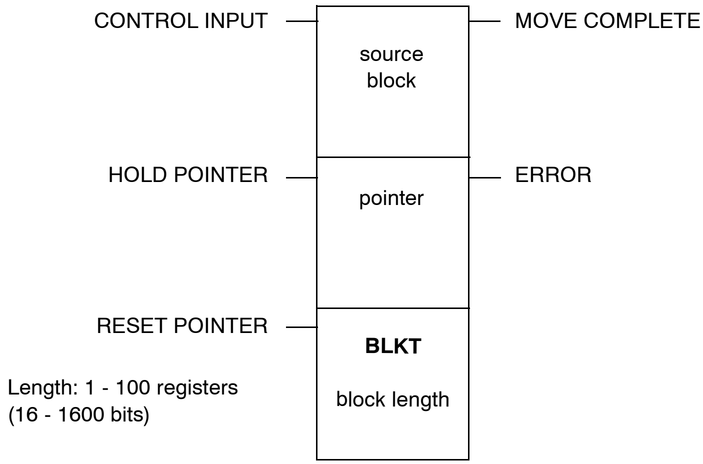
The BLKT (block-to-table) instruction combines the functions of R→T and BLKM in a single instruction. In one scan, it can copy data from a source block to a destination block in a table. The source block is of a fixed length. The block within the table is of the same length, but the overall length of the table is limited only by the number of registers in your system configuration.

	WARNING
⚠	**All the 4x registers in your PLC can be corrupted with data copied from the source block.** BLKT is a powerful instruction that can corrupt all the 4x registers in your PLC with data copied from the source block. You should use external logic in conjunction with the middle or bottom input to confine the value in the pointer to a safe range. **Failure to follow this precaution can result in death, serious injury, or equipment damage.**

Representation: BLKT - Block-to-Table Move

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = initiates the DX move
middle input	0x, 1x	None	ON = hold pointer
bottom input	0x, 1x	None	ON = reset pointer to zero
source block (top node)	4x	BYTE, WORD	First holding register in the block of contiguous registers whose content will be copied to a block of registers in the destination table.
pointer (middle node)	4x	BYTE, WORD	Pointer to the destination table
block length (bottom node)		INT, UINT	Block length (number of 4x registers) of the source block and of the destination block. Range: 1 ... 100.
Top output	0x	None	ON = operation successful
Middle output	0x	None	ON = error / move not possible

Parameter Description

Middle and Bottom Input

The middle and bottom input can be used to control the pointer so that source data is not copied into registers that are needed for other purposes in the logic program. When the middle input is ON, the value in the pointer register is frozen while the BLKT operation continues. This causes new data being copied to the destination to overwrite the block data copied on the previous scan.
When the bottom input is ON, the value in the pointer register is reset to zero. This causes the BLKT operation to copy source data into the first block of registers in the destination table.

Pointer (Middle Node)

The 4x register entered in the middle node is the pointer to the destination table. The first register in the destination table is the next contiguous register after the pointer, e.g. if the pointer register is 400107, then the first register in the destination table is 400108.

> **Note:** The destination table is segmented into a series of register blocks, each of which is the same length as the source block. Therefore, the size of the destination table is a multiple of the length of the source block, but its overall size is not specifically defined in the instruction. If left uncontrolled, the destination table could consume all the 4x registers available in the PLC configuration.

The value stored in the pointer register indicates where in the destination table the source data will begin to be copied. This value specifies the block number within the destination table.

BMDI: Block Move with Interrupts Disabled

20

At a Glance

Introduction

This chapter describes the instruction BMDI.

What's in this Chapter?

This chapter contains the following topics:

Short Description: BMDI - Block Move Interrupts Disabled

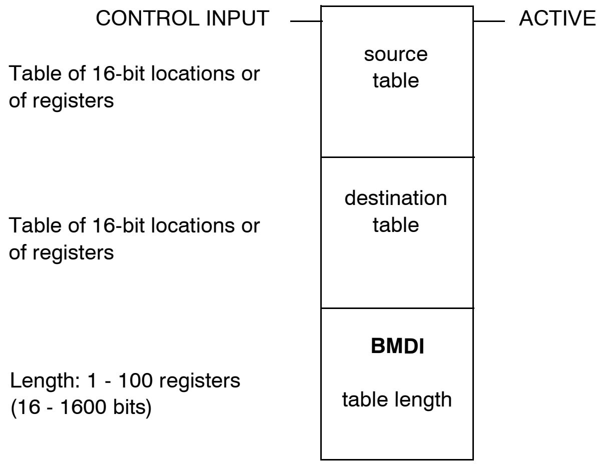
Function Description

The BMDI instruction masks the interrupt, initiates a block move (BLKM) operation, then unmasks the interrupts.

Representation: BMDI - Block Move Interrupts Disabled

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = masks interrupt, initiates a block move, then unmasks the interrupts
source table (top node)	0x, 1x, 3x, 4x	INT, UINT, WORD	Source table that will have its contents copied in the block move
destination table (middle node)	0x, 4x	INT, UINT, WORD	Destination table where the contents of the source table will be copied in the block move
table length (bottom node)		INT, UINT	Integer value, specifies the table size, i.e. the number of registers, in the source and destination tables (they are of equal length). Range: 1 ... 100.
Top output	0x	None	Echoes the state of the top input

BROT: Bit Rotate

21

At a Glance

Introduction

This chapter describes the instruction BROT.

What's in this Chapter?

This chapter contains the following topics:

Short Description

Function Description

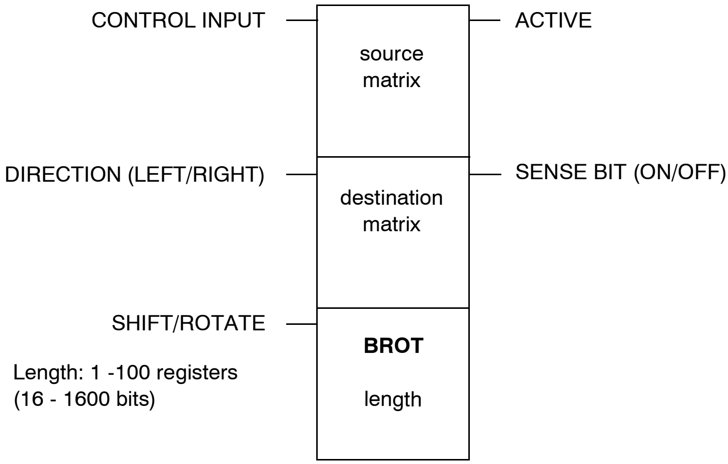
The BROT (bit rotate) instruction shifts the bit pattern in a source matrix, then posts the shifted bit pattern in a destination matrix. The bit pattern shifts left or right by one position per scan.

	WARNING
⚠	**Overriding of any disabled coils within a destination matrix without enabling them.** BROT will override any disabled coils within a destination matrix without enabling them. This can cause injury if a coil has been disabled for repair or maintenance if BROT unexpectedly changes the coil's state. **Failure to follow this precaution can result in death, serious injury, or equipment damage.**

Representation: BROT - Bit Rotate

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = shifts bit pattern in source matrix by one
Middle input	0x, 1x	None	ON= shift left OFF = shift right
Bottom input	0x, 1x	None	OFF = exit bit falls out of the destination matrix ON = exit bit wraps to start of the destination matrix
source matrix (top node)	0x, 1x, 3x, 4x	ANY_BIT	First reference in the source matrix, i.e. in the matrix that will have its bit pattern shifted
destination matrix (middle node)	0x, 4x	ANY_BIT	First reference in the destination matrix, i.e. in the matrix that shows the shifted bit pattern
length (bottom node)	0x	INT, UINT	Matrix length; range: 1 ... 100
Top output	0x	None	Echoes state of the top input
Middle output	0x	None	OFF = exit bit is 0 ON = exit bit is 1

Parameter Description

Matrix Length (Bottom Node)

The integer value entered in the bottom node specifies the matrix length, i.e. the number of registers or 16-bit words in each of the two matrices. The source matrix and destination matrix have the same length. The matrix length can range from 1 ... 100, e.g. a matrix length of 100 indicates 1600 bit locations.

Result of the Shift (Middle Output)

The middle output indicates the sense of the bit that exits the source matrix (the leftmost or rightmost bit) as a result of the shift.

CALL: Activate Immediate or Deferred DX Function

22

AT A GLANCE

Introduction

This chapter describes the instruction CALL.

What's in this Chapter?

This chapter contains the following topics:

Short Description: CALL - Activate Immediate or Deferred DX Function

Function Description

A CALL instruction activates an immediate or deferred DX function from a library of functions defined by function codes. The Copro copies the data and function code into its local memory, processes the data, and copies the results back to Controller memory.

Function Codes:

- **0-499:** User Immediate/Deferred DXs
- **500-9999:** System Immediate/Deferred DXs

The two MSBs of the top register are the Copro# in a multiple Copro system.

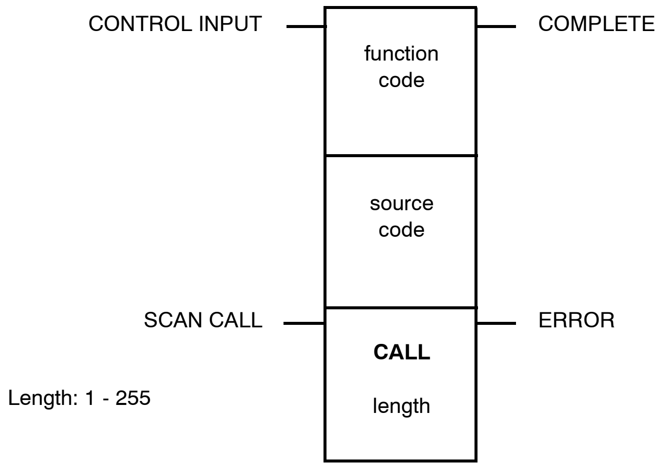
Representation: CALL - Activate Immediate DX Function

Overview

The content in this section applies specifically to the Immediate DX function of the CALL instruction.

Symbol

Representation of the instruction for an Immediate DX CALL



Parameter Description

Description of the instruction's parameters for an Immediate DX CALL

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON initiates the CALL.
Bottom input	0x, 1x	None	The input to the bottom node is used with an immediate DX function to keep scanning the instruction regardless of the state of the top input. A list of the codes, their names, and their function is detailed in the table below named Immediate DX Functions.
value (top node)	0x, 3x	INT, UINT	The top node is used to specify the function code to be executed. It may be entered explicitly as a constant or as a value in a 4xxxx holding register. The codes fall into two ranges: ● 0 through 499 are for user-definable DXs ● 500 through 9999 are for system DXs Both User-definable and System-definable codes apply to both immediate and deferred. Both User-definable and System-definable are provided by Schneider Electric.
register (middle node)	4x	INT, UINT	The 4xxxx register in the middle node is the first in a block of registers to be passed to the Copro for processing.
length (bottom node)		INT, UINT	The number of registers in the block is defined in the bottom node.
Top output	0x	None	ON when the function completes successfully.
Bottom output	0x	None	The output from the bottom node will go ON if an error is detected in the function.

Immediate DX Functions

This table lists the Immediate DX Functions

Name	Code	Function
f_config	500	Obtain Copro configuration data.
f_2md_fl	501	Convert a two-register long integer to 64-bit floating point.
f_fl_2md	502	Convert floating point to two-register long integer.
f_4md_fl	503	Convert a four-register long integer to floating point.
f_fl_4md	504	Convert floating point to four-register long integer.
f_1md_fl	505	Convert a one-register long integer to floating point.
f_fl_1m	506	Convert floating point to one-register long integer.
f_exp	507	Exponential function.
f_log	508	Natural logarithm.
f_log10	509	Base 10 logarithm.
f_pow	510	Raise to a power.
f_sqrt	511	Square root.
f_cos	512	Cosine.
f_sin	513	Sine.
f_tan	514	Tangent.
f_atan	515	Arc tangent x.
f_atan2	516	Arc tangent y/x.
f_asin	517	Arc sine.
f_acos	518	Arc cosine.
f_add	519	Add.
f_sub	520	Subtract.
f_mult	521	Multiply.
f_div	522	Divide.
f_deg_rad	523	Convert degrees to radians.
f_rad_deg	524	Convert radians to degrees.
f_swap	525	Swap byte positions within a register.
f_comp	526	Floating point compare.
f_dbwrite	527	Write Copro register database from PLC.
f_dbread	528	Read Copro register database from PLC.

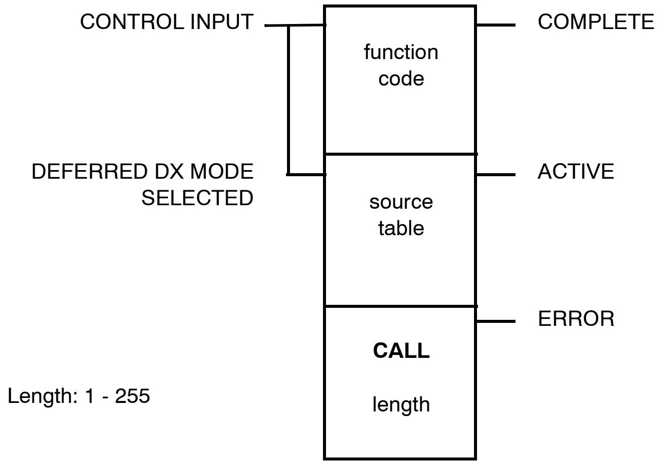
Representation: CALL - Activate Deferred DX Function

Overview

The content in this section applies specifically to the Deferred DX function of the CALL instruction.

Symbol

Representation of the instruction for a Deferred DX CALL



Parameter Description

Description of the instruction’s parameters for a Deferred DX CALL

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON initiates the CALL.
Middle input	0x, 1x	None	The instruction calls a deferred DX when the input to the middle node is enabled. A list of the codes, their names, and their function is detailed in the table below named Deferred DX Functions.
value (top node)	0x, 3x	INT, UINT	The top node is used to specify the function code to be executed. It may be entered explicitly as a constant or as a value in a 4xxxx holding register. The codes fall into two ranges: ● 0 through 499 are for user-definable DXs ● 500 through 9999 are for system DXs Both User-definable and System-definable codes apply to both immediate and deferred. Both User-definable and System-definable are provided by Schneider Electric.
register (middle node)	4x	INT, UINT	The 4xxxx register in the middle node is the first in a block of registers to be passed to the Copro for processing.
length (bottom node)		INT, UINT	The number of registers in the block is defined in the bottom node.
Top output	0x	None	ON when the function completes successfully.
Middle output	0x	None	The output from the middle node, which is used only with deferred DX functions, goes ON to indicate that the function s in process.
Bottom output	0x	None	The output from the bottom node will go ON if an error is detected in the function.

Deferred DX Functions

This table lists the Deferred DX Functions

Name	Code	Function
f_config	500	Obtain Copro configuration data.
f_d_dbwr	501	Write Copro register database from PLC.
f_d_dbrd	502	Read Copro register database from PLC.
f_dgets	515	Issue dgets() on comm line.
f_dputs	516	Issue dputs() on comm line.
f_sprintf	518	Generate a character string.
f_sscanf	519	Interpret a character string.
f_egets	520	IEEE-488 gets() function.
f_eputs	521	IEEE-488 puts() function.
f_ectl	522	IEEE-488 error control function.

CANT - Interpret Coils, Contacts, Timers, Counters, and the SUB Block

23

At A Glance

Introduction

This chapter describes the instruction CANT.

What's in this Chapter?

This chapter contains the following topics:

Short Description: CANT - Interpret Coils, Contacts, Timers, Counters, and the SUB Block

Function Description

This DX Loadable Function Block, upon initializing a triggering contact, analyzes your ladder logic to extract the specific column and the corresponding contact id's where power flow has stopped. The CANT block contains 20 registers. A MSTR block is used to export data from the CANT's 20 registers to a PC running the "Action Monitor" program.

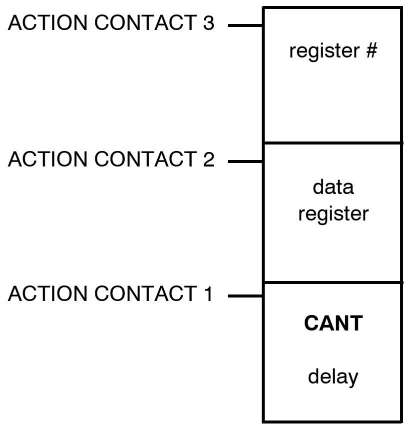
The CANT block is specifically used to interpret coils, contacts, timers, counters, and the SUB block. You may not use any other types of ladder logic instructions in a network. Otherwise, you receive incorrect results. If, however, you must use one of the other ladder logic instructions you may place them in a separate network linked to a coil that is referenced to the network containing the CANT block.

Note: Only 24-bit logic Quantum and 984 PLCs support the DX Loadable Function Block. 16-bit controllers will not work with this particular block.

Representation: CANT - Interpret Coils, Contacts, Timers, Counters, and the SUB Block

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	Action Contact 3 Please see the **Note** below.
Middle input	0x, 1x	None	Action Contact 2 Please see the **Note** below.
Bottom input	0x, 1x	None	Action Contact 1 Please see the **Note** below.
register # top node	4x	INT, UINT	Each CANT block contains a block of 10 setup registers, which will automatically fill these 10 registers with internal data.
data register middle node	4x	INT, UINT	This node is the start of the 4x output data registers. (For expanded and detailed information please see the section *Output Data Registers Table (Middle Node), p. 162*.)
delay bottom node		INT, UINT	A delay timer value with 10ms increments. The value 1 is assigned to OFF.

Note: When any of the above inputs are activated the CANT function block begins to solve the routine. The bottom node specifies a delay time in 10ms increments that the block uses to delay the start of the solve routine.

Parameter Description: CANT - Interpret Coils, Contacts, Timers, Counters, and the SUB Block

Output Data Registers Table (Middle Node)

Output Data Registers Table

Output Data Register	Description (Purpose)
4x	Contains the address of the "CANT in use flag" coil number Coil must be programmed with NO POWER CONNECTED FROM THE LEFT in the last network of your ladder logic
4x + 01	CANT version number in hexadecimal format (for example, 0105 for v1.05)
4x + 02	Hi Byte = Internal operational flags Lo Byte = MB+ address of a PLC
4x + 03	Output coil number (a variable that is dependent on the block's state)
4x + 04	The Id of the trigger contact or coil Bit 15 → 0 - if a coil; 1 - if a contact Bit 14-00 → coil or contact number (1 based)
4x + 05	Hi 12 bits = network number where logic fails (1 based) Lo 4 bits = column number where logic fails (1 based)
4x + 06	Rung #1: Hi Byte = node state Lo Byte = node type (opcode from node database)
4x + 07	Rung #1: Contact number (1 based)
4x + 08	Rung #2: Refer to 4x + 06
4x + 09	Rung #2: Refer to 4x + 07
4x + 10	Rung #3: Refer to 4x + 06
4x + 11	Rung #3: Refer to 4x + 07
4x + 12	Rung #4: Refer to 4x + 06
4x + 13	Rung #4: Refer to 4x + 07
4x + 14	Rung #5: Refer to 4x + 06
4x + 15	Rung #5 Refer to 4x + 07
4x + 16	Rung #6: Refer to 4x + 06
4x + 17	Rung #6: Refer to 4x + 07
4x + 18	Rung #7: Refer to 4x + 06
4x + 19	Rung #7: Refer to 4x + 07

Programming

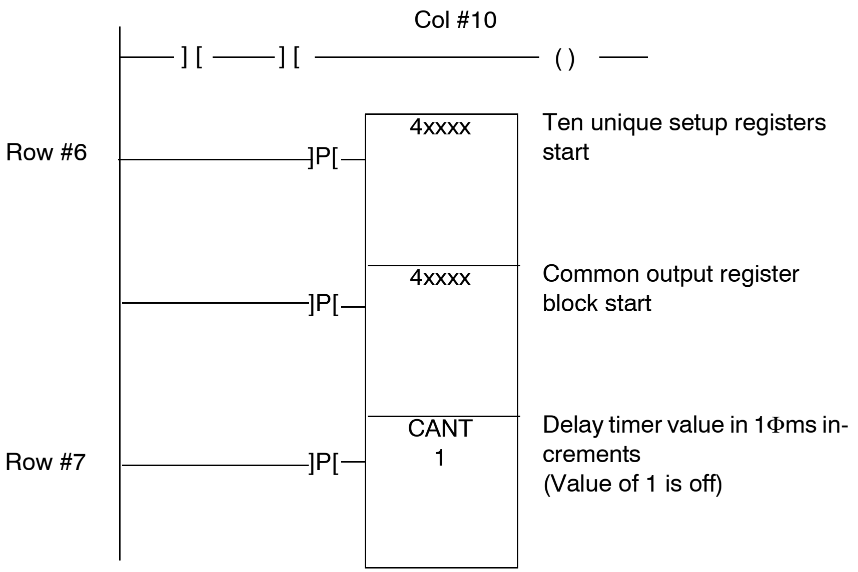
Each network can only contain one COIL and one CANT block, which must be located in Column 10, Row 5. Column 9 of the BOTTOM rung contains the Power Input for the triggers (Action Contacts) to the CANT block, which will provide more space for your ladder logic programming.

Note: This is not at the top of the block as it usually is with DX blocks.

In any of the available row positions 5, 6 or 7 you may have up to 3 triggers which must be a transitional type of either [P] or [N]. The CANT block node number will default to 22 (hexadecimal) and not be changed.

Ladder Node Setup

Ladder Node Setup



MSTR Write Data Setup

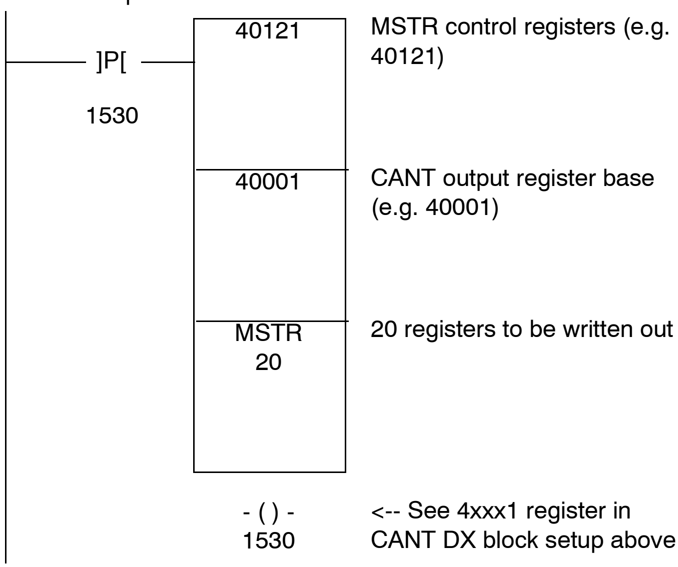
The purpose of the MSTR block is to send the 20 4x CANT registers to a PC based "Action Monitor" program. This transmittal of registers is done using either MB+ or an Ethernet TCP/IP Modbus.
Below is an example:
MSTR Statistics Control Registers

Register	Value	Description
400121	1	Write data function
400122	?	MSTR error register
400123	20	# of data registers to send
400124	40001	Start of data registers
400125	22	Destination MB+ address
400126	1	MB+ routing
400127	0	MB+ routing
400128	0	MB+ routing
400129	0	MB+ routing

Note: It is necessary to program a MSTR block for each receiving (PC) address if you want to transmit data to more than one PC running "Action Monitor".

MSTR Setup

MSTR Setup



CHS: Configure Hot Standby

24

At a Glance

Introduction

This chapter describes the instruction CHS.

What's in this Chapter?

This chapter contains the following topics:

Short Description

Function Description

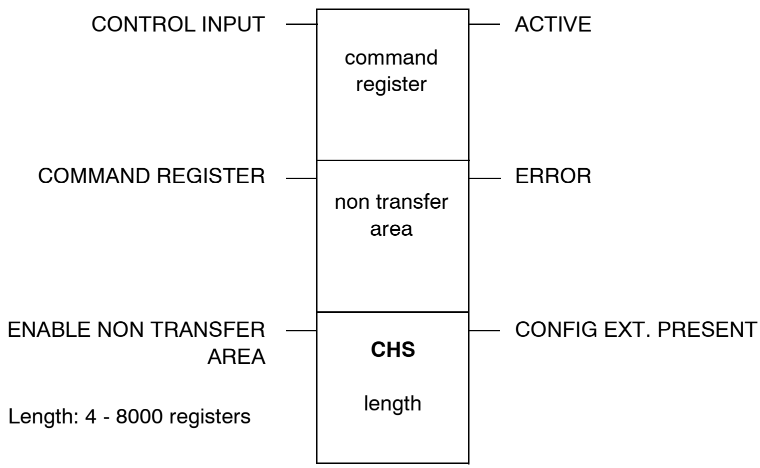
> **Note:** This instruction is only available if you have unpacked and installed the DX Loadables. For further information, see "*Installation of DX Loadables, p. 109*."

The logic in the CHS loadable is the engine that drives the Hot Standby capability in a Quantum PLC system. Unlike the HSBY instruction, the use of the CHS instruction in the ladder logic program is optional. However, the loadable software itself must be installed in the Quantum PLC in order for a Hot Standby system to be implemented.

Representation: CHS - Configure Hot Standby

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	Execute Hot Standby (unconditionally)
Middle input	0x, 1x	None	ON = Enable command register
Bottom input	0x, 1x	None	ON = Enable non transfer area OFF = non transfer area will not be used and the Hot Standby status register will not exist
command register (top node)	4x	INT, UINT, WORD	Hot Standby command register (For expanded and detailed information please see *Parameter Description Command Register (Top Node), p. 170*).)
nontransfer area (middle node)	4x	INT, UINT, WORD	First register in the nontransfer area of state RAM (For expanded and detailed information please see *Parameter Description Nontransfer Area (Middle Node), p. 171*.)
length (bottom node)		INT, UINT	Number of registers of the Hot Standby nontransfer area in state RAM; range 4 ... 8000
Top output	0x	None	Echoes the state of the top input
Middle output	0x	None	ON = System detects interface error
Bottom output	0x	None	ON = System configuration set by configuration extension

Detailed Description

Hot Standby System Configuration via the CHS Instruction

Program the CHS instruction in network 1, segment 1 of your ladder logic program and unconditionally connect the top input to the power rail via a horizontal short (as the HSBY instruction is programmed in a 984 Hot Standby system).

This method is particularly useful if you are porting Hot Standby code from a 984 application to a Quantum application. The structure of the CHS instruction is almost exactly the same as the HSBY instruction. You simply remove the HSBY instruction from the 984 ladder logic and replace it with a CHS instruction in the Quantum logic.

If you are using the CHS instruction in ladder logic, the only difference between it and the HSBY instruction is the use of the bottom output. This output senses whether or not method 2 has been used. If the Hot Standby configuration extension screens have been used to define the Hot Standby configuration, the configuration parameters in the screens will override any different parameters defined by the CHS instruction at system startup.

For detailes discussion of the issues related to the configuration extension capabilities of a Quantum Hot Standby system, refer to the *Modicon Quantum Hot Standby System Planning and Installation Guide.*

Parameter Description Execute Hot Standby (Top Input)

When the CHS instruction is inserted in ladder logic to control the Hot Standby configuration parameters, its top input must be connected directly to the power rail by a horizontal short. No control logic, such as contacts, should be placed between the rail and the input to the top node.

WARNING

Erratic behavior in the Hot Standby system

Although it is legal to enable and disable the nontransfer area while the Hot Standby system is running, we strongly discourage this practice. It can lead to erratic behavior in the Hot Standby system.

Failure to follow this precaution can result in death, serious injury, or equipment damage.

Parameter Description Command Register (Top Node)

The 4x register entered in the top node is the Hot Standby command register; eight bits in this register are used to configure and control Hot Standby system parameters:

Usage of command word:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Bit	Function
1 - 5	Not used
6	0 = swap Modbus port 3 address during switchover 1 = no swap
7	0 = swap Modbus port 2 address during switchover 1 = no swap
8	0 = swap Modbus port 1 address during switchover 1 = no swap
9 - 11	Not used
12	0 = allow exec upgrade only after application stops 1 = allow the upgrade without stopping the application
13	0 = force standby offline if there is a logic mismatch 1 = do not force
14	0 = controller B is in **OFFLINE** mode 1 = controller B is in **RUN**
15	0 = controller A is in **OFFLINE** mode 1 = controller A is in **RUN**
16	0 = disable keyswitch override 1 = enable the override

Note: The Hot Standby command register must be outside of the nontransfer area of state RAM.

Parameter Description Nontransfer Area (Middle Node)

The 4x register entered in the middle node is the first register in the nontransfer area of state RAM. The nontransfer area must contain at least four registers, the first three of which have a predefined usage:

Register	Content
Displayed and first implied	Reverse transfer registers for passing information from the standby to the primary PLC
Second implied	CHS status register

The content of the remaining registers is application-specific; the length is defined in the parameter "length" (bottom node).

The 4x registers in the nontransfer area are never transferred from the primary to the standby PLC during the logic scans. One reason for scheduling additional registers in the nontransfer area is to reduce the impact of state RAM transfer on the total system scan time.

CHS Status Register

Usage of status word:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Bit	Function
1	1 = the top output is ON (indicating Hot Standby system is active)
2	1 = the middle output is ON (indicating an error condition)
3 - 10	Not used
11	0 = PLC switch is set to A 1 = PLC switch is set to B
12	0 = PLC logic is matched 1 = there is a logic mismatch
13 - 14	The 2 bit value is: ● 0 1 if the other PLC is in **OFFLINE** mode ● 1 0 if other PLC is running in primary mode ● 1 1 if other PLC is running in standby mode
15 - 16	The 2 bit value is: ● 0 1 if this PLC is in **OFFLINE** mode ● 1 0 if this PLC is running in primary mode ● 1 1 if this PLC is running in standby mode

CKSM: Check Sum

25

At a Glance

Introduction

This chapter describes the instruction CKSM.

What's in this Chapter?

This chapter contains the following topics:

Short Description

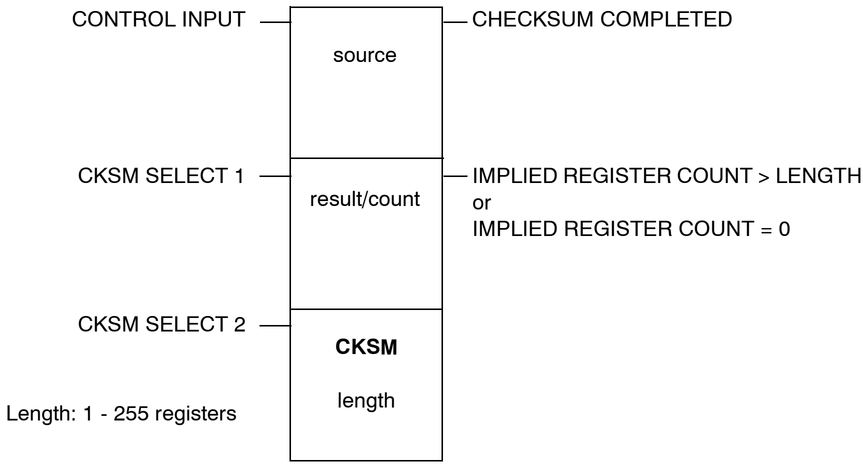
Function Description

Several PLCs that do not support Modbus Plus come with a standard checksum (CKSM) instruction. CKSM has the same opcode as the MSTR instruction and is not provided in executive firmware for PLCs that support Modbus Plus.

Representation: CKSM - Checksum

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	Initiates checksum calculation of source table (For expanded and detailed information please see Parameter Description: *Inputs, p. 177*.)
Middle input	0x,1x	None	CKSM select 1 (For expanded and detailed information please see Parameter Description: *Inputs, p. 177*.)
Bottom input	0x, 1x	None	CKSM select 2 (For expanded and detailed information please see Parameter Description: *Inputs, p. 177*.)
source (top node)	4x	INT, UINT	First holding register in the source table. The checksum calculation is performed on the registers in this table.
result/count (middle node)	4x	INT, UINT	First of two contiguous registers (For expanded and detailed information please see *Result / Count (Middle Node), p. 177*.)
length (bottom node)		INT	Number of 4x registers in the source table; range: 1 ... 255
Top output	0x	None	ON = Checksum calculation successful
Middle output	0x	None	ON = implied register count > length or implied register count =0

Parameter Description

Inputs

The states of the inputs indicate the type of checksum calculation to be performed:

CKSM Calculation	Top Input	Middle Input	Bottom Input
Straight Check	ON	OFF	ON
Binary Addition Check	ON	ON	ON
CRC-16	ON	ON	OFF
LRC	ON	OFF	OFF

Result / Count (Middle Node)

The 4x register entered in the middle node is the first of two contiguous 4x registers:

Register	Content
Displayed	Stores the result of the checksum calculation
First implied	Posts a value that specifies the number of registers selected from the source table as input to the calculation. The value posted in the implied register must be ≤ length of source table.

CMPR: Compare Register

26

At a Glance

Introduction

This chapter describes the instruction CMPR.

What's in this Chapter?

This chapter contains the following topics:

Short Description

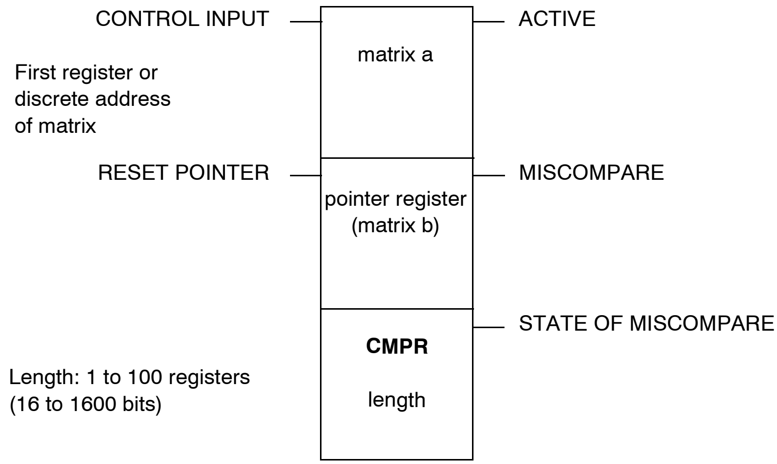
Function Description

The CMPR instruction compares the bit pattern in matrix a against the bit pattern in matrix b for miscompares. In a single scan, the two matrices are compared bit position by bit position until a miscompare is found or the end of the matrices is reached (without miscompares).

Representation: CMPR - Logical Compare

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = initiates compare operation
Middle input	0x, 1x	None	OFF = restart at last miscompare ON = restart at the beginning
matrix a (top node)	0x, 1x, 3x, 4x	ANY_BIT	First reference in matrix a, one of the two matrices to be compared
pointer register (middle node)	4x	WORD	Pointer to matrix b: the first register in matrix b is the next contiguous 4x register following the pointer register
length (bottom node)		INT, UINT	Matrix length; range: 1 ... 100
Top output	0x	None	Echoes state of the top input
Middle output	0x	None	ON = miscompare detected
Bottom output	0x	None	ON = miscompared bit in matrix a is 1 OFF = miscompared bit in matrix a is 0

Parameter Description

Pointer Register (Middle Node)

The pointer register entered in the middle node must be a 4x holding register. It is the pointer to matrix b, the other matrix to be compared. The first register in matrix b is the next contiguous 4x register following the pointer register.
The value stored inside the pointer register increments with each bit position in the two matrices that is being compared. As bit position 1 in matrix a and matrix b is compared, the pointer register contains a value of 1; as bit position 2 in the matrices are compared, the pointer value increments to 2; etc.
When the outputs signal a miscompare, you can check the accumulated count in the pointer register to determine the bit position in the matrices of the miscompare.

Matrix Length (Bottom Node)

The integer value entered in the bottom node specifies a length of the two matrices, i.e. the number of registers or 16-bit words in each matrix. (Matrix a and matrix b have the same length.) The matrix length can range from 1 ... 100, i.e. a length of 2 indicates that matrix a and matrix b contain 32 bits.

Coils

27

At A Glance

Introduction

This chapter describes the instruction element Coils.

What's in this Chapter?

This chapter contains the following topics:

Short Description: Coils

Function Description

A coil is a discrete output that is turned ON and OFF by power flow in the logic program. A single coil is tied to a 0xxxx reference in the PLC's state RAM. Because output values are updated in state RAM by the PLC, a coil may be used internally in the logic program or externally via the I/O map to a discrete output unit in the control system. When a coil is ON, it either passes power to a discrete output circuit or changes the state of an internal relay contact in state RAM.

Coil Types

There are two types of coils:

- Normal coil -( )-
  A normal or non-retentive or normal coil looses state when power to controller is lost.
  When power is removed from a PLC, a normal coil will be turned OFF. Once power is restored, the coil will always be in the OFF state on the first logic scan.
- Memory-retentive or latched coil -(M)- or -(L)-
  A memory-retentive or latched coil does NOT loose state when power to controller is lost.
  If a memory-retentive (or latched) coil is ON at the time a PLC loses power, the coil will come back up in an ON state when power is restored. The coil will maintain that ON state for the first logic scan, and then the logic program will take control.

Coils are referenced as 0xxxx. They may be disabled and forced ON or OFF. Disabling a coil stops the user programmed logic from changing the state of the coil.

Note: Disabled Coils used as destinations in DX function blocks may have their state overwritten by the function.

General Usage Guidelines: Coils

Overview

Once a 0x reference number has been assigned to a coil, it cannot be assigned to any other coils in the logic program.
An 0x reference number can be referenced to any number of relay contacts, which can then be controlled via the state of the coil with same reference number. Most panel software packages have a feature called tracing with which you can locate the positions in ladder logic of the contacts controlled by a coil. Refer to your software user manual for more details.

Enable/Disable Capabilities for Discrete Values

Via panel software, you may disable a logic coil or a discrete input in your logic program.
A disable condition will cause the following:
- Input field device to have no control over its assigned 1x logic
- Logic to have no control over the disable 9x value

Memory protection in the PLC must be OFF before you disable or enable a coil or a discrete input.

> **Note:** There is an important exception that you need to be aware of when disabling coils:
> Data transfer functions allow coils in their *destination* nodes to recognize the current ON/OFF state of ALL coils, whether those coils are disabled or not, and this recognition causes the logic to respond accordingly—maybe producing unexpected and undesirable effects.
> If you are expecting a disabled coil to remain disabled in the DX function, your application may experience unexpected and undesirable effects.

Forcing Discretes ON and OFF

Most panel software also provides FORCE ON and FORCE OFF capabilities. When a coil or discrete input is disabled, you can change its state from OFF to ON with FORCE ON, and from ON to OFF with FORCE OFF.
When a coil or discrete input is enabled, it cannot be forced ON or OFF.

COMM - ASCII Communications Function

28

At A Glance

Introduction

This chapter describes the COMM instruction.

What's in this Chapter?

This chapter contains the following topics:

Short Description: COMM - ASCII Communications Block

Function Description

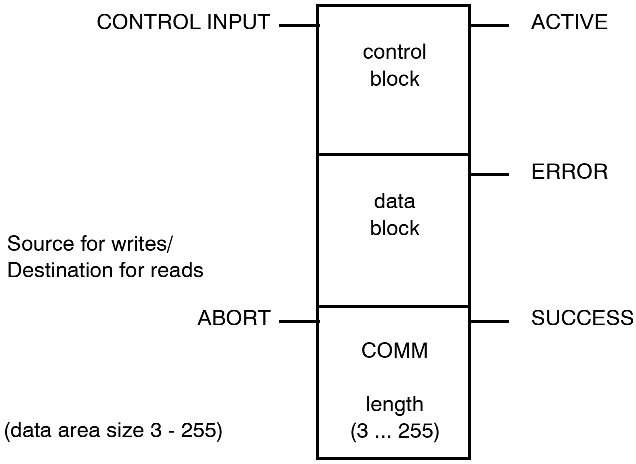
The ASCII Communications Function (COMM) block is used to transmit/receive ASCII data (in the form of a single ASCII character, 1 to 4 integers or 1 to 4 hexadecimal numbers) to or from the simple ASCII port. The COMM instruction gives you the ability to read and write canned messages to/from ASCII character input/output devices via one of the built-in communication ports on a Micro PLC or, if the PLC is a parent, via a comm port on one of the child PLCs on the expansion link.

Note: Available only on the Micro 311, 411, 512, and 612 controllers.

Representation: COMM - ASCII Communications Function

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON starts the COMM operation
Bottom input	0x, 1x	None	ON aborts the operation and sets the middle out.
control block (top node)	4x	INT, UINT	The 4xxxx register entered in the top node is the first of 10 contiguous holding registers in the control block. (For the register usage please see the Register Usage Table below.
data block (middle node)	4x	INT, UINT	The middle node contains the first 4xxxx register of the data block - a table where variable message data is placed. In a read operation, the data block is a destination table. In a write operation the data block is a source table.
length (bottom node)		INT, UINT	The integer value entered in the bottom node specifies the length, which is the number of registers in the data block. The length can range from 3 through 255.
(Top output)	0x	None	Echoes the state of the top input.
Middle output	0x	None	ON = error detected (for one scan).
Bottom output	0x	None	ON = operation complete (for one scan).

Register Usage Table

This table details the register usage for the top node.

Register	Usage
4xxxx + 0	Operation Code
4xxxx + 1	Error Status
4xxxx + 2	Number of data fields provided/expected
4xxxx + 3	Number of data fields processed
4xxxx + 4	Reserved
4xxxx + 5	Port Number (1 for local, 2 for child #1, 3 for child #2, etc.
4xxxx + 6	Reserved
4xxxx + 7	Reserved
4xxxx + 8	Reserved
4xxxx + 9	Active Status Timer

COMP: Complement a Matrix

29

At a Glance

Introduction

This chapter describes the instruction COMP.

What's in this Chapter?

This chapter contains the following topics:

Short Description

Function Description

The COMP instruction complements the bit pattern, i.e. changes all 0’s to 1’s and all 1’s to 0’s, of a source matrix, then copies the complemented bit pattern into a destination matrix. The entire COMP operation is accomplished in one scan.

WARNING

Overriding of any disabled coils in the destination matrix without enabling them.

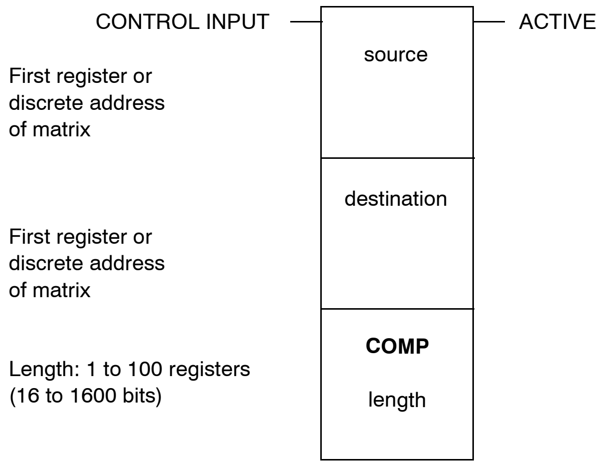
COMP will override any disabled coils in the destination matrix without enabling them. This can cause injury if a coil has been disabled for repair or maintenance because the coil’s state can be changed by the COMP operation.

Failure to follow this precaution can result in death, serious injury, or equipment damage.

Representation: COMP - Logical Compliment

Symbol

Representation of the instruction



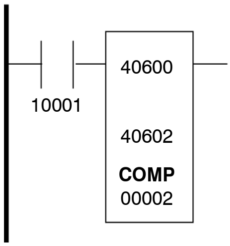
Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = initiates the complement operation
source (top node)	0x, 1x, 3x, 4x	ANY_BIT	First reference in the source matrix, which contains the original bit pattern before the complement operation
destination (middle node)	0x, 4x	ANY_BIT	First reference in the destination matrix where the complemented bit pattern will be posted
length (bottom node)		INT, UINT	Matrix length; range: 1 ... 100.
Top output	0x	None	Echoes state of the top input

A COMP Example

When contact 10001 passes power, the bit pattern in the *source matrix* (registers 40600 and 40601) is complemented, then the complemented bit pattern is posted in the *destination matrix* (registers 40602 and 40603). The original bit pattern is maintained in the *source matrix*.



source matrix
40600 = 1111111100000000   40601 = 1111111100000000

Complemented *destination matrix*
40602 = 0000000011111111   40603 = 0000000011111111

Parameter Description

Matrix Length (Bottom Node)	The integer value entered in the bottom node specifies a matrix length, i.e. the number of registers or 16-bit words in the matrices. Matrix length can range from 1 ... 100. A length of 2 indicates that 32 bits in each matrix will be complemented.

Contacts

30

At A Glance

Introduction

This chapter describes the instruction element Contacts.

What's in this Chapter?

This chapter contains the following topics:

Short Description: Contacts

Function Description

Contacts are used to pass or inhibit power flow in a ladder logic program.

Representation: Contacts

Function Description

They are discrete, which means each consumes one I/O point in ladder logic. A single contact can be tied to a 0x or 1x reference number in the PLC's state RAM, in which case each contact consumes one node in a ladder network.
Four kinds of contacts are available:
- Normally open (N.O.) contacts
- Normally closed (N.C.) contacts
- Positive transitional (P.T.) contacts
- Negative transitional (N.T.) contacts

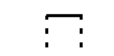
Referencing Normally Open/ Normally Closed Contacts

Normally open -| |- and normally closed -|\|- contacts may be referenced by inputs (1xxxx) or coils (0xxxx).

<table>
<tr><td></td><td></td><td colspan="2">Field Device state vs. Programmed Contact Flow</td></tr>
<tr><td>Field Device</td><td>Programmed Contact</td><td>Field Contact Closed</td><td>Field Contact Open</td></tr>
<tr><td>-| |-</td><td>-| |-</td><td>Passes Power</td><td></td></tr>
<tr><td></td><td>-|\|-</td><td></td><td>Passes Power</td></tr>
<tr><td>-|\|-</td><td>-| |-</td><td></td><td>Passes Power</td></tr>
<tr><td></td><td></td><td>Passes Power</td><td></td></tr>
</table>

Referencing Transitional Contacts

Transitional contacts positive -| ↑ |- and negative -| ↓ |- contacts may be referenced by inputs (1xxxx) or coils (0xxxx).

<table>
<tr><td colspan="2">State Table Transition</td><td colspan="3">Power Flow at Transition</td></tr>
<tr><td>-|↑|-</td><td>Off to On</td><td>On</td><td rowspan="2"></td><td>1 Scan Power</td></tr>
<tr><td>-|↓|-</td><td>On to Off</td><td>Off</td><td>Flow Pulse</td></tr>
</table>

> **Note:** A transitional contact will pass power continuously if the referenced coil is skipped by a SKP instruction or by the segment scheduler. A transitional contact may not pass power if it is referenced to an input that has been scheduled to read from the I/O drop more than once per scan via the segment scheduler.

CONV - Convert Data

31

At A Glance

Introduction

This chapter describes the instruction CONV.

What's in this Chapter?

This chapter contains the following topics:

Short Description: CONV - Convert Data

Function

The Convert block is a 484-replacement instruction, and it is one of four replacement instructions. The CONV block is used to convert
- Discrete data to a holding register
- Holding-register data to discrete data

The conversion can be either
- Binary to binary
- BCD to binary (discrete to register)
- Binary to BCD (register to discrete)

This block uses 12 bits in 12 bits out, but if the conversion is straight binary to binary, bits 11 and 12 are forced off.

In converting discretes to a holding register, the source is specified as a constant which implies a 1xxxx and the destination is specified as a constant which implies a 4xxxx (for example, 00049 implies 40049).

In converting a register to output discretes, the source is specified as a holding register (4xxxx) and the destination is specified as a constant which implies a 0xxxx. For example 00032 implies 12 coils with 00032.

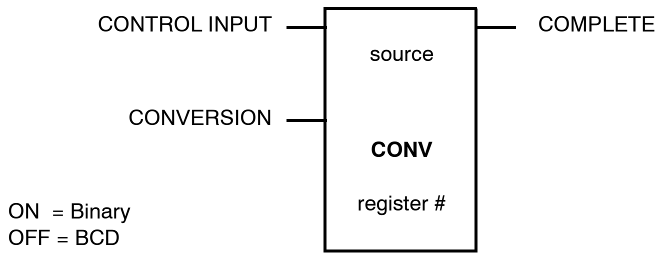
Important: Care should be taken when converting register data to discretes as coils may inadvertently be activated.

Note: Available only on the 984-351 and 984-455 PLCs.

Representation: CONV - Convert Data

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON initiates specified operation
Bottom input	0x, 1x	None	ON = Binary OFF = BCD
source (top node)	4x	INT, UINT	Converts content of register
register (bottom node)	3x	INT, UINT	
Top output	0x	None	Operation Successful

CTIF - Counter, Timer, and Interrupt Function

32

At A Glance

Introduction

This chapter describes the CTIF instruction.

What's in this Chapter?

This chapter contains the following topics:

Short Description: CTIF - Counter, Timer, and Interrupt Function

Function Description

The CTIF block is used by a parent PLC to access child functions over an I/O expansion bus. The Parent function block will complete in the same scan. If multiple blocks exist, the last one executed will be used.

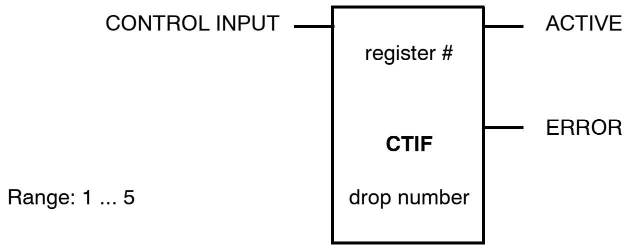
The CTIF instruction is used with the Micro PLCs to set up the inputs for hard-wired interrupt and/or hard-wired counter/timer operations. This instruction always starts and finishes in the same scan. The CTIF instruction is a configuration/operation tool for Modicon Micro PLCs that contain hardware interrupts (all models except the 110CPU311 Models). The actual counter/timer and interrupts are in the PLC hardware, and the CTIF instruction is what is used to set up this hardware.

> **Note:** The Counter, Timer, Interrupt function (CTIF) is only available on Micro 311, 411, 512, and 612 controllers.

Representation: CTIF - Counter, Timer, Interrupt Function

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON initiates specified operation
register # (top node)	4x	INT	The 4xxxx register entered in the top node is the first of four contiguous holding registers in the CTIF parameter block. (For expanded and detailed information about the four registers please see the section named *Parameter Description: CTIF - Register Usage Table (Top Node), p. 208*.)
drop number (bottom node)		INT	The integer value entered in the bottom node indicates the drop number where the operation will be performed. The drop number is in the range of 1 through 5.
Top output	0x	None	Echoes state of the top input
Bottom output	0x	None	Error

Parameter Description: CTIF - Register Usage Table (Top Node)

Overview of Section	The top node holds four contiguous registers, 4x through 4x+3. This section describes how those registers are used and configured in the top node.

First Register (4x) Usage

The first register, 4x, gives you information either about the type of error generated or about the type of operation being performed. When you configure the register you need to consider both how the bits will be used, **Bit Usage**, and the results of **ON**/ **OFF Combinations**.

Here is a graphic demonstrating the **Bit Usage** for the first register (4x),

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

and the following table describes the **Bit Usage** for the first register (4x).

Bit	Usage
1 - 4	Reserved
5 - 8	Error/Operation type messages
9 - 14	Reserved
15	Set Mode
16	Get Mode

The following table describes the **ON**/**OFF Combinations** for bits 5 through 8 and the error/operation type message generated by the first register (4x).

Bit	5	6	7	8	Description
	0	0	0	0	No error detected
	0	0	0	1	Unsupported operation type specified
	0	0	1	0	Interrupt 2 not supported in this model
	0	0	1	1	Interrupt 3 not supported while counter is selected
	0	1	0	0	Counter value of 0 specified
	0	1	0	1	Counter value too big (counter value > 16,383)
	0	1	1	0	Operation type supported only on local drop
	0	1	1	1	Specified drop not in I/O map
	1	0	0	0	No subroutine for enabled interrupt
	1	0	0	1	Remote drop is unhealthy
	1	0	1	0	Function not supported remotely

The following table describes **Bit Usage** and the **ON**/**OFF Combinations** for bits 15 and 16 of the first register (4x).

Bit	15	16	Description
	0	0	Set Mode
	0	1	Get Mode

Second Register (4x+1) Usage

The second register, 4x+1, allows you to control the set-up for the **Set Mode** operation. When you configure the register you need to consider both how the bits will be used, **Bit Usage**, and the results of the **ON/OFF Combinations**.
Here is a graphic demonstrating the **Bit Usage** for the second register (4x+1).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

The following tables describe both **Bit Usage** and the **ON/OFF Combinations** for bits 1 through 16 of the second register (4x+1).
The following table describes **Bit Usage** and **ON/OFF Combinations** for bits 1 and 2 of the second register (4x+1).

Bit	Usage
1	Terminal-count loading 0 - Disable 1 - Enable
2	Reserved

The following table describes **Bit Usage** and **ON/OFF Combinations** for bits 3 and 4 of the second register (4x+1).

Bit	3	4	Description
	0	1	Disable interrupt service for Interrupt 3
	1	0	Enable interrupt service for Interrupt 3

The following table describes **Bit Usage** and **ON/OFF Combinations** for bits 5 and 6 of the second register (4x+1).

Bit	5	6	Description
	0	1	Disable interrupt service for Interrupt 2
	1	0	Enable interrupt service for Interrupt 2

The following table describes **Bit Usage** and **ON/OFF Combinations** for bits 7 and 8 of the second register (4x+1).

Bit	7	8	Description
	0	1	Disable interrupt service for Interrupt 1
	1	0	Enable interrupt service for Interrupt 1

The following table describes **Bit Usage** and **ON/OFF Combinations** for bits 9 and 10 of the second register (4x+1).

Bit	9	10	Description
	0	1	Disable interrupt service for timer/counter interrupt
	1	0	Enable interrupt service for timer/counter interrupt

The following table describes **Bit Usage** and **ON/OFF Combinations** for bits 11 and 12 of the second register (4x+1).

Bit	11	12	Description
	0	1	Disable auto-restart operation
	1	0	Enable auto-restart operation

The following table describes **Bit Usage** and **ON/OFF Combinations** for bits 13 and 14 of the second register (4x+1).

Bit	13	14	Description
	0	1	Stop counter/timer operation
	1	0	Start counter/timer operation

The following table describes **Bit Usage** and **ON/OFF Combinations** for bits 15 and 16 of the second register (4x+1).

Bit	15	16	Description
	0	1	Counter Mode
	1	0	Timer Mode

Third Register (4x+2) Usage

The third register, 4x+2, gives you the status for the **Get Mode** operation. When you configure the register you need to consider both how the bits will be used, **Bit Usage**, and the results of the **ON**/**OFF Combinations**.
Here is a graphic demonstrating the **Bit Usage** for the third register (4x+2).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

The following table describes **Bit Usage** and **ON**/**OFF Combinations** for bits 1 through 16 for the third register (4x+2).

Bit	Usage
1	No subroutine for Interrupt 3
2	No subroutine for Interrupt 2
3	No subroutine for Interrupt 1
4	No subroutine for timer/counter interrupt
5 - 9	Reserved
10	Interrupt 3 0 - Disabled 1 - Enabled
11	Interrupt 2 0 - Disabled 1 - Enabled
12	Interrupt 1 0 - Disabled 1 - Enabled
13	Interrupt serve for time/counter input 0 - Disabled 1 - Enabled
14	Auto restart operation 0 - Disabled 1 - Enabled
15	Counter/timer operation 0 - Stopped 1 - Started
16	0 - Counter Mode 1 - Timer Mode

Fourth Register (4x+3) Usage

The fourth register marks the current count value of the timer/counter interrupt. The count value can be set either by the instruction block (set automatically) or by the user.

- **Get Mode**
  Instruction block sets the current count
- **Set Mode**
  User sets the counter/timer

DCTR: Down Counter

33

At a Glance

Introduction

This chapter describes the instruction DCTR.

What's in this Chapter?

This chapter contains the following topics:

Short Description

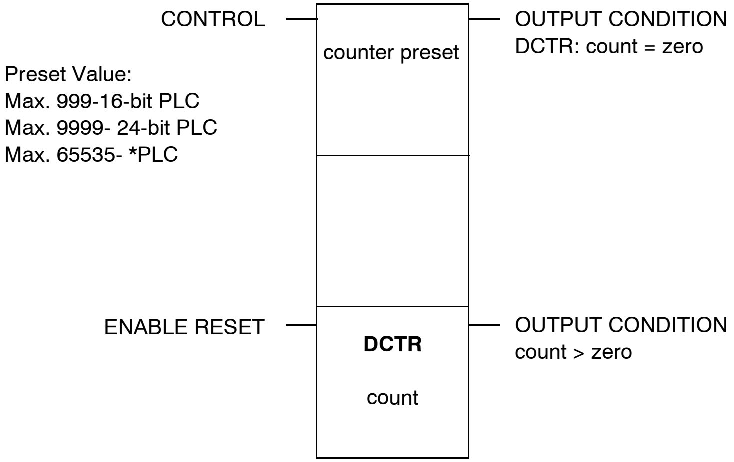
Function Description

The DCTR instruction counts control input transitions from OFF to ON down from a counter preset value to zero.

Representation: DCTR - Down Counter

Symbol

Representation of the instruction



*Available on the following
- E685/785 PLCs
- L785 PLCs
- Quantum Series PLCs

Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	OFF → ON = initiates the counter operation
Bottom input	0x, 1x	None	OFF = accumulated count is reset to preset value ON = counter accumulating
counter preset (top node)	3x, 4x	INT, UINT	Preset value, can be displayed explicitly as an integer (range 1 ... 65 535) or stored in a register Preset Value: Max. 999 - 16-bit PLC Max. 9999 - 24-bit PLC Max. 65535 - *PLC
accumulated count (bottom node)	4x	INT, UINT	Count value (actual value); which decrements by one on each transition from OFF to ON of the top input until it reaches zero.
Top output	0x	None	ON = accumulated count = 0
Bottom output	0x	None	ON = accumulated count > 0

DIOH: Distributed I/O Health

34

At a Glance

Introduction

This chapter describes the instruction DIOH.

What's in this Chapter?

This chapter contains the following topics:

Short Description

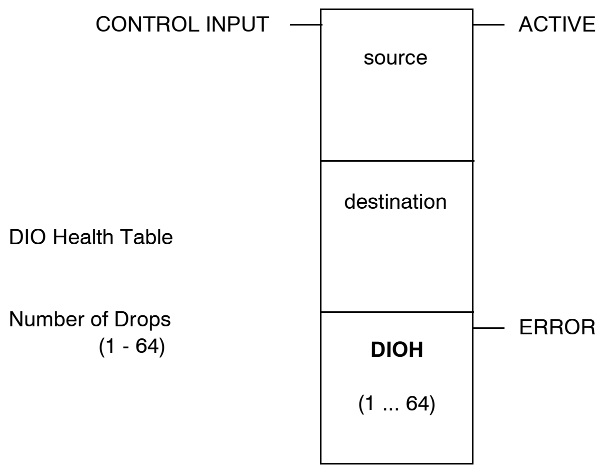
Function Description

The DIOH instruction lets you retrieve health data from a specified group of drops on the distributed I/O network. It accesses the DIO health status table, where health data for modules in up to 189 distributed drops is stored.

Representation: DIOH - Distributed I/O Health

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = initiates the retrieval of the specified status words from the DIO health table into the destination table
source (top node)		INT, UINT	The *source* value entered in the top node is a four-digit constant in the form *xxyy*, where: • *xx* is a decimal value in the range 00 ... 16, indicating the slot number in which the relevant DIO processor resides. The value 00 can always be used to indicate the Modbus Plus ports on the PLC, regardless of the slot in which it resides. • *yy* is a decimal value in the range 1 ... 64, indicating the drop number on the appropriate token ring. For example, if you are interested in retrieving drop status starting at distributed drop #1 on a network being handled by a DIO processor in slot 3, enter 0301 in the top node.
destination (middle node)	4x	INT, UINT, WORD	First holding register in the destination table, i.e. in a block of contiguous registers where the retrieved health status information is stored
length (bottom node)		INT, UINT	Length of the destination table, range 1 ... 64
Top output	0x	None	Echoes the state of the top input
Bottom output	0x	None	ON = invalid source entry

Parameter Description

Source Value (Top Node)

The source value entered in the top node is a four-digit constant in the form **`xxyy`**, where:

Digits	Meaning
`xx`	Decimal value in the range 00 ... 16, indicating the slot number in which the relevant DIO processor resides. The value 00 can always be used to indicate the Modbus Plus ports on the PLC, regardless of the slot in which it resides.
`yy`	Decimal value in the range 1 ... 64, indicating the drop number on the appropriate token ring

For example, if you are interested in retrieving drop status starting at distributed drop #1 on a network being handled by a DIO processor in slot 3, enter **`0301`** in the top node.

Length of Destination Table (Bottom Node)

The integer value entered in the bottom node specifies the length, i.e. the number of 4x registers, in the destination table. The length is in the range 1 ... 64.

> **Note:** If you specify a length that excedes the number of drops available, the instruction will return status information only for the drops available. For example, if you specify the 63rd drop number (yy) in the top node register and then request a length of 5, the instruction will give you only two registers (the 63rd and 64th drop status words) in the destination table.

DISA - Disabled Discrete Monitor

35

At A Glance

Introduction

This chapter describes the instruction DISA.

What's in this Chapter?

This chapter contains the following topics:

Short Description: DISA - Disabled Discrete Monitor

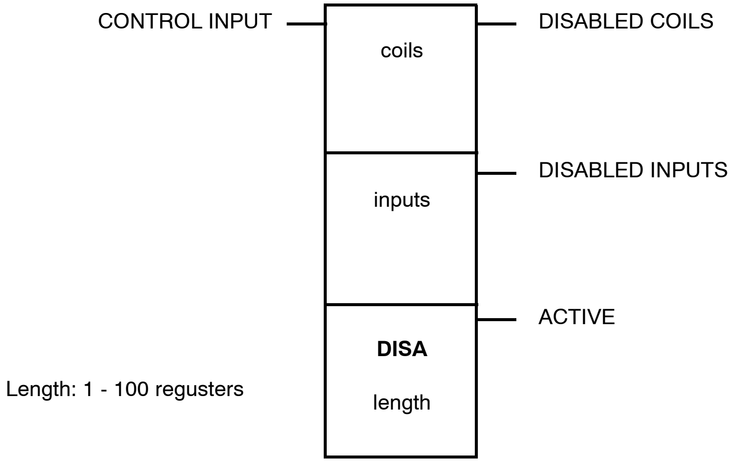
Function

The Disabled Discrete Monitor (DISA) is a loadable function, an instruction that monitors disabled coils and inputs. Therefore, DISA monitors the disabled states of all 0xxxx and 1xxxx addresses.

Representation: DISA - Disabled Discrete Monitor

Symbol

Representation of the instruction



Note: The NSUP loadable must be loaded prior to loading the DISA loadable.

Parameter Description

Description of the instruction’s parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	Disabled coils table
coils (top node)	4x	INT, UINT	Number of disabled coils found (even if > NNN)
	4x+#	INT, UINT	Address of ‘#’ disabled coil found
inputs (middle node)	4y	INT, UINT	Number of disabled input discretes found (even if > NNN)
	4y+#	INT, UINT	Address of ‘#’’ disabled discrete found
length (bottom node)		INT, UINT	Passes power when top input receives power
Top output	0x	None	ON if disabled coils are found
Middle output	0x	None	ON if disabled inputs are found
Bottom output	0x	None	Echoes state of top input

DIV: Divide

36

At a Glance

Introduction

This chapter describes the instruction DIV.

What's in this Chapter?

This chapter contains the following topics:

Short Description

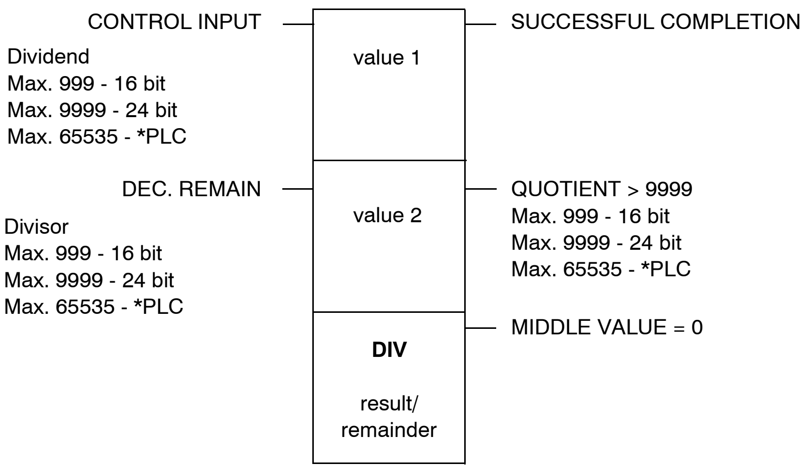
Function Description

The DIV instruction divides unsigned value 1 (its top node) by unsigned value 2 (its middle node) and posts the quotient and remainder in two contiguous holding registers in the bottom node.

Representation: DIV - Single Precision Division

Symbol

Representation of the instruction



*Available on the following
- E685/785 PLCs
- L785 PLCs
- Quantum Series PLCs

Parameter Description

Description of the instruction’s parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = value 1 divided by value 2
Middle input	0x, 1x	None	ON = decimal remainder OFF = fraction remainder
value 1 (top node)	3x, 4x	INT, UINT	Dividend, can be displayed explicitly as an integer (range 1 ... 9999)* or stored in two contiguous registers (displayed for high-order half, implied for low-order half) *Max. 999 - 16 bit Max. 9999 - 24 bit Max. 65535 - *PLC (See availability list above.)
value 2 (middle node)	3x, 4x	INT, UINT	Divisor, can be displayed explicitly as an integer (range 1 ... 9999) or stored in a register *Max. 999 - 16 bit Max. 9999 - 24 bit Max. 65535 - *PLC (See availability list above.)
result / remainder (bottom node)	4x	INT, UINT	First of two contiguous holding registers: displayed: result of division implied: remainder (either a decimal or a fraction, depending on the state of middle input)
Top output	0x	None	ON = division successful
Middle output	0x	None	ON = overflow: if result > 9999*, a 0 value is returned *Max. 999 - 16 bit Max. 9999 - 24 bit Max. 65535 - *PLC (See availability list above.)
Bottom output	0x	None	ON = value 2 = 0

Example

Quotient of Instruction DIV

The state of the middle input indicates whether the remainder will be expressed as a decimal or as a fraction. For example, if value 1 = 8 and value 2 = 3, the decimal remainder (middle input ON) is 6666; the fractional remainder (middle input OFF) is 2.

DLOG: Data Logging for PCMCIA Read/Write Support

37

At a Glance

Introduction

This chapter describes the instruction DLOG.

What's in this Chapter?

This chapter contains the following topics:

Short Description

Function Description

Note: This instruction is only available with the PLC family TSX Compact.

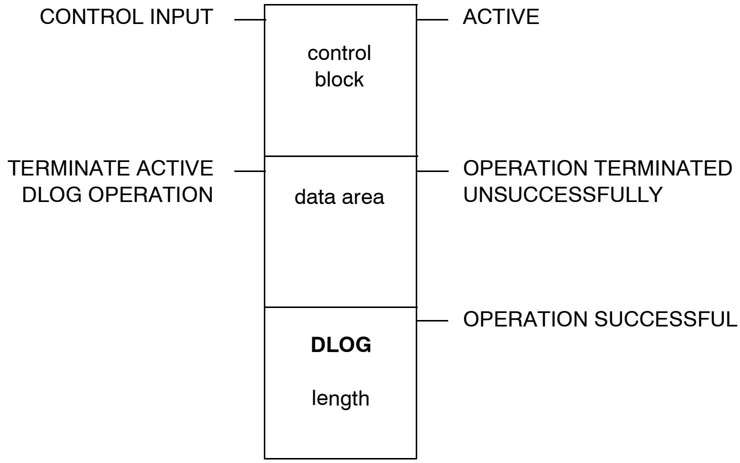
PCMCIA read and write support consists of a configuration extension to be implemented using a DLOG instruction. The DLOG instruction provides the facility for an application to copy data to a PCMCIA flash card, copy data from a PCMCIA flash card, erase individual memory blocks on a PCMCIA flash card, and to erase an entire PCMCIA flash card. The data format and the frequency of data storage are controlled by the application.

Note: The DLOG instruction will only operate with PCMCIA linear flash cards that use AMD flash devices.

Representation: DLOG

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = DLOG operation enabled, it should remain ON until the operation has completed successfully or an error has occurred.
Middle input	0x, 1x	None	ON = stops the currently active operation
control block (top node)	4x	INT, UINT	First of five contiguous registers in the DLOG control block (For expanded and detailed information please see *Control Block (Top Node), p. 238*.)
data area (middle node)	4x	INT, UINT	First 4x register in a data area used for the source or destination of the specified operation (For expanded and detailed information please see *Data Area (Middle Node), p. 239*.)
length (bottom node)		INT, UINT	Maximum number of registers reserved for the data area, range: 0 ... 100.
Top output	0x	None	Echoes state of the top input
Middle output	0x	None	ON = error during DLOG operation (operation terminated unsuccessfully)
Bottom output	0x	None	ON = DLOG operation finishes successfully (operation successful)

Parameter Description

Control Block (Top Node)

The 4x register entered in the top node is the first of five contiguous registers in the DLOG control block.
The control block defines the function of the DLOG command, the PCMCIA flash card window and offset, a return status word, and a data word count value.

Register	Function	Content
Displayed	Error Status	Displays DLOG errors in HEX values
First implied	Operation Type	1 = Write to PCMCIA Card 2 = Read to PCMCIA Card 3 = Erase One Block 4 = Erase Entire Card Content
Second implied	Window (Block Identifier)	This register identifies a particular block (PCMCIA memory window) located on the PCMCIA card (1 block=128k bytes) The number of blocks are dependent on the memory size of the PCMCIA card. (e.g.. 0 ... 31 Max. for a 4Meg PCMCIA card).
Third implied	Offset (Byte Address within the Block)	Particular range of bytes located within a particular block on the PCMCIA card. Range: 1 ... 128k bytes
Fourth implied	Count	Number of 4x registers to be written or read to the PCMCIA card. Range: 0 ... 100.

Note: PCMCIA Flash Card address are address on a Window:Offset basis. Windows have a set size of 128k bytes (65 535 words (16-bit values)). No Write or Read operation can cross the boundary from one window to the next. Therefore, offset (third implied register) plus length (fourth implied register) must always be less or equal to 128k bytes (65 535 words).

Data Area (Middle Node)

The 4x register entered in the middle node is the first register in a contiguous block of 4x word registers, that the DLOG instruction will use for the source or destination of the operation specified in the top node's control block.

Operation	State Ram Reference	Function
Write	4x	Source Address
Read	4x	Destination Address
Erase Block	none	None
Erase Card	none	None

Length (Bottom Node)

The integer value entered in the bottom node is the length of the data area, i.e., the maximum number of words (registers) allowed in a transfer to/from the PCMCIA flash card. The length can range from 0 ... 100.

Run Time Error Handling

Error Codes

The displayed register of the control block contains the following DLOG errors in Hex-code.

Error Code in Hex	Content
1	The count parameter of the control block > the DLOG block length during a WRITE operation (01)
2	PCMCIA card operation failed when intially started (write/read/erase)
3	PCMCIA card operation failed during execution (write/read/erase)

DMTH - Double Precision Math

38

At a Glance

Introduction

This chapter describes the four double precision math operations executed by the instruction DMTH. The four operations are Addition, Subtraction, Multiplication, and Division.

What's in this Chapter?

This chapter contains the following topics:

Short Description: DMTH - Double Precision Math - Addition, Subtraction, Multiplication, and Division

Function Description

The Double Precision Math (DMTH) instruction performs double precision addition, subtraction, multiplication, or division (set by bottom node). DMTH uses 2 registers appended together to form one operand.

Each DMTH instruction operates on the same two operands.

- **OP1** = 4x, 4x + 1 (top node)
- **OP2** = 4y, 4y + 1 (middle node)

Function Codes

The DMTH instruction performs any one of four possible double precision math operations. DMTH performs the operation by calling a function. To call the desired function enter a function code in the bottom node. Function codes range from 1 ... 4.

Code	DMTH Function	Function Performed	Result Registers
1	Double Precision Addition	Add (OP1) + (OP 2)	(4y + 3, 4y + 4)
2	Double Precision Subtraction	Subtract (OP1) - (OP 2)	(4y + 2, 4y + 3)
3	Double Precision Multiplication	Multiply (OP1) * (OP 2)	(4y + 2, 4y + 3)
			(4y + 4, 4y + 5)
4	Double Precision Division	Divide (OP1)\(OP 2)	(4y + 2, 4y + 3) quotient
			(4y + 4, 4y + 5) remainder

Notes

- For numbers spread over more than one register, the least significant 4 digits are stored in the highest holding register.
- Results, flags, and remainders are stored in the registers following **OP2**.
- Registers not used by the chosen math function may be used for other purposes.
- The Subtract Function uses the outputs to indicate the result of comparison between Operands **OP1** and **OP2**.

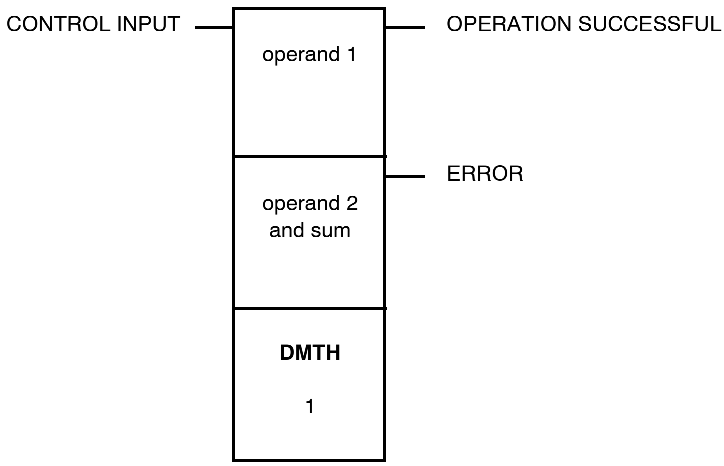
Representation: DMTH - Double Precision Math - Addition, Subtraction, Multiplication, and Division

Explanation of This Section

This section describes the Addition, Subtraction, Multiplication, and Division operations, which are the four operations performed by the instruction DMTH. Each operations has a Symbol, which is a graphical representation of the instruction, and a Parameter Description, which is a table-format representation of the instruction.

Symbol - Addition

Representation of the instruction for the Addition operation



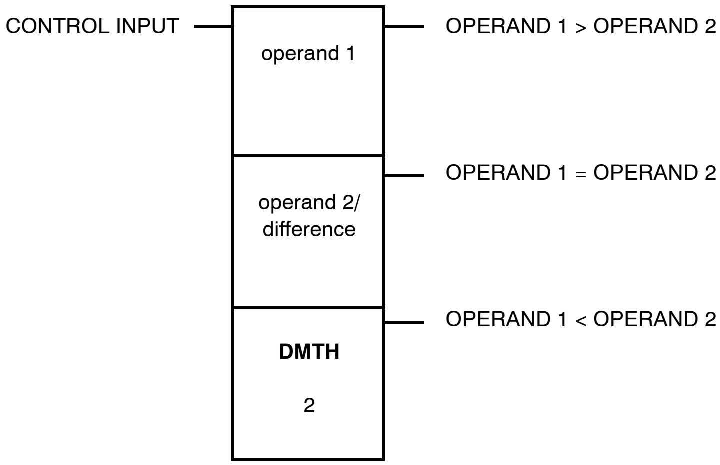
Parameter Description - Addition

Description of the instruction's parameters for the Addition operation

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON adds operands and posts sum in designated registers.
operand 1 (top node)	4x	INT, UINT	The first of two contiguous 4xxxx registers is entered in the top node. The second 4xxxx register is implied. Operand 1 is stored here. Each register holds a value in the range 0000 through 9999, for a combined double precision value in the range 0 through 99,999,999. The high-order half of operand 1 is stored in the displayed register, and the low-order half is stored in the implied register.
operand 2 and sum (middle node)	4x	INT, UINT	The first of six contiguous 4x registers is entered in the middle node. The remaining five registers are implied: ● The displayed register and the first implied register store the high-order and low-order halves of operand 2, respectively, for a combined double precision value in the range 0 through 99,999,999 ● The value stored in the second implied register indicates whether an overflow condition exists (a value of 1 = overflow) ● The third and fourth implied registers store the high-order and low-order halves of the double precision sum, respectively ● The fifth implied register is not used in the calculation but must exist in state RAM
Top output	0x	None	ON = operation successful
Middle output	0x	None	On = operand out of range or invalid

Symbol - Subtraction

Representation of the instruction for the Subtraction operation



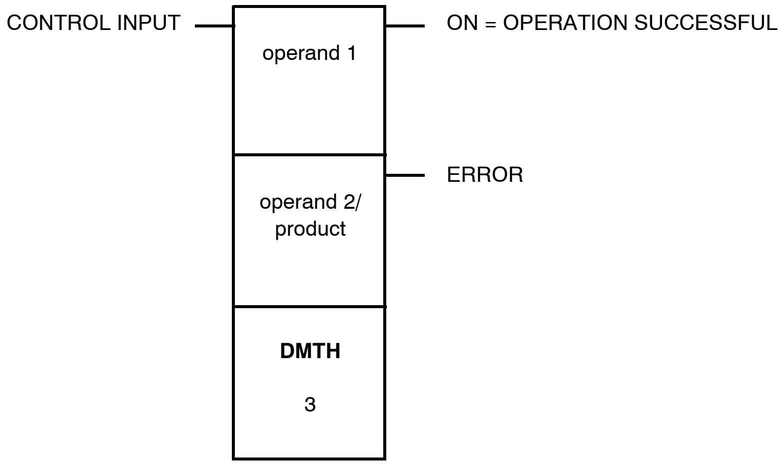
Parameter Description - Subtraction

Description of the instruction’s parameters for the Subtraction operation

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON subtracts operand 2 from operand 1 and posts difference in designated registers.
operand 1 (top node)	4x	INT, UINT	The first of two contiguous 4xxxx registers is entered in the top node. The second 4xxxx register is implied. Operand 1 is stored here. Each register holds a value in the range 0000 through 9999, for a combined double precision value in the range 0 through 99,999,999. The high-order half of operand 1 is stored in the displayed register, and the low-order half is stored in the implied register.
operand 2 difference (middle node)	4x	INT, UINT	The first of six contiguous 4xxxx registers is entered in the middle node. The remaining five registers are implied: • The displayed register and the first implied register store the high-order and low-order halves of operand 2, respectively, for a combined double precision value in the range 0 through 99,999,999 • The value stored in the second implied register indicates whether an overflow condition exists (a value of 1 = overflow) • The third and fourth implied registers store the high-order and low-order halves of the double precision sum, respectively • The fifth implied register is not used in the calculation but must exist in state RAM
Top output	0x	None	ON = operand 1 > operand 2
Middle output	0x	None	ON = operand 1 = operand 2
Bottom output	0x	None	ON = operand 1 < operand 2

Symbol - Multiplication

Representation of the instruction for the Multiplication operation



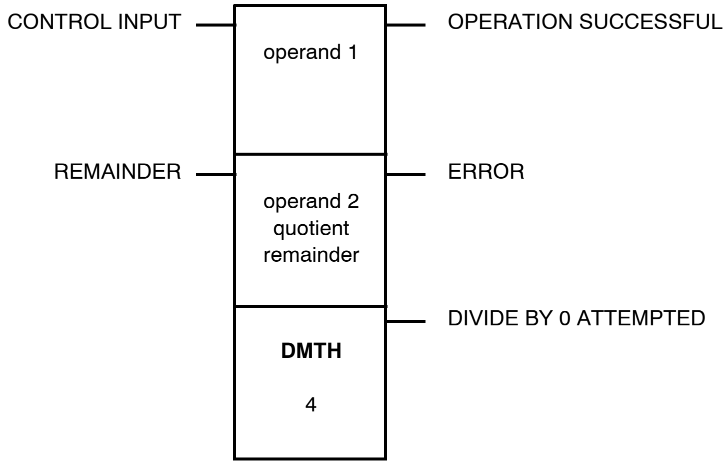
Parameter Description - Multiplication

Description of the instruction's parameters for the Multiplication operation

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = operand 1 x operand 2 and product posted in designated registers.
operand 1 (top node)	4x	INT, UINT	The first of two contiguous 4xxxx registers is entered in the top node. The second 4xxxx register is implied. Operand 1 is stored here. The second 4x register is implied. Each register holds a value in the range 0000 through 9999, for a combined double precision value in the range 0 through 99,999,999. The high-order half of operand 1 is stored in the displayed register, and the low-order half is stored in the implied register.
operand 2/ product (middle node)	4x	INT, UINT	The first of six contiguous 4xxxx registers is entered in the middle node. The remaining five registers are implied: ● The displayed register and the first implied register store the high-order and low-order halves of operand 2, respectively, for a combined double precision value in the range 0 through 99,999,999 ● The last four implied registers store the double precision product in the range 0 through 9,999,999,999,999,999
Top output	0x	None	ON = operation successful
Middle output	0x	None	ON = operand out of range

Symbol - Division

Representation of the instruction for the Division operation



Parameter Description - Division

Description of the instruction's parameters for the Division operation

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = operand 1 divided by operand 2 and result posted in designated registers.
Middle input	0x, 1x	None	ON = decimal remainder OFF = fractional remainder
operand 1 (top node)	4x	INT, UINT	The first of two contiguous 4xxxx registers is entered in the top node. The second 4xxxx register is implied. Operand 1 is stored here. The second 4x register is implied. Each register holds a value in the range 0000 through 9999, for a combined double precision value in the range 0 through 99,999,999. The high-order half of operand 1 is stored in the displayed register, and the low-order half is stored in the implied register.
operand 2 quotient remainder (middle node)	4x	INT, UINT	The first of six contiguous 4x registers is entered in the middle node. The remaining five registers are implied: ● The displayed register and the first implied register store the high-order and low-order halves of operand 2, respectively, for a combined double precision value in the range 0 through 99,999,999 **Note:** Since division by 0 is illegal, a 0 value causes an error; an error trapping routine sets the remaining middle-node registers to 0000 and turns the bottom output ON. ● The second and third implied registers store an eight-digit quotient ● The fourth and fifth implied registers store the remainder. If the remainder is expressed as a fraction, it is eight digits long and both registers are used, if the remainder is expressed as a decimal, it is four digits long and only the fourth implied register is used
Top output	0x	None	ON = operation successful
Middle output	0x	None	ON = an operand out of range
Bottom output	0x	None	On = operand 2 is 0

DRUM: DRUM Sequencer

39

At a Glance

Introduction

This chapter describes the instruction DRUM.

What's in this Chapter?

This chapter contains the following topics:

Short Description

Function Description

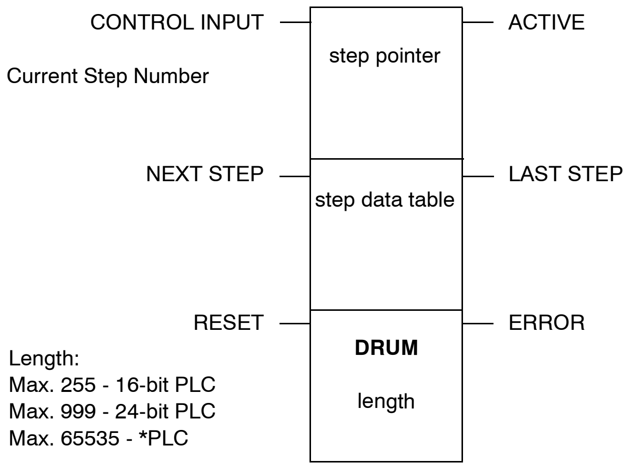
> **Note:** This instruction is only available if you have unpacked and installed the DX Loadables. For further information, see *Installation of DX Loadables, p. 109*."

The DRUM instruction operates on a table of 4x registers containing data representing each step in a sequence. The number of registers associated with this step data table depends on the number of steps required in the sequence. You can pre-allocate registers to store data for each step in the sequence, thereby allowing you to add future sequencer steps without having to modify application logic. DRUM incorporates an output mask that allows you to selectively mask bits in the register data before writing it to coils. This is particularly useful when all physical sequencer outputs are not contiguous on the output module. Masked bits are not altered by the DRUM instruction, and may be used by logic unrelated to the sequencer.

Representation: DRUM

Symbol

Representation of the instruction



*Available on the following
- E685/785 PLCs
- L785 PLCs
- Quantum Series PLCs

Parameter Description

Description of the instruction's parameters

Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = initiates DRUM sequencer
Middle input	0x, 1x	None	ON = step pointer increments to next step
Bottom input	0x, 1x	None	ON = reset step pointer to 0
step pointer (top node)	4x	INT, UINT	Current step number
step data table (middle node)	4x	INT, UINT	First register in a table of step data information (For expanded and detailed information please see *Step Data Table (Middle Node), p. 254*.)
length (bottom node)		INT, UINT	Number of application-specific registers used in the step data table, range: 1 .. 999 Length:Max. 255 - 16-bit PLC Max. 999 - 24-bit PLC Max. 65535 - *PLC
Top output	0x	None	Echoes state of the top input
Middle output	0x	None	ON = step pointer value = length
Bottom output	0x	None	ON = Error

Parameter Description

Step Pointer (Top Node)

The 4x register entered in the top node stores the current step number. The value in this register is referenced by the DRUM instruction each time it is solved. If the middle input to the block is ON, the contents of the register in the top node are incremented to the next step in the sequence before the block is solved.

Step Data Table (Middle Node)

The 4x register entered in the middle node is the first register in a table of step data information.
The first six registers in the step data table hold constant and variable data required to solve the block:

Register	Name	Content
Displayed	masked output data	Loaded by DRUM each time the block is solved; contains the contents of the current step data register masked with the outputmask register
First implied	current step data	Loaded by DRUM each time the block is solved; contains data from the step pointer, causes the block logic to automatically calculate register offsets when accessing step data in the step data table
Second implied	output mask	Loaded by user before using the block, DRUM will not alter output mask contents during logic solve; contains a mask to be applied to the data for each sequencer step
Third implied	machine ID number	Identifies DRUM/ICMP blocks belonging to a specific machine configuration; value range: 0 ... 9 999 (0 = block not configured); all blocks belonging to same machine configuration have the same machine ID number
Fourth implied	profile ID number	Identifies profile data currently loaded to the sequencer; value range: 0... 9 999 (0 = block not configured); all blocks with the same machine ID number must have the same profile ID number
Fifth implied	steps used	Loaded by user before using the block, DRUM will not alter steps used contents during logic solve; contains between 1 ... 999 for 24 bit CPUs, specifying the actual number of steps to be solved; the number must be greater or less than the table length in the bottom node

The remaining registers contain data for each step in the sequence.

Length (Bottom Node)

The integer value entered in the bottom node is the length, i.e., the number of application-specific registers used in the step data table. The length can range from 1 ... 999 in a 24-bit CPU.
The total number of registers required in the step data table is the length + 6. The length must be greater or equal to the value placed in the steps used register in the middle node.

DV16: Divide 16 Bit

40

At a Glance

Introduction

This chapter describes the instruction DV16.

What's in this Chapter?

This chapter contains the following topics:

Short Description

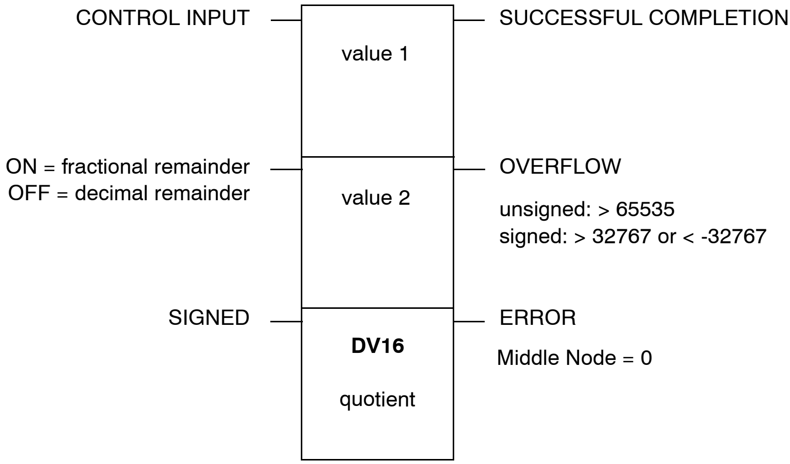
Function Description

The DV16 instruction performs a signed or unsigned division on the 16-bit values in the top and middle nodes (value 1 / value 2), then posts the quotient and remainder in two contiguous 4x holding registers in the bottom node.

Representation: DV16 - 16-bit Division

Symbol

Representation of the instruction



Parameter Description

Description of the instruction's parameters

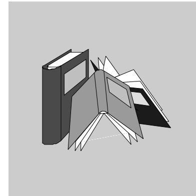
Parameters	State RAM Reference	Data Type	Meaning
Top input	0x, 1x	None	ON = enables value 1 / value 2
Middle input	0x, 1x	None	OFF = decimal remainder ON = fractional remainder
Bottom input	0x, 1x	None	ON = signed operation OFF = unsigned operation
value 1 (top node)	3x, 4x	INT, UINT	Dividend, can be displayed explicitly as an integer (range 1 ... 65 535) or stored in two contiguous registers (displayed for high-order half, implied for low-order half)
value 2 (middle node)	3x, 4x	INT, UINT	Divisor, can be displayed explicitly as an integer (range 1 ... 65 535, enter e.g. #65535) or stored in a register
quotient (bottom node)	4x	INT, UINT	First of two contiguous holding registers: displayed: result of division implied: remainder (either a decimal or a fraction, depending on the state of middle input)
Top output	0x	None	ON = Divide operation completed successfully
Middle output	0x	None	ON = overflow: quotient > 65 535 in unsigned operation -32 768 > quotient > 32 767 in signed operation
Bottom output	0x	None	ON = value 2 = 0

Example

Quotient of Instruction DV16

The state of the middle input indicates whether the remainder will be expressed as a decimal or as a fraction. For example, if value 1 = 8 and value 2 = 3, the decimal remainder (middle input OFF) is 6666; the fractional remainder (middle input ON) is 2.

Glossary



A

active window

The window, which is currently selected. Only one window can be active at any one given time. When a window is active, the heading changes color, in order to distinguish it from other windows. Unselected windows are inactive.

Actual parameter

Currently connected Input/Output parameters.

Addresses

(Direct) addresses are memory areas on the PLC. These are found in the State RAM and can be assigned input/output modules.
The display/input of direct addresses is possible in the following formats:
- Standard format (400001)
- Separator format (4:00001)
- Compact format (4:1)
- IEC format (QW1)

ANL_IN

ANL_IN stands for the data type "Analog Input" and is used for processing analog values. The 3x References of the configured analog input module, which is specified in the I/O component list is automatically assigned the data type and should therefore only be occupied by Unlocated variables.

ANL_OUT

ANL_OUT stands for the data type "Analog Output" and is used for processing analog values. The 4x-References of the configured analog output module, which is specified in the I/O component list is automatically assigned the data type and should therefore only be occupied by Unlocated variables.

ANY

In the existing version "ANY" covers the elementary data types BOOL, BYTE, DINT, INT, REAL, UDINT, UINT, TIME and WORD and therefore derived data types.

ANY_BIT In the existing version, "ANY_BIT" covers the data types BOOL, BYTE and WORD.

ANY_ELEM In the existing version "ANY_ELEM" covers the elementary data types BOOL, BYTE, DINT, INT, REAL, UDINT, UINT, TIME and WORD.

ANY_INT In the existing version, "ANY_INT" covers the data types DINT, INT, UDINT and UINT.

ANY_NUM In the existing version, "ANY_NUM" covers the data types DINT, INT, REAL, UDINT and UINT.

ANY_REAL In the existing version "ANY_REAL" covers the data type REAL.

Application window The window, which contains the working area, the menu bar and the tool bar for the application. The name of the application appears in the heading. An application window can contain several document windows. In Concept the application window corresponds to a Project.

Argument Synonymous with Actual parameters.

ASCII mode American Standard Code for Information Interchange. The ASCII mode is used for communication with various host devices. ASCII works with 7 data bits.

Atrium The PC based controller is located on a standard AT board, and can be operated within a host computer in an ISA bus slot. The module occupies a motherboard (requires SA85 driver) with two slots for PC104 daughter boards. From this, a PC104 daughter board is used as a CPU and the others for INTERBUS control.

B

Back up data file (Concept EFB) The back up file is a copy of the last Source files. The name of this back up file is "backup??.c" (it is accepted that there are no more than 100 copies of the source files. The first back up file is called "backup00.c". If changes have been made on the Definition file, which do not create any changes to the interface in the EFB, there is no need to create a back up file by editing the source files (**Objects** → **Source**). If a back up file can be assigned, the name of the source file can be given.

Base 16 literals Base 16 literals function as the input of whole number values in the hexadecimal system. The base must be denoted by the prefix 16#. The values may not be preceded by signs (+/-). Single underline signs (_) between figures are not significant.

Example
16#F_F or 16#FF (decimal 255)
16#E_0 or 16#E0 (decimal 224)

Base 8 literal
Base 8 literals function as the input of whole number values in the octal system. The base must be denoted by the prefix 3.63kg. The values may not be preceded by signs (+/-). Single underline signs (_) between figures are not significant.

Example
8#3_1111 or 8#377 (decimal 255)
8#34_1111 or 8#340 (decimal 224)

Basis 2 literals
Base 2 literals function as the input of whole number values in the dual system. The base must be denoted by the prefix 0.91kg. The values may not be preceded by signs (+/-). Single underline signs (_) between figures are not significant.

Example
2#1111_1111 or 2#11111111 (decimal 255)
2#1110_1111 or 2#11100000 (decimal 224)

Binary connections
Connections between outputs and inputs of FFBs of data type BOOL.

Bit sequence
A data element, which is made up from one or more bits.

BOOL
BOOL stands for the data type "Boolean". The length of the data elements is 1 bit (in the memory contained in 1 byte). The range of values for variables of this type is 0 (FALSE) and 1 (TRUE).

Bridge
A bridge serves to connect networks. It enables communication between nodes on the two networks. Each network has its own token rotation sequence – the token is not deployed via bridges.

BYTE
BYTE stands for the data type "Bit sequence 8". The input appears as Base 2 literal, Base 8 literal or Base 1 16 literal. The length of the data element is 8 bit. A numerical range of values cannot be assigned to this data type.

C

Cache
The cache is a temporary memory for cut or copied objects. These objects can be inserted into sections. The old content in the cache is overwritten for each new Cut or Copy.

Call up The operation, by which the execution of an operation is initiated.

Coil A coil is a LD element, which transfers (without alteration) the status of the horizontal link on the left side to the horizontal link on the right side. In this way, the status is saved in the associated Variable/ direct address.

Compact format (4:1) The first figure (the Reference) is separated from the following address with a colon (:), where the leading zero are not entered in the address.

Connection A check or flow of data connection between graphic objects (e.g. steps in the SFC editor, Function blocks in the FBD editor) within a section, is graphically shown as a line.

Constants Constants are Unlocated variables, which are assigned a value that cannot be altered from the program logic (write protected).

Contact A contact is a LD element, which transfers a horizontal connection status onto the right side. This status is from the Boolean AND- operation of the horizontal connection status on the left side with the status of the associated Variables/direct Address. A contact does not alter the value of the associated variables/direct address.

D

Data transfer settings Settings, which determine how information from the programming device is transferred to the PLC.

Data types The overview shows the hierarchy of data types, as they are used with inputs and outputs of Functions and Function blocks. Generic data types are denoted by the prefix "ANY".

- ANY_ELEM
  - ANY_NUM
    ANY_REAL (REAL)
    ANY_INT (DINT, INT, UDINT, UINT)
  - ANY_BIT (BOOL, BYTE, WORD)
  - TIME
- System data types (IEC extensions)
- Derived (from "ANY" data types)

DCP I/O station With a Distributed Control Processor (D908) a remote network can be set up with a parent PLC. When using a D908 with remote PLC, the parent PLC views the remote PLC as a remote I/O station. The D908 and the remote PLC communicate via the system bus, which results in high performance, with minimum effect on the cycle time. The data exchange between the D908 and the parent PLC takes place at 1.5 Megabits per second via the remote I/O bus. A parent PLC can support up to 31 (Address 2-32) D908 processors.

DDE (Dynamic Data Exchange) The DDE interface enables a dynamic data exchange between two programs under Windows. The DDE interface can be used in the extended monitor to call up its own display applications. With this interface, the user (i.e. the DDE client) can not only read data from the extended monitor (DDE server), but also write data onto the PLC via the server. Data can therefore be altered directly in the PLC, while it monitors and analyzes the results. When using this interface, the user is able to make their own "Graphic-Tool", "Face Plate" or "Tuning Tool", and integrate this into the system. The tools can be written in any DDE supporting language, e.g. Visual Basic and Visual-C++. The tools are called up, when the one of the buttons in the dialog box extended monitor uses Concept Graphic Tool: Signals of a projection can be displayed as timing diagrams via the DDE connection between Concept and Concept Graphic Tool.

Decentral Network (DIO) A remote programming in Modbus Plus network enables maximum data transfer performance and no specific requests on the links. The programming of a remote net is easy. To set up the net, no additional ladder diagram logic is needed. Via corresponding entries into the Peer Cop processor all data transfer requests are met.

Declaration Mechanism for determining the definition of a Language element. A declaration normally covers the connection of an Identifier with a language element and the assignment of attributes such as Data types and algorithms.

Definition data file (Concept EFB) The definition file contains general descriptive information about the selected FFB and its formal parameters.

Derived data type Derived data types are types of data, which are derived from the Elementary data types and/or other derived data types. The definition of the derived data types appears in the data type editor in Concept.
Distinctions are made between global data types and local data types.

Derived Function Block (DFB) A derived function block represents the Call up of a derived function block type. Details of the graphic form of call up can be found in the definition " Function block (Item)". Contrary to calling up EFB types, calling up DFB types is denoted by double vertical lines on the left and right side of the rectangular block symbol.

The body of a derived function block type is designed using FBD language, but only in the current version of the programming system. Other IEC languages cannot yet be used for defining DFB types, nor can derived functions be defined in the current version.
Distinctions are made between local and global DFBs.

DINT

DINT stands for the data type "double integer". The input appears as Integer literal, Base 2 literal, Base 8 literal or Base 16 literal. The length of the data element is 32 bit. The range of values for variables of this data type is from –2 exp (31) to 2 exp (31) –1.

Direct display

A method of displaying variables in the PLC program, from which the assignment of configured memory can be directly and indirectly derived from the physical memory.

Document window

A window within an Application window. Several document windows can be opened at the same time in an application window. However, only one document window can be active. Document windows in Concept are, for example, sections, the message window, the reference data editor and the PLC configuration.

Dummy

An empty data file, which consists of a text header with general file information, i.e. author, date of creation, EFB identifier etc. The user must complete this dummy file with additional entries.

DX Zoom

This property enables connection to a programming object to observe and, if necessary, change its data value.

E

Elementary functions/ function blocks (EFB)

Identifier for Functions or Function blocks, whose type definitions are not formulated in one of the IEC languages, i.e. whose bodies, for example, cannot be modified with the DFB Editor (Concept-DFB). EFB types are programmed in "C" and mounted via Libraries in precompiled form.

EN / ENO (Enable / Error display)
If the value of EN is "0" when the FFB is called up, the algorithms defined by the FFB are not executed and all outputs contain the previous value. The value of ENO is automatically set to "0" in this case. If the value of EN is "1" when the FFB is called up, the algorithms defined by the FFB are executed. After the error free execution of the algorithms, the ENO value is automatically set to "1". If an error occurs during the execution of the algorithm, ENO is automatically set to "0". The output behavior of the FFB depends whether the FFBs are called up without EN/ENO or with EN=1. If the EN/ENO display is enabled, the EN input must be active. Otherwise, the FFB is not executed. The projection of EN and ENO is enabled/disabled in the block properties dialog box. The dialog box is called up via the menu commands **Objects → Properties...** or via a double click on the FFB.

Error
When processing a FFB or a Step an error is detected (e.g. unauthorized input value or a time error), an error message appears, which can be viewed with the menu command **Online → Event display...** . With FFBs the ENO output is set to "0".

Evaluation
The process, by which a value for a Function or for the outputs of a Function block during the Program execution is transmitted.

Expression
Expressions consist of operators and operands.

F

FFB (functions/ function blocks)
Collective term for EFB (elementary functions/function blocks) and DFB (derived function blocks)

Field variables
Variables, one of which is assigned, with the assistance of the key word ARRAY (field), a defined Derived data type. A field is a collection of data elements of the same Data type.

FIR filter
Finite Impulse Response Filter

Formal parameters
Input/Output parameters, which are used within the logic of a FFB and led out of the FFB as inputs/outputs.

Function (FUNC)
A Program organization unit, which exactly supplies a data element when executing. A function has no internal status information. Multiple call ups of the same function with the same input parameter values always supply the same output values. Details of the graphic form of function call up can be found in the definition " Function block (Item)". In contrast to the call up of function blocks, the function call ups only have one unnamed output, whose name is the name of the function itself. In FBD each call up is denoted by a unique number over the graphic block; this number is automatically generated and cannot be altered.

Function block (item) (FB)
A function block is a Program organization unit, which correspondingly calculates the functionality values, defined in the function block type description, for the output and internal variables, when it is called up as a certain item. All output values and internal variables of a certain function block item remain as a call up of the function block until the next. Multiple call up of the same function block item with the same arguments (Input parameter values) supply generally supply the same output value(s).
Each function block item is displayed graphically by a rectangular block symbol. The name of the function block type is located on the top center within the rectangle. The name of the function block item is located also at the top, but on the outside of the rectangle. An instance is automatically generated when creating, which can however be altered manually, if required. Inputs are displayed on the left side and outputs on the right of the block. The names of the formal input/output parameters are displayed within the rectangle in the corresponding places.
The above description of the graphic presentation is principally applicable to Function call ups and to DFB call ups. Differences are described in the corresponding definitions.

Function block dialog (FBD)
One or more sections, which contain graphically displayed networks from Functions, Function blocks and Connections.

Function block type
A language element, consisting of: 1. the definition of a data structure, subdivided into input, output and internal variables, 2. A set of operations, which is used with the elements of the data structure, when a function block type instance is called up. This set of operations can be formulated either in one of the IEC languages (DFB type) or in "C" (EFB type). A function block type can be instanced (called up) several times.

Function counter
The function counter serves as a unique identifier for the function in a Program or DFB. The function counter cannot be edited and is automatically assigned. The function counter always has the structure: .n.m

n = Section number (number running)
m = Number of the FFB object in the section (number running)

G

Generic data type
A Data type, which stands in for several other data types.

Generic literal
If the Data type of a literal is not relevant, simply enter the value for the literal. In this case Concept automatically assigns the literal to a suitable data type.

Global derived data types
Global Derived data types are available in every Concept project and are contained in the DFB directory directly under the Concept directory.

Global DFBs
Global DFBs are available in every Concept project and are contained in the DFB directory directly under the Concept directory.

Global macros
Global Macros are available in every Concept project and are contained in the DFB directory directly under the Concept directory.

Groups (EFBs)
Some EFB libraries (e.g. the IEC library) are subdivided into groups. This facilitates the search for FFBs, especially in extensive libraries.

I

I/O component list
The I/O and expert assemblies of the various CPUs are configured in the I/O component list.

IEC 61131-3
International norm: Programmable controllers – part 3: Programming languages.

IEC format (QW1)
In the place of the address stands an IEC identifier, followed by a five figure address:
- %0x12345 = %Q12345
- %1x12345 = %I12345
- %3x12345 = %IW12345
- %4x12345 = %QW12345

IEC name conventions (identifier)

An identifier is a sequence of letters, figures, and underscores, which must start with a letter or underscores (e.g. name of a function block type, of an item or section). Letters from national sets of characters (e.g. ö,ü, é, õ) can be used, taken from project and DFB names.
Underscores are significant in identifiers; e.g. "A_BCD" and "AB_CD" are interpreted as different identifiers. Several leading and multiple underscores are not authorized consecutively.
Identifiers are not permitted to contain space characters. Upper and/or lower case is not significant; e.g. "ABCD" and "abcd" are interpreted as the same identifier.
Identifiers are not permitted to be Key words.

IIR filter

Infinite Impulse Response Filter

Initial step (starting step)

The first step in a chain. In each chain, an initial step must be defined. The chain is started with the initial step when first called up.

Initial value

The allocated value of one of the variables when starting the program. The value assignment appears in the form of a Literal.

Input bits (1x references)

The 1/0 status of input bits is controlled via the process data, which reaches the CPU from an entry device.

> **Note:** The x, which comes after the first figure of the reference type, represents a five figure storage location in the application data store, i.e. if the reference 100201 signifies an input bit in the address 201 of the State RAM.

Input parameters (Input)

When calling up a FFB the associated Argument is transferred.

Input words (3x references)

An input word contains information, which come from an external source and are represented by a 16 bit figure. A 3x register can also contain 16 sequential input bits, which were read into the register in binary or BCD (binary coded decimal) format. Note: The x, which comes after the first figure of the reference type, represents a five figure storage location in the user data store, i.e. if the reference 300201 signifies a 16 bit input word in the address 201 of the State RAM.

Instantiation

The generation of an Item.

Instruction (IL)

Instructions are "commands" of the IL programming language. Each operation begins on a new line and is succeeded by an operator (with modifier if needed) and, if necessary for each relevant operation, by one or more operands. If several operands are used, they are separated by commas. A tag can stand before the instruction, which is followed by a colon. The commentary must, if available, be the last element in the line.

Instruction (LL984)

When programming electric controllers, the task of implementing operational coded instructions in the form of picture objects, which are divided into recognizable contact forms, must be executed. The designed program objects are, on the user level, converted to computer useable OP codes during the loading process. The OP codes are deciphered in the CPU and processed by the controller's firmware functions so that the desired controller is implemented.

Instruction list (IL)

IL is a text language according to IEC 1131, in which operations, e.g. conditional/unconditional call up of Function blocks and Functions, conditional/unconditional jumps etc. are displayed through instructions.

INT

INT stands for the data type "whole number". The input appears as Integer literal, Base 2 literal, Base 8 literal or Base 16 literal. The length of the data element is 16 bit. The range of values for variables of this data type is from –2 exp (15) to 2 exp (15) –1.

Integer literals

Integer literals function as the input of whole number values in the decimal system. The values may be preceded by the signs (+/-). Single underline signs (_) between figures are not significant.

Example
-12, 0, 123_456, +986

INTERBUS (PCP)

To use the INTERBUS PCP channel and the INTERBUS process data preprocessing (PDP), the new I/O station type INTERBUS (PCP) is led into the Concept configurator. This I/O station type is assigned fixed to the INTERBUS connection module 180-CRP-660-01.
The 180-CRP-660-01 differs from the 180-CRP-660-00 only by a clearly larger I/O area in the state RAM of the controller.

Item name

An Identifier, which belongs to a certain Function block item. The item name serves as a unique identifier for the function block in a program organization unit. The item name is automatically generated, but can be edited. The item name must be unique throughout the Program organization unit, and no distinction is made between upper/lower case. If the given name already exists, a warning is given and another name must be selected. The item name must conform to the IEC name conventions, otherwise an error message appears. The automatically generated instance name always has the structure: FBI_n_m

FBI = Function block item
n = Section number (number running)
m = Number of the FFB object in the section (number running)

J

Jump Element of the SFC language. Jumps are used to jump over areas of the chain.

K

Key words Key words are unique combinations of figures, which are used as special syntactic elements, as is defined in appendix B of the IEC 1131-3. All key words, which are used in the IEC 1131-3 and in Concept, are listed in appendix C of the IEC 1131-3. These listed keywords cannot be used for any other purpose, i.e. not as variable names, section names, item names etc.

L

Ladder Diagram (LD) Ladder Diagram is a graphic programming language according to IEC1131, which optically orientates itself to the "rung" of a relay ladder diagram.

Ladder Logic 984 (LL) In the terms Ladder Logic and Ladder Diagram, the word Ladder refers to execution. In contrast to a diagram, a ladder logic is used by engineers to draw up a circuit (with assistance from electrical symbols),which should chart the cycle of events and not the existing wires, which connect the parts together. A usual user interface for controlling the action by automated devices permits ladder logic interfaces, so that when implementing a control system, engineers do not have to learn any new programming languages, with which they are not conversant.
The structure of the actual ladder logic enables electrical elements to be linked in a way that generates a control output, which is dependant upon a configured flow of power through the electrical objects used, which displays the previously demanded condition of a physical electric appliance.
In simple form, the user interface is one of the video displays used by the PLC programming application, which establishes a vertical and horizontal grid, in which the programming objects are arranged. The logic is powered from the left side of the grid, and by connecting activated objects the electricity flows from left to right.

Landscape format Landscape format means that the page is wider than it is long when looking at the printed text.

Language element Each basic element in one of the IEC programming languages, e.g. a Step in SFC, a Function block item in FBD or the Start value of a variable.

Library Collection of software objects, which are provided for reuse when programming new projects, or even when building new libraries. Examples are the Elementary function block types libraries.
EFB libraries can be subdivided into Groups.

Literals Literals serve to directly supply values to inputs of FFBs, transition conditions etc. These values cannot be overwritten by the program logic (write protected). In this way, generic and standardized literals are differentiated.
Furthermore literals serve to assign a Constant a value or a Variable an Initial value. The input appears as Base 2 literal, Base 8 literal, Base 16 literal, Integer literal, Real literal or Real literal with exponent.

Local derived data types Local derived data types are only available in a single Concept project and its local DFBs and are contained in the DFB directory under the project directory.

Local DFBs Local DFBs are only available in a single Concept project and are contained in the DFB directory under the project directory.

Local link The local network link is the network, which links the local nodes with other nodes either directly or via a bus amplifier.

Local macros Local Macros are only available in a single Concept project and are contained in the DFB directory under the project directory.

Local network nodes The local node is the one, which is projected evenly.

Located variable Located variables are assigned a state RAM address (reference addresses 0x,1x, 3x, 4x). The value of these variables is saved in the state RAM and can be altered online with the reference data editor. These variables can be addressed by symbolic names or the reference addresses.

Collective PLC inputs and outputs are connected to the state RAM. The program access to the peripheral signals, which are connected to the PLC, appears only via located variables. PLC access from external sides via Modbus or Modbus plus interfaces, i.e. from visualizing systems, are likewise possible via located variables.

M

Macro

Macros are created with help from the software Concept DFB.
Macros function to duplicate frequently used sections and networks (including the logic, variables, and variable declaration).
Distinctions are made between local and global macros.

Macros have the following properties:
- Macros can only be created in the programming languages FBD and LD.
- Macros only contain one single section.
- Macros can contain any complex section.
- From a program technical point of view, there is no differentiation between an instanced macro, i.e. a macro inserted into a section, and a conventionally created macro.
- Calling up DFBs in a macro
- Variable declaration
- Use of macro-own data structures
- Automatic acceptance of the variables declared in the macro
- Initial value for variables
- Multiple instancing of a macro in the whole program with different variables
- The section name, the variable name and the data structure name can contain up to 10 different exchange markings (@0 to @9).

MMI

Man Machine Interface

Multi element variables

Variables, one of which is assigned a Derived data type defined with STRUCT or ARRAY.
Distinctions are made between Field variables and structured variables.

N

Network

A network is the connection of devices to a common data path, which communicate with each other via a common protocol.

Network node

A node is a device with an address (164) on the Modbus Plus network.

Node address The node address serves a unique identifier for the network in the routing path. The address is set directly on the node, e.g. with a rotary switch on the back of the module.

O

Operand An operand is a Literal, a Variable, a Function call up or an Expression.

Operator An operator is a symbol for an arithmetic or Boolean operation to be executed.

Output parameters (Output) A parameter, with which the result(s) of the Evaluation of a FFB are returned.

Output/discretes (0x references) An output/marker bit can be used to control real output data via an output unit of the control system, or to define one or more outputs in the state RAM. Note: The x, which comes after the first figure of the reference type, represents a five figure storage location in the application data store, i.e. if the reference 000201 signifies an output or marker bit in the address 201 of the State RAM.

Output/marker words (4x references) An output/marker word can be used to save numerical data (binary or decimal) in the State RAM, or also to send data from the CPU to an output unit in the control system. Note: The x, which comes after the first figure of the reference type, represents a five figure storage location in the application data store, i.e. if the reference 400201 signifies a 16 bit output or marker word in the address 201 of the State RAM.

P

Peer processor The peer processor processes the token run and the flow of data between the Modbus Plus network and the PLC application logic.

PLC Programmable controller

Program The uppermost Program organization unit. A program is closed and loaded onto a single PLC.

Program cycle A program cycle consists of reading in the inputs, processing the program logic and the output of the outputs.

Program organization unit A Function, a Function block, or a Program. This term can refer to either a Type or an Item.

Programming device Hardware and software, which supports programming, configuring, testing, implementing and error searching in PLC applications as well as in remote system applications, to enable source documentation and archiving. The programming device could also be used for process visualization.

Programming redundancy system (Hot Standby) A redundancy system consists of two identically configured PLC devices, which communicate with each other via redundancy processors. In the case of the primary PLC failing, the secondary PLC takes over the control checks. Under normal conditions the secondary PLC does not take over any controlling functions, but instead checks the status information, to detect mistakes.

Project General identification of the uppermost level of a software tree structure, which specifies the parent project name of a PLC application. After specifying the project name, the system configuration and control program can be saved under this name. All data, which results during the creation of the configuration and the program, belongs to this parent project for this special automation.
General identification for the complete set of programming and configuring information in the Project data bank, which displays the source code that describes the automation of a system.

Project data bank The data bank in the Programming device, which contains the projection information for a Project.

Prototype data file (Concept EFB) The prototype data file contains all prototypes of the assigned functions. Further, if available, a type definition of the internal status structure is given.

R

REAL REAL stands for the data type "real". The input appears as Real literal or as Real literal with exponent. The length of the data element is 32 bit. The value range for variables of this data type reaches from 8.43E-37 to 3.36E+38.

> **Note:** Depending on the mathematic processor type of the CPU, various areas within this valid value range cannot be represented. This is valid for values nearing ZERO and for values nearing INFINITY. In these cases, a number value is not shown in animation, instead NAN (**N**ot **A N**umber) oder INF (**INF**inite).

Real literal

Real literals function as the input of real values in the decimal system. Real literals are denoted by the input of the decimal point. The values may be preceded by the signs (+/-). Single underline signs (_) between figures are not significant.

Example
-12.0, 0.0, +0.456, 3.14159_26

Real literal with exponent

Real literals with exponent function as the input of real values in the decimal system. Real literals with exponent are denoted by the input of the decimal point. The exponent sets the key potency, by which the preceding number is multiplied to get to the value to be displayed. The basis may be preceded by a negative sign (-). The exponent may be preceded by a positive or negative sign (+/-). Single underline signs (_) between figures are not significant. (Only between numbers, not before or after the decimal poiont and not before or after "E", "E+" or "E-")

Example
-1.34E-12 or -1.34e-12
1.0E+6 or 1.0e+6
1.234E6 or 1.234e6

Reference

Each direct address is a reference, which starts with an ID, specifying whether it concerns an input or an output and whether it concerns a bit or a word. References, which start with the code 6, display the register in the extended memory of the state RAM.
0x area = Discrete outputs
1x area = Input bits
3x area = Input words
4x area = Output bits/Marker words
6x area = Register in the extended memory

Note: The x, which comes after the first figure of each reference type, represents a five figure storage location in the application data store, i.e. if the reference 400201 signifies a 16 bit output or marker word in the address 201 of the State RAM.

Register in the extended memory (6x reference)

6x references are marker words in the extended memory of the PLC. Only LL984 user programs and CPU 213 04 or CPU 424 02 can be used.

RIO (Remote I/O)

Remote I/O provides a physical location of the I/O coordinate setting device in relation to the processor to be controlled. Remote inputs/outputs are connected to the consumer control via a wired communication cable.

RP (PROFIBUS)

RP = Remote Peripheral

RTU mode Remote Terminal Unit
The RTU mode is used for communication between the PLC and an IBM compatible personal computer. RTU works with 8 data bits.

Rum-time error Error, which occurs during program processing on the PLC, with SFC objects (i.e. steps) or FFBs. These are, for example, over-runs of value ranges with figures, or time errors with steps.

S

SA85 module The SA85 module is a Modbus Plus adapter for an IBM-AT or compatible computer.

Section A section can be used, for example, to describe the functioning method of a technological unit, such as a motor.
A Program or DFB consist of one or more sections. Sections can be programmed with the IEC programming languages FBD and SFC. Only one of the named programming languages can be used within a section.
Each section has its own Document window in Concept. For reasons of clarity, it is recommended to subdivide a very large section into several small ones. The scroll bar serves to assist scrolling in a section.

Separator format (4:00001) The first figure (the Reference) is separated from the ensuing five figure address by a colon (:).

Sequence language (SFC) The SFC Language elements enable the subdivision of a PLC program organizational unit in a number of Steps and Transitions, which are connected horizontally by aligned Connections. A number of actions belong to each step, and a transition condition is linked to a transition.

Serial ports With serial ports (COM) the information is transferred bit by bit.

Source code data file (Concept EFB) The source code data file is a usual C++ source file. After execution of the menu command **Library** → **Generate data files** this file contains an EFB code framework, in which a specific code must be entered for the selected EFB. To do this, click on the menu command **Objects** → **Source**.

Standard format (400001) The five figure address is located directly after the first figure (the reference).

Standardized literals

If the data type for the literal is to be automatically determined, use the following construction: 'Data type name'#'Literal value'.

Example
INT#15 (Data type: Integer, value: 15),
BYTE#00001111 (data type: Byte, value: 00001111)
REAL#23.0 (Data type: Real, value: 23.0)

For the assignment of REAL data types, there is also the possibility to enter the value in the following way: 23.0.
Entering a comma will automatically assign the data type REAL.

State RAM

The state RAM is the storage for all sizes, which are addressed in the user program via References (Direct display). For example, input bits, discretes, input words, and discrete words are located in the state RAM.

Statement (ST)

Instructions are "commands" of the ST programming language. Instructions must be terminated with semicolons. Several instructions (separated by semi-colons) can occupy the same line.

Status bits

There is a status bit for every node with a global input or specific input/output of Peer Cop data. If a defined group of data was successfully transferred within the set time out, the corresponding status bit is set to 1. Alternatively, this bit is set to 0 and all data belonging to this group (of 0) is deleted.

Step

SFC Language element: Situations, in which the Program behavior follows in relation to the inputs and outputs of the same operations, which are defined by the associated actions of the step.

Step name

The step name functions as the unique flag of a step in a Program organization unit. The step name is automatically generated, but can be edited. The step name must be unique throughout the whole program organization unit, otherwise an Error message appears.
The automatically generated step name always has the structure: S_n_m

S = Step
n = Section number (number running)
m = Number of steps in the section (number running)

Structured text (ST)

ST is a text language according to IEC 1131, in which operations, e.g. call up of Function blocks and Functions, conditional execution of instructions, repetition of instructions etc. are displayed through instructions.

Structured variables

Variables, one of which is assigned a Derived data type defined with STRUCT (structure).
A structure is a collection of data elements with generally differing data types (Elementary data types and/or derived data types).

SY/MAX

In Quantum control devices, Concept closes the mounting on the I/O population SY/MAX I/O modules for RIO control via the Quantum PLC with on. The SY/MAX remote subrack has a remote I/O adapter in slot 1, which communicates via a Modicon S908 R I/O system. The SY/MAX I/O modules are performed when highlighting and including in the I/O population of the Concept configuration.

Symbol (Icon)

Graphic display of various objects in Windows, e.g. drives, user programs and Document windows.

T

Template data file (Concept EFB)

The template data file is an ASCII data file with a layout information for the Concept FBD editor, and the parameters for code generation.

TIME

TIME stands for the data type "Time span". The input appears as Time span literal. The length of the data element is 32 bit. The value range for variables of this type stretches from 0 to 2exp(32)-1. The unit for the data type TIME is 1 ms.

Time span literals

Permitted units for time spans (TIME) are days (D), hours (H), minutes (M), seconds (S) and milliseconds (MS) or a combination thereof. The time span must be denoted by the prefix t#, T#, time# or TIME#. An "overrun" of the highest ranking unit is permitted, i.e. the input T#25H15M is permitted.

Example
t#14MS, T#14.7S, time#18M, TIME#19.9H, t#20.4D, T#25H15M, time#5D14H12M18S3.5MS

Token

The network "Token" controls the temporary property of the transfer rights via a single node. The token runs through the node in a circulating (rising) address sequence. All nodes track the Token run through and can contain all possible data sent with it.

Traffic Cop

The Traffic Cop is a component list, which is compiled from the user component list. The Traffic Cop is managed in the PLC and in addition contains the user component list e.g. Status information of the I/O stations and modules.

Transition

The condition with which the control of one or more Previous steps transfers to one or more ensuing steps along a directional Link.

U

UDEFB

User defined elementary functions/function blocks
Functions or Function blocks, which were created in the programming language C, and are available in Concept Libraries.

UDINT

UDINT stands for the data type "unsigned double integer". The input appears as Integer literal, Base 2 literal, Base 8 literal or Base 16 literal. The length of the data element is 32 bit. The value range for variables of this type stretches from 0 to 2exp(32)-1.

UINT

UINT stands for the data type "unsigned integer". The input appears as Integer literal, Base 2 literal, Base 8 literal or Base 16 literal. The length of the data element is 16 bit. The value range for variables of this type stretches from 0 to (2exp16)-1.

Unlocated variable

Unlocated variables are not assigned any state RAM addresses. They therefore do not occupy any state RAM addresses. The value of these variables is saved in the system and can be altered with the reference data editor. These variables are only addressed by symbolic names.

Signals requiring no peripheral access, e.g. intermediate results, system tags etc, should primarily be declared as unlocated variables.

V

Variables

Variables function as a data exchange within sections between several sections and between the Program and the PLC.
Variables consist of at least a variable name and a Data type.
Should a variable be assigned a direct Address (Reference), it is referred to as a Located variable. Should a variable not be assigned a direct address, it is referred to as an unlocated variable. If the variable is assigned a Derived data type, it is referred to as a Multi-element variable.
Otherwise there are Constants and Literals.

Vertical format Vertical format means that the page is higher than it is wide when looking at the printed text.

W

Warning When processing a FFB or a Step a critical status is detected (e.g. critical input value or a time out), a warning appears, which can be viewed with the menu command **Online** → **Event display...** . With FFBs the ENO output remains at "1".

WORD WORD stands for the data type "Bit sequence 16". The input appears as Base 2 literal, Base 8 literal or Base 1 16 literal. The length of the data element is 16 bit. A numerical range of values cannot be assigned to this data type.

Index

E

F

G

H

I

J

L

M

Q

R

S

T

U

V

W

X