Modicon
Micro Controllers
Ladder Logic
Manual

890 USE 146 00 Ver 1.0

September, 1997

Schneider Automation, Inc.
One High Street
North Andover, Massachusetts 01845

Preface

The data and illustrations in this book are not binding. We reserve the right to modify our products in line with our policy of continuous product improvement. Information in this document is subject to change without notice and should not be construed as a commitment by Schneider Automation, Inc. SA assumes no responsibility for any errors that may appear in this document.

Version 1.0 November, 1997

Contents

Chapter 1
Ladder Logic Operating System for the Micro PLCs

- The Modicon Micro Programmable Logic Controllers
- Memory Allocation
- Memory Backup
- Choosing a PLC Operating Mode
- The Ladder Logic Instruction Set

Modicon Micro Programmable Logic Controllers

A programmable logic controller (PLC) is a solid-state device with digital processing capabilities designed for real-time control of industrial and manufacturing applications. A PLC comprises input and output (I/O) units and a central processing unit (CPU).

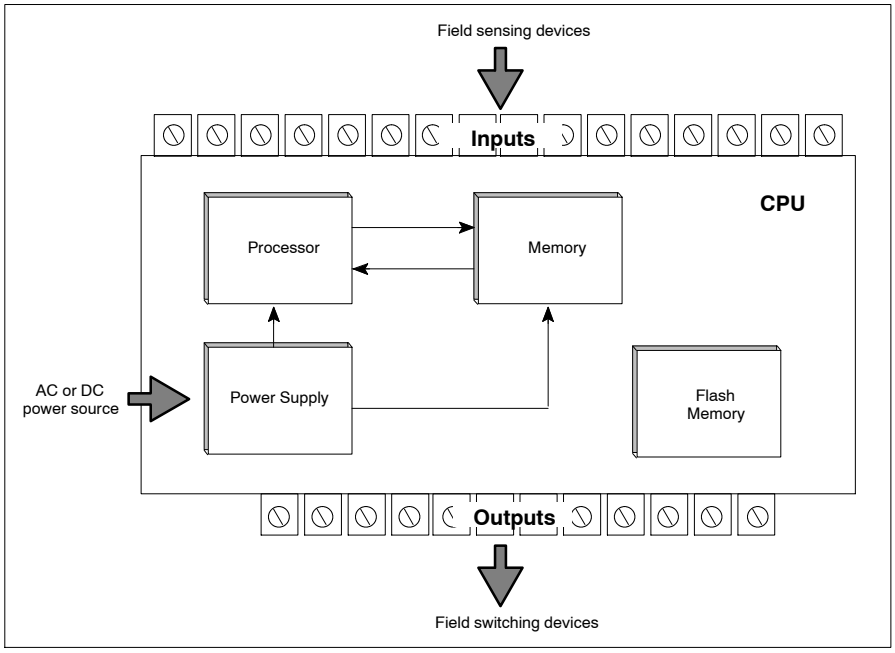
The Modicon Micro PLCs are fixed I/O devices. The input and output components are built into the same physical box with the CPU. The package provides a small, light-weight, low-cost, and self-contained solution for a wide range of control applications.

Theory of Operation

The block diagram below shows the major components of a Micro PLC. The PLC monitors the state of field devices by receiving signals from its inputs, solves a user logic program stored in its CPU, and then directs further field device activity by sending control signals to its outputs.

Inputs

The inputs are located in a terminal block across the top of the PLC. Inputs are field-wired to sensing devices in your application such as pushbuttons, selector switches, motor starter contacts, thumbwheels, or limit switches. If an input senses that a field sensor is closed, the input converts the field voltage to a logic-level signal understood by the CPU that describes the state of the sensor—a logic 1 indicates an ON or CLOSED state, and a logic 0 indicates an OFF or OPEN state.



CPU

Within the CPU are the digital processor, memory, and power supply. These components interact to solve application logic and pass control signals to the outputs. The CPU reads the converted input data, executes the user logic program stored in its memory, then writes the appropriate output signals to the field switching devices. The process of reading input signals, solving logic based on the states of the inputs, and then updating the output devices is called *scanning*.

Flash Memory

Also contained in the CPU is a Flash Memory component where the PLC's operating system resides. The contents of Flash are nonvolatile—they do not require battery backup.

The operating system residing in Flash is a collection of supervisory programs that give the PLC its identity by:

- Defining the language in which the application program is written—i.e., ladder logic
- Allocating the CPU's memory resources for specific purposes
- Determining the structure in which the PLC stores and handles data

The ladder logic operating system defines the functional capabilities of the Modicon Micro PLCs. Those capabilities are the primary focus of this book.

Outputs

The outputs are located in the terminal block across the bottom of the PLC. Outputs switch the supplied control voltage that energizes or de-energizes the field switching devices in your application. If an output is turned ON by the CPU, the control voltage is switched to activate the addressed device.

System Performance

Scan Time

The time it takes for the CPU to solve the ladder logic program and to update all the I/O under its control is called *scan time*. Scan time comprises logic solve time, I/O servicing time, and the time it takes to perform system overhead tasks.

The maximum amount of time allowed for the PLC to scan a user logic program one time is 250 ms. If the scan has not completed in that amount of time, a *watchdog timer* in the CPU stops the application and sends a timeout error message to the programming panel. This maximum scan time limit prevents the PLC from entering infinite loops in the logic program.

Logic Solve Time

The time it takes the CPU to solve the control logic in the program, independent of any service or administrative time, is called *logic solve time*.

- Logic solve time for the 110CPU311 and 110CPU411 Micro PLCs is 4.25 ms/K nodes of ladder logic
- Logic solve time for the 110CPU512 and 110CPU612 Micro PLCs is 2.5 ms/K nodes of ladder logic

Programming Note for 512XX and 612XX Controllers

In very small user logic test situations (e.g., using a contact to switch a coil as a fast oscillator), in Single or Child mode operation, the fast scantime [2.5 milliseconds per 1000 nodes programmed in a 512/612 Micro] may inhibit correct operation of the internal hardware output LED circuit and the internal output device circuit.

Both circuits react independently to user logic, so the LED may not reflect actual output operation.

The more logic that is programmed, the longer the scantime will be; and both LEDs and output circuits will then show the correct programmed response.

Consult the hardware manual provided with your unit to determine the response or switching time of the output device. [For example, the internal output relay has a maximum switching rate of 5 Hz.]

When the Micro is set up as a parent, this hardware restriction should not be seen, since each added Child Micro in the Parent configuration adds 3 milliseconds to the scantime.

Memory Allocation

The ladder logic operating system determines the way memory resources in a Modicon Micro PLC are allocated. It divides available system memory into three classes:

- *User data memory*—for variable data that changes during program execution
- *System configuration memory*—for storing system data tables such as the I/O map and PLC setup values
- *User program memory*—where the ladder logic program is created and edited

User Data Memory

The PLC relates each input and output signal in the control process to a reference number that is stored in a user data memory table and can be used in the ladder logic program. (The user data memory table is sometimes referred to as the *state RAM* table.)

- 110CPU311 and 110CPU411 PLCs have 512 words of user data memory
- 110CPU512 and 110CPU61200/03 PLCs have 2048 words of user data memory
- 110CPU61204 PLC has 8192 words of user data memory

☞ **Note** The execution buffer in the 61204 is large enough to load the XMIT and/or Gas loadable without reducing the 8K of available user logic.

Reference Numbering

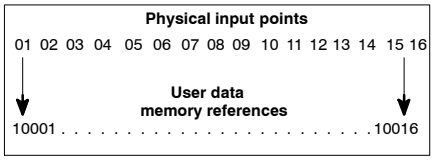
For ladder logic programming, the Modicon Micro PLCs use a reference numbering system to handle input/output information and internal logic. Each reference number has a leading digit that identifies the I/O data type; the leading digit is followed by a string of four digits that defines that I/O point's unique location in user data memory.

There are four reference types:

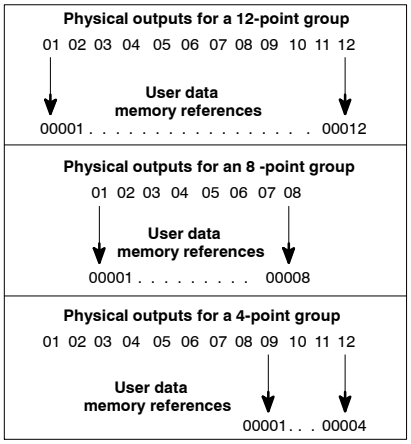
I/O Reference Numbering System	
Reference Number	**Description**
0xxxx	A discrete output (or coil). A 0*x* reference can be used to drive real output data through an output unit in the control system or it can be used to set one or more coils in state RAM. A specific 0*x* reference may be used only once as a coil in a logic program, but that coil status may be used multiple times to drive contacts in the program
1xxxx	A discrete input. The ON/OFF status of a 1*x* reference is controlled by field data sent to the CPU from an input unit. It can be used to drive contacts in a logic program
3xxxx	An input register. A 3*x* register holds information represented by A 16-bit number and received from an external source—e.g., a thumbwheel, an analog signal, data from a high speed counter. A 3*x* register can also hold 16 consecutive discrete input signals, which may be entered into the register in binary or binary coded decimal (BCD) format.
4xxxx	An output or holding register. A 4*x* register may be used to store numerical data (binary or decimal) in state RAM or to send the data from the CPU to an output unit in the control system.
Note: The *x* following the leading character in each reference type represents a four-digit address location in user data memory—e.g., the reference 40201 indicates that the reference is a 16-bit output or holding register located at address 201 in state RAM.	

Each word in user data memory is 16 bits long. The (ON/OFF) state of each discrete I/O point is represented by the 1 or 0 value assigned to an individual bit in a word (16 0*x* or 1*x* references per word).

For I/O mapping, physical input point #1 is mapped to the lowest numbered internal input in the first group of 16, physical input #2 to the next highest internal input, etc., as shown here:



Discrete outputs are mapped similarly according to their groupings:



In the case of analog I/O, each input channel and each output channel is mapped to a full word in user data memory (3*x* registers for inputs and 4*x* registers for outputs).

System Configuration Memory

The PLC configuration is a key piece of overhead contained in system memory. The information contained in the configuration determines such things as:

- The operating mode of the PLC—i.e., single, parent, or child
- The parameters under which the PLC's communication ports can operate
- The ranges of available 0*x*, 1*x*, 3*x*, and 4*x* references available for programming
- The number of I/O locations supported by the PLC

☞ **Note** The 61204 requires Modsoft, or Modsoft Lite version 2.5 or higher.

With your programming panel software, you can access the configuration and specify many of these parameters.

System configuration memory is pre-assigned to support the following default PLC configuration:

Default PLC Setup Values		
Parameter	**110CPU Model**	
	311 / 411	**512 / 612**
Number of 0*x* outputs	1024	1536
Number of 1*x* inputs	256	512
Number of 3*x* inputs	32	48
Number of 4*x* outputs	400	1872
Number of I/O locations	5	5
Number of segments of ladder logic	2 (one for standard ladder logic and one for interrupts and subroutines)	2 (one for standard ladder logic and one for interrupts and subroutines)

These default values make use of all the memory available for PLC setup. You may replicate pieces of system configuration memory to suit the I/O requirements in a specific application.

For example, if you are using a 110CPU31101 PLC and your application requires 35 register inputs instead of the default 32 registers, you could reassign the extra three words from elsewhere in the setup table. Say the application does not require all 1024 discrete outputs—you could specify 976 discrete outputs in the PLC setup table, then reallocate the extra 48 bits as the three additional (16-bit) input register words.

Note The total amount of memory configured for PLC setup cannot exceed the sum of the values shown in the table of default PLC setup values.

User Program Memory

Depending on the model PLC you are using, the amount of memory available for ladder logic programming is:

- 1024 words (for 110CPU311 and 110CPU411 PLCs)
- 2048 word (for 110CPU512 and 110CPU61200/03 PLCs)
- 8192 word (110CPU61204 PLC)

These are the total amounts of memory available for program logic. However, certain optional PLC functionality—e.g., additional loadable instructions—consume some of the memory set aside for user programming.

User program memory is divided into two *segments*. The first is where all ladder logic for standard application control resides. The second is reserved for subroutine logic, which can be called either by an instruction called **JSR** in ladder logic or by a high-speed interrupt input (available on 110CPU411, 110CPU512, and 110CPU612 PLCs.)

Note For more information about subroutines, see Chapter 10.

Here are the loadable instructions that can be used with the Modicon Micro PLCs and the amount of user program memory that each consumes:

Loadable Name	Size (Words)	Default Opcode	Function
EARS	760*	5F	For developing an early alarm reporting system
EUCA	160*	1F	An engineering unit conversion algorithm
FN*xx*	user-defined	5F	Lets you custom design your own DX loadable
XMIT	*	1E	Sends Modbus messages from master to multiple slaves or sends ASCII strings from Modbus port to ASCII printers
Gxxx	*	1F	Measures gas flow rate meeting AGA 3 and AGA 8 requirements

* These values will vary with the 61204.

For more information on these loadable instructions, refer to the following Modicon technical publications:

- ***Event Alarm Reporting System User Guide*** (GM-EARS-001)
- ***EUCA Loadable Function Block User Guide*** (GM-EUCA-001)
- ***Custom Loadable Support Software Programming Manual*** (GM-CLSS-001)
- ***XMIT Loadable Function Block User Guide*** (840 USE 113 00)
- ***Gas Loadable Function Block User Guide*** (890 USE 137 00)

Memory Backup

User data memory, user program memory, and system configuration memory can be backed up in any of three different ways:

- With an optional (110XCP98000) lithium battery
- With an optional (110XCP99000) battery capacitor
- In a reserved area in the PLC's Flash

Optional Backup Techniques

If a lithium battery assembly or battery capacitor assembly is used, it automatically backs up the current memory values in the event of a power loss. When power is restored, the PLC comes back up operational with the configuration and program values that were present at the time power was lost.

The lithium battery safely backs up memory data for one year. The battery capacitor can back up a typical user logic program for up to 72 hours (see the installation manual distributed with your PLC for more details).

Using Flash for Backup

A portion of the Flash memory in all Modicon Micro PLCs is reserved for storing the system configuration, user logic, and user data memory, except for the 61204. Because of limitations of Flash storage capabilities in the 61204, a battery is linked to the hardware. This feature allows you to back up your configuration and user logic even if you do not use a battery or battery capacitor.

To store the memory in Flash, you must issue a *save to Flash* command from your panel software. The values in memory at the time you issue the save are the only memory values stored in Flash. A *save to Flash* operation is allowed only in a PLC after it has been configured and while it is stopped—i.e., not scanning ladder logic.

If memory is restored to the PLC from Flash backup after a power loss, the values that were current at the time of your last save operation will be restored.

PLC Power-up Procedures

When the PLC receives power, it first checks system configuration memory to see if a valid configuration exists. If a valid configuration has been saved via the optional battery backup, these values will be present in user data memory. The PLC will configure itself with these values and be ready to operate.

If the PLC does not detect a valid configuration in user data memory, it will check the Flash backup. If a valid configuration has been saved in Flash, the PLC will configure itself with these values and be ready to operate.

If the PLC cannot find a valid configuration in memory or in Flash, it will power up in an *unconfigured* condition. You need to connect a programming panel to the PLC and configure it before it can be programmed or before it can solve logic.

Storing a PLC with User Logic Saved to Flash

☞ **Note** If you have saved a logic program to Flash in a PLC and are taking that PLC out of service, remember that all values stored in Flash are nonvolatile.

The PLC will immediately start using the stored program when it is powered up again sometime in the future. Potential problems could occur should the PLC be put in long-term storage, then installed in a new application.

If you are not sure how the PLC will be used in the future, you might want to clear logic from Flash before you take it out of service. To do this:

Step 1. Delete all the networks in the logic program.

Step 2. Set all the PLC's configuration parameters to their default values.

Step 3. Make sure the PLC is stopped.

Step 4. Then use your panel software to *save to Flash*.

PLC Operating Modes

A Modicon Micro PLC can be configured to operate in one of three modes:

- *Single* mode—operating as a standalone programmable control system, managing its own fixed I/O resources (and, in the case of the 110CPU512 and 110CPU612 PLCs, able to manage additional A120 I/O resources)
- *Parent* mode—operating as the one PLC on an I/O expansion link whose CPU can manage the fixed I/O resources of all the PLCs on that link
- *Child* mode—operating as a PLC on an I/O expansion link, allowing some or all of its fixed I/O resources to be accessed and managed by the parent PLC on the link

The I/O Expansion Link

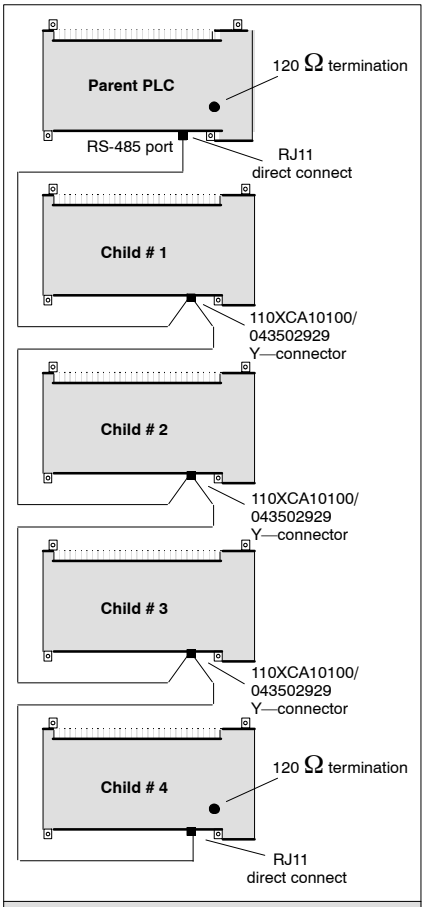
An *I/O expansion link* comprises a parent PLC and 1 ... 4 child PLCs connected via standard six-position telephone cables. Each cable has an RJ11 connector on both ends. PLC-to-PLC connections are made at the RS-485 (**exp link**) port on each unit.

Only one PLC on the link can be configured as the parent. All other PLCs on the link must be configured as child PLCs. A PLC in single operating mode cannot be used on an expansion link.

Each child PLC is uniquely addressed with a child ID # in the range #1 ... #4. The fixed I/O resources of the child PLCs can be accessed and controlled by logic running in the parent.

☞ **Note** It is your responsibility as a user to make sure that each child PLC is given a unique child ID number. The child ID assignment is made by connecting the programming panel to the child and entering the number as part of the child's configuration.

I/O expansion is accomplished via serial, point-to-point connections between the parent and child PLCs, as shown below.



An I/O Expansion Network

A120 I/O Expansion

110CPU512 and 110CPU612 PLCs are equipped with a 30-pin expansion port that allows the units to communicate with racks of A120 I/O. This port is dedicated to A120 I/O communications.

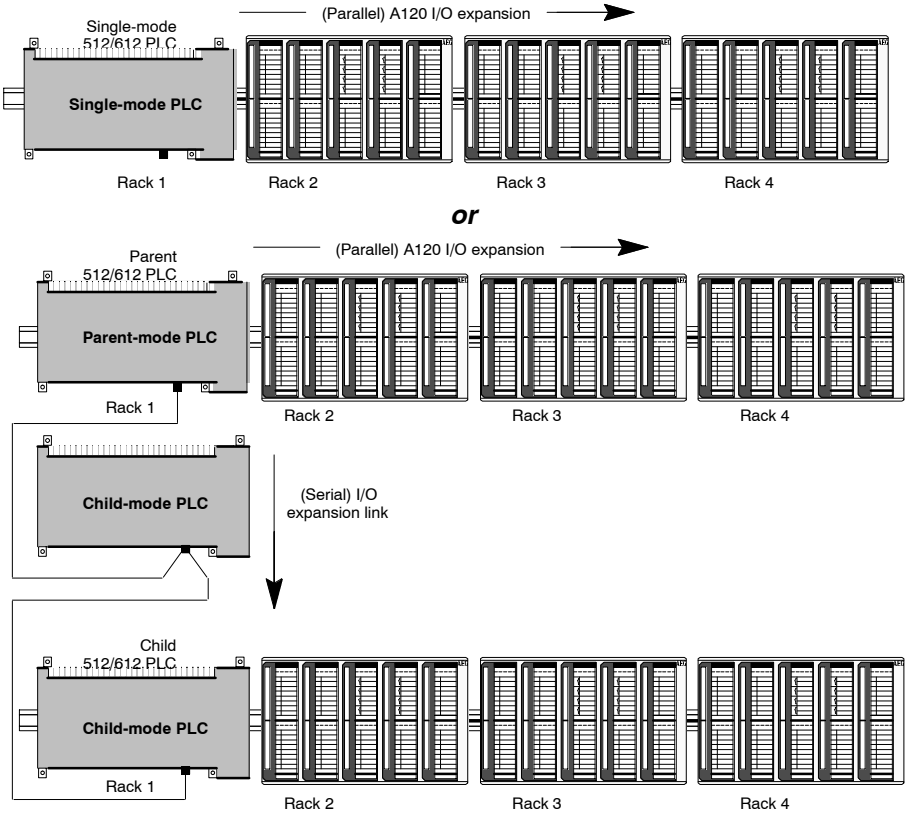
☞ **Note** 110CPU311 and 110CPU411 PLCs do not support A120 I/O expansion.

With A120 I/O expansion, 2 ... 4 racks are interconnected along a parallel bus physically mounted on DIN rail. The PLC itself is always configured as rack 1, and the A120 I/O housing are configured as racks 2 ... 4.

A120 I/O expansion can be employed by the PLC in any of its three operating modes.

A120 I/O can be accessed only by the PLC to which it is connected. This means that the ladder logic program driving the A120 I/O and all the associated A120 I/O mapping must be stored in the PLC to which the the A120 I/O is connected.

☞ **Note** If a child PLC on a serial I/O expansion link uses A120 I/O expansion, the A120 I/O associated with that child cannot be accessed by the parent on the link. The child must be independently programmed with its own ladder logic, PLC configuration, and I/O map to handle that A120 I/O.



The Ladder Logic Instruction Set

The ladder logic operating system, which resides in a Modicon Micro PLC's Flash RAM, contains the instruction set listed below. Note that some models of the Micro have an enhanced instruction set with functionality not available on the lower end models.

Standard Ladder Logic Instructions (available on all Micro PLCs)	
Instruction	**Description**
Relay Logic	
─┤ ├─	A normally open (N.O.) contact
─┤/├─	A normally closed (N.C.) contact
─┤↑├─	A positive transitional contact
─┤↓├─	A negative transitional contact
─( )─	A normal coil
─(M)─	A memory-retentive coil
Counters	
UCTR	An up counter from 0 to a specified preset
DCTR	A down counter to 0 from a specified preset
Timers	
T1.0	A timer that increments in seconds
T0.1	A timer that increments in tenths of a second
T.01	A timer that increments in hundredths of a second
T1MS	A timer that increments in ms
Integer Math	
ADD	Addition
SUB	Subtraction or greater than/less than operations
MUL	Multiplication
DIV	Division
Data Move	
R→T	A register-to-table move
T→R	A table-to-register move
T→T	A table-to-table move
BLKM	A block move
FIN	A first-in operation to a queue
FOUT	A first-out operation from a queue
SRCH	A table search for a bit pattern in one of the registers

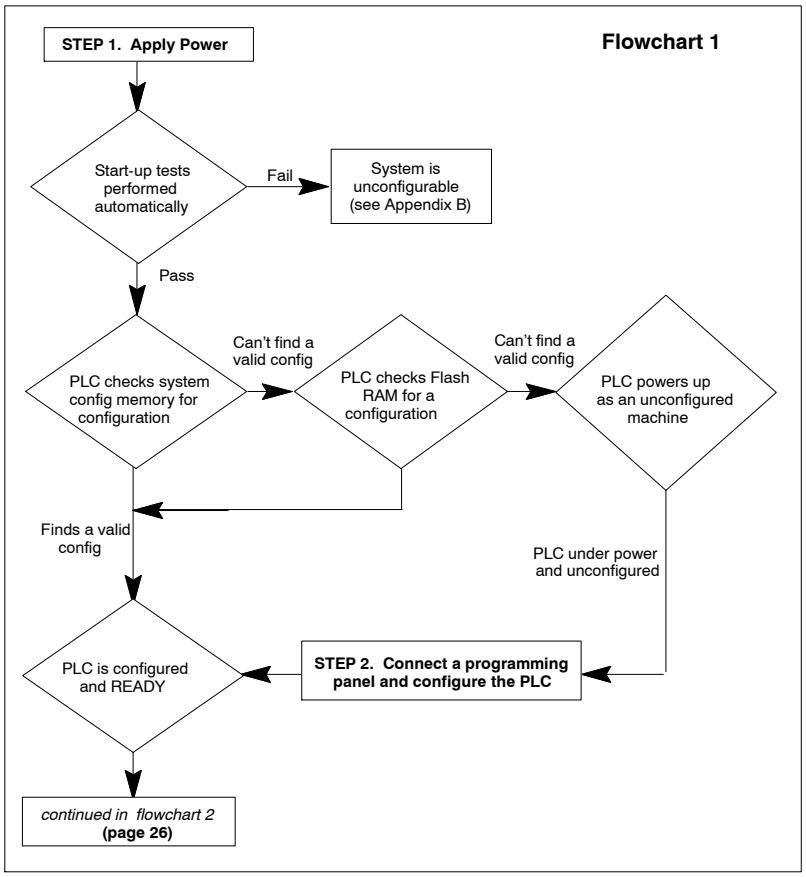
Standard Ladder Logic Instructions (continued)	
Instruction	**Description**
Data Matrix	
AND	A logical AND of two matrices
OR	A logical OR of two matrices
XOR	A logical exclusive OR of two matrices
COMP	A logical complement of the bit pattern in a matrix
CMPR	A logical compare of the bit patterns in two matrices
MBIT	A bit modify—i.e., changing the current (1, 0) value of the bit
SENS	A bit sense—i.e., reporting the current (1, 0) value of the bit
BROT	A bit rotation—i.e., shifting the bit positions left or right in a matrix
ASCII	
COMM	An ASCII read or write communication operation
Sequencing	
SCIF	Drum sequencing and input comparison operations
Subroutines	
JSR	Jumps the logic scan from control logic to a ladder logic subroutine programmed in the last segment
LAB	Labels the entry location for the called subroutine in the last segment
RET	Returns the logic scan to its previous place in logic prior to the JSR
CTIF	Sets up the high-speed inputs for interrupt and counter/timer operations
Other	
STAT	Checks and reports the health of the PLC and its I/O
SKP	Causes the logic scan to skip specified networks in the program

Enhanced Ladder Logic Instructions (available in specified 110CPU512 and 110CPU612 Models only)	
Instruction	**Description**
BLKT	A block-to-table move
TBLK	A table-to-block move
CKSM	Performs CRC-16, LRC, straight, or binary checksum operations
PID2	Performs proportional-integral-derivative control functions
EMTH	Performs extended math functions such as square root, process square root, log, antilog, and floating point operations

Chapter 2
Start-up Procedures

- ❒ Getting Started
- ❒ Autoconfiguration Parameters
- ❒ Autoconfigured Communication Ports
- ❒ Modifying the Configuration Parameters
- ❒ Addressing I/O Locations
- ❒ Addressing A120 I/O
- ❒ Addressing I/O on an Expansion Link
- ❒ Splitting I/O between Parent and Child PLCs
- ❒ Generalized Data Transfer
- ❒ PLC Operations

Getting Started



Applying Power

As soon as you apply power to a Modicon Micro PLC, it will attempt to start operating. The operating system tries to retrieve any previously stored configuration data from memory backup.

Starting a Previously Configured PLC

If the PLC has been started before and has had a configuration (and possibly a logic program) saved in its memory, it will immediately start operating using the stored values.

If the PLC has an optional battery backup, it will find the previous configuration parameters in its system configuration memory and the previous user logic values in its user program memory. Configuration and user logic may alternatively be saved to the PLC's Flash RAM if you are not using a battery backup.

As the flowchart above shows, the operating system checks the PLC's system configuration memory first. If it finds a

valid configuration stored there, it uses those values to operate. If it does not find a valid configuration in system configuration memory, it checks the PLC's Flash RAM for a valid configuration. If it finds a valid configuration stored there, it uses those values to operate.

If the previous condition of the PLC was in RUN mode, the PLC will begin scanning its logic immediately. You do not need to connect a programming panel to it.

If the previous condition of the PLC was in STOPPED mode, you will need to connect a programming panel to one of the **Comm** ports on the PLC in order to start it.

Starting an Unconfigured PLC

If the operating system cannot find a valid configuration in the PLC's Flash or in its system configuration memory, it will power up as an *unconfigured* machine. A PLC will power up unconfigured the first time it is ever been started or when its configuration values have been cleared or corrupted.

You need to configure the PLC before you can write a logic program or service the I/O.

Configuring a Modicon Micro PLC

Step 1. Connect a programming panel, such as MODSOFT Lite or the HHP*, to an RS-232 comm port on the PLC.

Step 2. Using the panel's menuing system, go to the configuration editor. (The path to the configuration editor will vary depending on the panel you are using, but it is a high-level screen that can be reached with minimal keystrokes.)

Step 3. Make sure that the panel knows which PLC model type (e.g., a 110CPU31101, a 110CPU51200) it is about to configure. The HHP* dis plays this information auto matically at startup; MOD SOFT Lite prompts you to select the model type from a list.

Step 4. Select the desired operating mode for the PLC you want to configure. The operating mode can be either single, parent, or child.

Step 5. Transfer the configuration parameters from the panel to the PLC.

Result. The panel automatically configures the PLC with a full set of valid parameters based on the model and operating mode you specify. At this point, the PLC is configured.

* The 520VPU19200 HHP does not support the 61204.

Autoconfiguration Parameters

Based on the PLC model type and PLC operating mode that you specify, the panel automatically configures the PLC with a full set of valid parameters. These *autoconfiguration parameters* are shown in the following three tables.

Autoconfiguring a PLC in Single Operating Mode

If you configure a PLC in single operating mode, the autoconfigured parameters shown below are all you need to begin your ladder logic programming.

Autoconfiguration Parameters for a Single Mode Micro PLC

Parameter	110CPU Models	
	311 / 411	**512 / 612**
Number of 0*x* references	1024	1536
Number of 1*x* references	256	512
Number of 3*x* references	32	48
Number of 4*x* references	400	1872
Number of ladder logic segments	2 (the first for control logic and the second for subroutines)	2 (the first for control logic and the second for subroutines)
RS-232 port (**comm 1**)	Dedicated Modbus mode: 8-bit RTU communications, 9600 baud, even parity, 1 STOP bit, Modbus address #1	Dedicated Modbus mode: 8-bit RTU communications, 9600 baud, even parity, 1 STOP bit, Modbus address #1
RS-232 port (**comm 2**)	N/A	Dedicated Modbus mode: 8-bit RTU communications, 9600 baud, even parity, 1 STOP bit, Modbus address #1
RS-485 port (**exp. link**)	Dedicated ASCII 8-bit ASCII communications, 9600 baud, even parity, 1 STOP bit	Dedicated ASCII 8-bit ASCII communications, 9600 baud, even parity, 1 STOP bit

Autoconfiguring a PLC in Parent Operating Mode

If you specify parent operating mode, you must specify the number of child PLCs that will be allowed on the I/O expansion link. The number must be in the range 1 ... 4.

Once you have specified this number, the PLC is ready to be programmed.

Autoconfiguration Parameters for a Parent Mode PLC

Parameter	110CPU Models	
	311 / 411	**512 / 612**
Number of 0*x* references	1024	1536
Number of 1*x* references	256	512
Number of 3*x* references	32	48
Number of 4*x* references	400	1872
Number of child PLCs on the I/O expansion link	*must be user-specified*	*must be user-specified*
Number of ladder logic segments	2 (the first for control logic and the second for subroutines)	2 (the first for control logic and the second for subroutines)
RS-232 port (**comm 1**)	Modbus/ASCII toggling mode: 8-bit RTU/8-bit ASCII communications, 9600 baud, even parity, 1 STOP bit, Modbus address #1	Dedicated Modbus mode: 8-bit RTU communications, 9600 baud, even parity, 1 STOP bit, Modbus address #1
RS-232 port (**comm 2**)	N/A	Modbus/ASCII toggling mode: 8-bit RTU/8-bit ASCII communications, 9600 baud, even parity, 1 STOP bit, Modbus address #1
RS-485 port (**exp. net**)	I/O expansion network: 9-bit data communications, 125 ,000 baud, 1 STOP bit	I/O expansion network: 9-bit data communications, 125 ,000 baud, 1 STOP bit

Autoconfiguring a PLC in Child Operating Mode

If you specify child operating mode, you must assign a child ID number to the PLC. The number must be in the range 1 ... 4, and it must be unique to the particular child you are configuring with respect to all other child PLCs to be placed on the I/O expansion link.

Once you have specified the child ID #, the PLC is ready to be programmed.

Autoconfiguration Parameters for a Child Mode PLC

Parameter	110CPU Models	
	311 / 411	**512 / 612**
Number of 0*x* references	1024	1536
Number of 1*x* references	256	512
Number of 3*x* references	32	48
Number of 4*x* references	400	1872
Child ID #	*must be user-specified*	*must be user-specified*
Number of ladder logic segments	2 (the first for control logic and the second for subroutines)	2 (the first for control logic and the second for subroutines)
RS-232 port (**comm 1**)	Modbus/ASCII toggling mode: 8-bit RTU/8-bit ASCII communications, 9600 baud, even parity, 1 STOP bit, Modbus address #1	Dedicated Modbus mode: 8-bit RTU communications, 9600 baud, even parity, 1 STOP bit, Modbus address #1
RS-232 port (**comm 2**)	N/A	Modbus/ASCII toggling mode: 8-bit RTU/8-bit ASCII communications, 9600 baud, even parity, 1 STOP bit, Modbus address #1
RS-485 port (**exp. net**)	I/O expansion network: 9-bit data communications, 125 ,000 baud, 1 STOP bit	I/O expansion network: 9-bit data communications, 125 ,000 baud, 1 STOP bit

Some Autoconfiguration Examples

Let's look at some MODSOFT Lite configuration overview screens and talk about the meaning of the displayed parameters. Below we show three screens for a 110CPU51200 PLC, configured in each of its three operating modes. MODSOFT Lite examples are used here to illustrate conceptual issues related to PLC configuring. MODSOFT Lite is not the only programming software available for configuring a Micro; these examples are used because the individual screens contain more values than those in the HHP*. For a thorough description of MODSOFT Lite or HHP* editing procedures, refer to the programming manual provided with your software package.

```
Utility          OverView I/OMap   Ports            Loadable          Quit
F1———————F2———————F3———————F4———————F5———————F6———————F7—Lev 8—F8—OFF——F9———
                        CONFIGURATION OVERVIEW
                                    Size of Full Logic Area  02122
     PLC  :                         No. of I/O Map Words     00153
     PLC Type       984 -  MICRO-S
     Model                512/00    I/O  :
     Memory                 3.1K    Number of Segments       2
     Micro Child ID         None    Number of Children       0
                                    I/O Locations            18

                                    Specials  :
      Ranges   :                    Battery Coil             00081
      0xxxx    00001  - 01536       Timer Register           40011
      1xxxx    10001  - 10512       Time of Day Clock        40012  - 40019
      3xxxx    30001  - 30048
      4xxxx    40001  - 41872
```

Screen 1. 110CPU51200 PLC with Autoconfigured Single-mode Parameters

```
Utility          OverView I/OMap   Ports            Loadable          Quit
F1———————F2———————F3———————F4———————F5———————F6———————F7—Lev 8—F8—OFF——F9———
                        CONFIGURATION OVERVIEW
                                    Size of Full Logic Area  02117
     PLC  :                         No. of I/O Map Words     00158
     PLC Type       984 -  MICRO-P
     Model                512/00    I/O  :
     Memory                 3.1K    Number of Segments       2
     Micro Child ID         None    Number of Children       1
                                    I/O Locations            18

                                    Specials  :
      Ranges   :                    Battery Coil             00081
      0xxxx    00001  - 01536       Timer Register           40011
      1xxxx    10001  - 10512       Time of Day Clock        40012  - 40019
      3xxxx    30001  - 30048
      4xxxx    40001  - 41872
```

Screen 2. 110CPU51200 PLC with Autoconfigured Parent-mode Parameters

```
Utility          OverView I/OMap   Ports            Loadable          Quit
F1———————F2———————F3———————F4———————F5———————F6———————F7-Lev 8-F8-OFF——F9———
                        CONFIGURATION OVERVIEW
                                   Size of Full Logic Area  02122
     PLC  :                        No. of I/O Map Words     00153
     PLC Type       984 -  MICRO-C
     Model                 512/00  I/O  :
     Memory                  3.1K  Number of Segments       2
     Micro Child ID           01   Number of Children       0
                                   I/O Locations            18

                                   Specials  :
                                   Battery Coil             00081
      Ranges   :                   Timer Register           40011
      0xxxx    00001  - 01536      Time of Day Clock        40012  - 40019
      1xxxx    10001  - 10512
      3xxxx    30001  - 30048
      4xxxx    40001  - 41872
```

Screen 3. 110CPU51200 PLC with Autoconfigured Child-mode Parameters

PLC Operating Mode

The operating mode is described in the `PLC Type` entry in the top left data field of the screens. **`MICRO-S`** indicates single mode; **`MICRO-P`** indicates parent mode; and **`MICRO-C`** indicates child mode.

Child ID

The child ID # must be specified for a PLC that is configured in child operating mode. The MODSOFT Lite configuration defaults to an ID of 1. When you are configuring more than one child on an I/O expansion link, you need to make sure that each has a unique ID# in the range 1 ... 4.

This parameter does not apply to parent and single PLCs. For PLCs in either of these modes, the `Micro Child ID` is specified as `NONE`.

0*x*, 1*x*, 3*x*, and 4*x* Reference Ranges

The range of internal memory references is the same in all modes. The autoconfigured range assignments are the maximum number of references available for 110CPU51200 model.

☞ **Note** The range of references is smaller for 110CPU311 and 110CPU411 models.

Number of Ladder Logic Segments

The autoconfigured number of ladder logic segments is 2. The first segment is available for normal control logic, and the second segment is available for subroutine logic.

Number of Child PLCs

If the PLC is configured in parent operating mode, you must specify the number of child PLCs that it can access on the I/O expansion link. The MODSOFT Lite configuration defaults to 1. If you want the ability to put more than one child on the link, change this parameter.

This parameter does not apply to single and child PLCs. For PLCs in either of these modes, the `Number of Children` is specified as `0`.

I/O Locations

An I/O location is a unit of I/O associated with a particular type of Micro PLC. These I/O locations, which are described in more detail later in this

chapter, include the fixed I/O built into the PLC and any A120 I/O modules connected to the PLC over the parallel expansion port.

☞ **Note** Only 110CPU512 and 110CPU612 models support A120 I/O; 110CPU311 and 110CPU411 models do not.

The 110CPU512 models all default to 18 I/O locations. This number allows you to support three or four fixed I/O locations—the discrete I/O, the high-speed inputs, and the generalized data transfer capability (more on these later)—as well as up to 15 slots of A120 I/O.

☞ **Note** 110CPU311 and 110CPU411 models will default to a much smaller number of I/O locations because these units do not support A120 I/O.

The Battery Coil

The operating system automatically sets aside reference 00081 as the battery coil. This coil operates much like the **batt low** LED on the PLC in that it turns ON when the optional battery needs to be replaced. You can tie this coil to an external alarm or display that warns you of the need for battery replacement.

When the battery coil goes ON, the battery should be replaced within 14 days.

The Timer Register

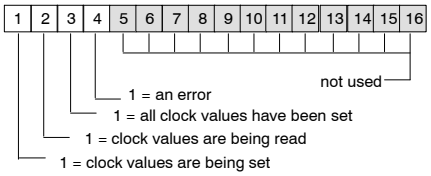
The operating system automatically sets aside output register 40011 as a free-running timer. This register is available to you for 10 millisecond applications in a ladder logic.

The Time of Day Clock

The operating system automatically reserves a block of eight contiguous output registers (40012 ... 40019) to store data from the PLC's time of day clock (in the 110CPU411, 110CPU512, and 110CPU612 models). You need to initialize the clock in order to use it.

The 16 bits in each register are used to store the following information:

- ❒ Register 40012—the control register:



- ❒ Register 40013—the day of the week (Sunday = 1, Monday = 2, etc.)
- ❒ Register 40014—the month of the year (Jan. = 1, Dec. = 12)
- ❒ Register 40015—day of the month (1 ... 31)
- ❒ Register 40016—year (0 ... 99)
- ❒ Register 40017—hour in military time (0 ... 23)
- ❒ Register 40018—minute (0 ... 59)
- ❒ Register 40019—second ((0 ... 59)

For example, if you read the TOD clock at 9:25:30 on Thursday, March 18, 1993, the block of register will display the following information:

40012 = 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
40013 = 5 (decimal format)
40014 = 3 (decimal format)
40015 = 18 (decimal format)
40016 = 93 (decimal format)
40017 = 9 (decimal format)
40018 = 25 (decimal format)
40019 = 30 (decimal format)

Autoconfigured Communication Ports

The RS-485 Port

For a parent or child PLC, the RS-485 (**exp link**) port must be used for interconnecting units on the I/O expansion link. In both these operating modes, the autoconfigured port parameters are set and cannot be changed.

For a single PLC, the **exp link** port cannot be set for I/O expansion; it must be either used for ASCII communications or disabled. The autoconfigured parameters for this port in single mode are for ASCII communications.

☞ **Note** Only one communication port can be set to perform ASCII communication functions.

The RS-232 Port(s)

The 110CPU512 and 110CPU612 Micro PLCs have two RS-232 communication ports, while the 110CPU311 and 110CPU411 models have only one. The autoconfigured parameters assigned to these ports depend on both the model and the operating mode of the PLC.

If a PLC is in parent or child operating mode, the panel software autoconfigures one of the RS-232 ports to a mode that supports communications between the PLC and either an ASCII input/output device or a *Modbus master* device.

☞ **Note** Modbus is the protocol that handles ladder logic programming communications between a programming panel and the PLC. The programming panel is considered the Modbus master device, and the PLC is considered the Modbus slave device.

In Modbus/ASCII toggling mode, the RS-232 port uses its DSR line to inform the PLC whether an ASCII device or a Modbus master device is connected.

- ❒ When an ASCII device—e.g., a printer or a character display monitor—is attached to **comm 1**, the DSR line becomes INACTIVE and the port communicates 8-bit ASCII data
- ❒ When a Modbus master device—e.g., the HHP* or a computer running MODSOFT Lite—is connected to **comm 1**, the DSR line becomes ACTIVE, and the port communicates 8-bit RTU data

If the PLC is a 110CPU311 or 110CPU411 model in parent or child mode, Modbus/ASCII toggle mode is autoconfigured on the **comm 1** port. If the PLC is a 110CPU512 or 110CPU612 model in parent or child mode, Modbus/ASCII toggle mode is autoconfigured on the **comm 2** port, and the **comm 1** port is autoconfigured for dedicated Modbus communications.

* The 520VPU19200 HHP does not support the 61204.

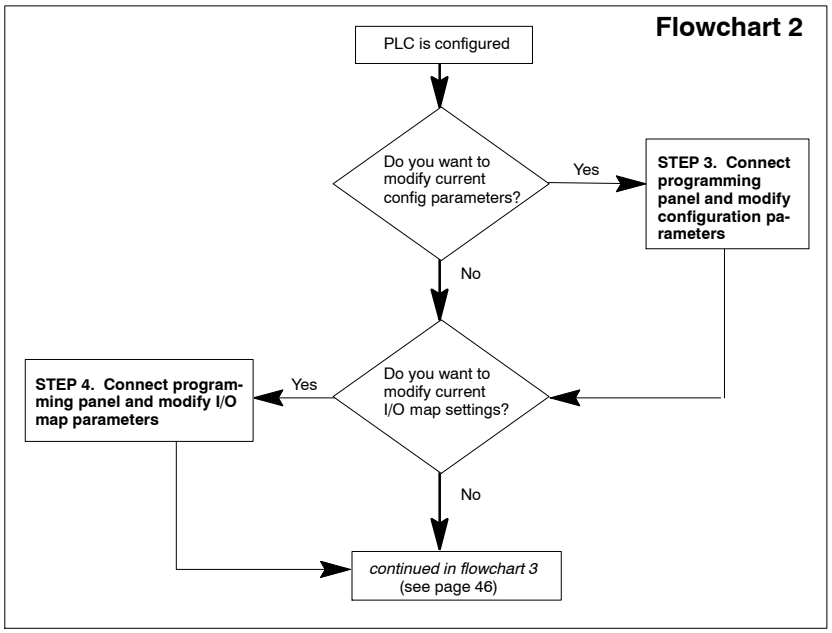
Comm 2 of the 61204 is a Modbus slave port in the default condition, the same as in other versions of the 612 and 512. But this port is also capable of control by the XMIT block which, when enabled, permits the port to be a temporary master in either ASCII or RTU mode. Refer to the **XMIT Loadable Function Block User Guide** (840 USE 113 00). Comm2 can only support one communication function block at a time. For example, either the Comm or XMIT block. If both the Comm and XMIT blocks are active, the XMIT block will operate properly while the Comm block will not.

In single-mode PLCs, the RS-232 ports are always autoconfigured for dedicated *Modbus* communications. This is because the RS-485 port is autoconfigured for ASCII, and only one port on the PLC can be support ASCII communications.

All RS-232 ports are autoconfigured for 9600 baud communications, which enables you to attach a programming panel to the PLC at any RS-232 port. Devices that do not communicate via the Modbus protocol cannot be used at a dedicated Modbus port.

☞ **Note** If you are using 9600 baud on one RS-232 port, you should not exceed 2400 baud on the other RS-232 port.

Modifying the Configuration Parameters



Depending on the programming panel you are using, you may be able to change many of the autoconfiguration settings for a PLC. The HHP* allows you to change only a few auto-configured parameters, whereas MODSOFT Lite gives you a great deal of flexibility in setting up the configuration.

Caution If you are using an HHP* to make changes to an existing PLC configuration, you will erase all ladder logic, I/O map, and ASCII message data currently stored in PLC memory.

The Number of References

With MODSOFT Lite you can change the mix of references in your configuration. However, you cannot increase the total register count.

For example, if your application requires 32 more 0*x* references, you can add 32 to the available total if you decrease the number of 1*x* references by 32 or if you decrease the the number of 3*x* or 4*x* references by 2 (3*x* and 4*x* registers contain 16 bits; 0*x* and 1*x* references are single bits).

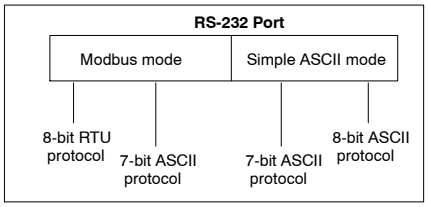
* The 520VPU19200 HHP does not support the 61204.

The Number of Logic Segments

The autoconfigured value of 2 should not be changed even if you do not plan to use subroutines. The second segment will never be scanned unless it is called by a JSR instruction or by a hardwired interrupt.

RS-232 Port Communication Parameters

The RS-232 ports can be set to operate in either of two modes—Modbus or simple ASCII. In Modbus mode, the port is a slave to the Modbus master device that is connected to it; this device is generally a programming panel. In simple ASCII mode, the port is read or written using ladder logic (see Chapter 7 for a description of the ASCII **COMM** ladder logic instruction).



Modbus Mode

In Modbus mode, the port can communicate using either an 8-bit remote terminal unit (RTU) protocol or a 7-bit ASCII protocol. On the **comm 1** RS-232 port, RTU can be supported only at 9600 baud, and ASCII can be supported only at 2400 baud. The **comm 1** port is also restricted to Even parity and 1 STOP bit for RTU and ASCII. The Modbus address of the port can be set in the range 1 ... 247.

☞ **Note** A Micro PLC can be a node on a Modbus network by assigning it a unique Modbus network address. If the PLC is not on a Modbus network, the default address of 1 should be kept. If the PLC is on a Modbus network, its address must be unique with respect to all other nodes on the network, in the range 1 ... 247. (See the ***Modicon Modbus Protocol Reference Guide***, PI-MBUS-300, for details.)

If your PLC has a **comm 2** RS-232 port, there are more optional port parameters available to you in Modbus mode:

Optional Comm Parameters for the comm 2 Port	
Baud	50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, 19200
Comm mode	7-bit ASCII, 8-bit RTU
Parity check	Odd, Even, None
STOP bits	1, 2
Modbus address	1 ... 247

☞ **Note** Comm1 and Comm2 (RJ45) RS–232 MUST not exceed12,000 combined baud rate, while using the (RJ11) RS–485 parent/child communication port. When you violate this rule the parent/child communications are disrupted. We support both Comm1 and Comm 2 at 9600 baud rate without any concerns when the PLC is running in *single* mode.

The following two combinations of RS-232 port parameters are *not* supported on the **comm1** or **exp link** ports for simple ASCII communications:

- ❒ 7-bit ASCII with 1 STOP bit and no parity
- ❒ 8-bit ASCII with 2 STOP bits and even or odd parity

Simple ASCII Mode

In simple ASCII mode, an RS-232 port can communicate only with an ASCII protocol, utilizing either 7-bit or 8-bit resolution. RTU communications are not permitted in Simple ASCII mode.

An RS-232 port in simple ASCII mode can be given any of the following port parameters:

Optional Comm Parameters for Simple ASCII	
Baud	1200, 2400, 4800, 9600
Comm mode	7-bit ASCII, 8-bit ASCII
Parity check	Odd, Even, None
STOP bits	1, 2

Modem Communication Capabilities

The **comm 1** port and **comm2** port (where available) on your Modicon Micro PLC are equipped with circuitry that supports modem hand-shaking signals. In order support modem communications, the port on the PLC must be in dedicated Modbus mode and a special adapter must be used on the modem end of the cable connection.

Caution Because of the special way the DSR line functions when the port is in the Modbus/ASCII toggling mode, the comm 1 port cannot communicate over a modem when it is set in this mode.

Four adapter kits are available from Modicon with the parts you will need to customize an adapter for your modem:

RJ45-to-D-shell Adapter Kits	
Adapter Description	**Part Number**
RJ45-to-9-pin D-shell, male	110XCA20301
RJ45-to-9-pin D-shell, female	110XCA20302
RJ45-to-25-pin D-shell, male	110XCA20401
RJ45-to-25-pin D-shell, female	110XCA20402

If you want to set up the unit for modem communications, place the comm port in dedicated Modbus mode, and set its port parameters to accommodate the modem—e.g., 2400 baud, ASCII mode.

RS-485 Port Communication Parameters

If the **exp net** RS-485 port is used as the dedicated ASCII port (the case only if the PLC is in single operating mode), the following communication parameters are available:

Optional Comm Parameters for the exp net Port	
Baud	1200, 2400, 4800, 9600
Comm mode	7-bit ASCII, 8-bit ASCII
Parity check	Odd, Even, None
STOP bits	1, 2

If the RS-485 port is used for I/O expansion—i.e., if the PLC is in parent or child operating mode—then the auto-configured port parameters are fixed and cannot be modified.

Addressing I/O Locations

The I/O map is a table in the PLC's system configuration memory that links reference numbers in the PLC's user data memory (0*x*, 1*x*, 3*x*, and 4*x*) to actual field inputs and outputs.

Fixed I/O Locations

A Modicon Micro PLC has five fixed I/O locations reserved for it in the I/O map editor.

- ❒ Location 1 for addressing fixed discrete input and output resources
- ❒ Location 2 for addressing counter/interrupt inputs
- ❒ Location 3 for addressing timer/counter inputs
- ❒ Location 4 for addressing fixed analog inputs and outputs
- ❒ Location 5 for addressing the transfer registers for a *generalized data transfer* operation between a parent and child PLC

Some of these locations may not be used for all PLC models—e.g., location 4 is reserved for fixed analog I/O which is available only in the 110CPU612s. When not used, a reserved fixed I/O location in the I/O map must be left empty—it cannot be used to address another type of I/O.

When you look at the I/O map in your panel software, the types of I/O points in each fixed I/O location are specified by an alphanumeric *location type*. The table below shows the standard location types for the fixed resources on all models of Micro PLCs.

I/O Map Location Types for Fixed I/O

I/O Location	110CPU Model	Fixed Resources	Location Type
Discrete (1)	31100, 41100 51200, 61200, 61204	16 ... 24 VDC in / 12 relay out	MIC128
	31101, 41101, 51201	16 ... 115 VAC in / 8 triac out 4 relay out	MIC131
	31102, 41102, 51202	16 ... 230 VAC in / 8 triac out 4 relay out	MIC134
	31103, 41103, 51203, 61203	16 ... 24 VDC in / 12 FET out	MIC137
Counter / Interrupt (2)	All 411, 512, and 612 Models	8-bit counter/interrupt in	MIC140
Timer / Counter (3)	N/A in 311 Models	16-bit timer/Current count value	MIC147
Analog (4) All output channels have 12-bit resolution	612 Models only	4 in (0 ... 10, 12-bit), 2 out	MIC141
		4 in (1 ... 5, 12-bit), 2 out	MIC142
		4 in (± 10, 12-bit), 2 out	MIC143
		4 in (0 ... 10, 15-bit), 2 out	MIC144
		4 in (1 ... 5, 14-bit), 2 out	MIC145
		4 in (± 10, 16-bit), 2 out	MIC146
Generalized Data Transfer (5)	All Models, set to Parent mode	1 word in, 1 word out	MIC148
		2 words in, 2 words out	MIC149
		4 words in, 4 words out	MIC150
		8 words in, 8 words out	MIC151

The operating system reserves the first twelve 0*x* references and the first sixteen 1*x* references (00001 ... 00016 and 10001 ... 10016) for the fixed discrete I/O resident on the unit.

For example, the fixed resources of a 110CPU51201 PLC in single operating mode would be addressed as follows:

- Location type **MIC131** in the first location to specify the discrete I/O points; the 115 VAC inputs are addressed to references 10001 ... 10016, the triac outputs are addressed to references 00001 ... 00008, the relay outputs are addressed to references 00009 ... 00012
- Location type **MIC140** in the second location to specify the high-speed interrupt/counter inputs, which are addressed to references 10081 ... 10088
- Location type **MIC147** in the third location to specify the high-speed timer/counter input, which is addressed to register 30001

The last two locations, for analog I/O and generalized data transfer, are not available in this I/O map. Only 110CPU612 PLCs support analog I/O, and only parent and child PLCs support generalized data transfer.

Below is a sample I/O map screen from MODSOFT Lite illustrating the way the discrete addressing is displayed:

```
Utility          Del I/O  HoldTme  Get I/O                                Quit
F1———F2———F3———F4———F5———F6———F7—Lev 8—F8—OFF——F9——
                                MICRO I/O MAP

                                Single I/O

 PLC         :  MICRO 512/01            Holdup Time  :  N/A
 Used Inputs :  040 of 512 Points       Used Outputs :  016 of 512 Points
 Next Input  :  10017                   Next Output  :  00017_
 ═══════════════════════════════════════════════════════════════════════
 Location      Type        Reference Numbers       Data     Description
                         Inputs        Outputs     Type

 DISC   I/O:  MIC131   10001-10016   00001-00016   BIN   16@115V I/O 8TR/4RY
 INT/CTR IN:  MIC140   10081-10088        -        BIN   8 INTPT/CNTR INP
 TMR/CTR IN:  MIC147   30001-30001        -        BIN   16BIT TMR/CNTR VAL
 ANALOG I/O:  NotAvl        -             -
 DATA TRANS:  NotAvl        -             -
```

Representing Fixed Analog Data

The 612 model Micro PLCs support four channels of fixed analog inputs and two channels of fixed analog outputs.

The Analog Input Channels

Each of the four input channels is addressed to a word in user data memory, followed by a fifth status word. Bits in the status word signal warning and out-of--range conditions in the input channels. The format of the status word is shown on page 32.

Word 5 Fixed Analog Input Status

If the bit is set to 1, the condition is TRUE

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

Channel 4 out of range
Channel 3 out of range
Channel 2 out of range
Channel 1 out of range
Channel 4 warning
Channel 3 warning
Channel 2 warning
Channel 1 warning

The analog input device in the PLC provides 16 bits of resolution. You may specify how this 16 bits is to be interpreted based on the I/O location identifier—i.e., the MIC number—you select in the I/O map screen.

The six tables that follow show the range of fixed analog input representations available to you:

MIC141 Analog Input Signals (with 12-bit resolution)

Voltage	Data Count (Decimal)	Operating Results
Less than 0	0 with channel out-of-range and warning bits set in status word	Under range
0 . . . 5 . . . 9.9976	0 . . . 2048 . . . 4095	Recommended operating range
Greater than 9.9976 and less than 10.24	4095 with channel warning bit set in he status word	High warning range
Greater than or equal to 10.24	4095 with channel out-of-range bit set in he status word	Over range

MIC142 Analog Input Signals (with 12-bit resolution)			
Voltage	**Current (mA)**	**Data Count (Decimal)**	**Operating Results**
Less than .52	Less than 2.08	0 with the channel out-of-range and warning bits set in status word	Under range
Greater than or equal to .52 and less than 1.00	Greater than or equal to 2.08 and less than 4.00	0 with the channel warning bit set in the status word	Low warning range
1.00 . . . 3.00 . . . 4.999	4.00 . . . 12.00 . . . 19.996	0 . . . 2048 . . . 4095	Recommended operating range
Greater than 4.999 and less than 5.12	Greater than 19.996 and less than 20.480	4095 with the channel warning bit set in the status word	High warning range
Greater than or equal to +5.12	Greater than or equal to 20.480	4095 with the channel out-of-range an warning bits set in the status word	Over range

MIC143 Analog Input Signals (with 12-bit resolution)		
Voltage	**Data Count (Decimal)**	**Operating Results**
Less than or equal to –10.24	0 with the channel out-of-range and warning bits set in the status word	Under range
Less than –10.00 and greater than –10.24	0 with the channel warning bit set in the status word	Low warning range
–10 . . . 0 . . . +9.995	0 . . . 2048 . . . 4095	Recommended operating range
Greater than +9.995 and less than +10.24	4095 with the channel warning bit set in the status word	High warning range
Greater than or equal to +10.24	4095 with the channel out-of-range and warning bits set in the status word	Over range

MIC144 Analog Input Signals (with 15-bit resolution)		
Voltage	**Data Count (Decimal)**	**Operating Results**
Less than 0	0 with the channel out-of-range and warning bits set in the status word	Under range
0 . . . 5 . . . 10	0 . . . 16,000 (3E80 hex) . . . 32,000 (7D00 hex)	Recommended operating range
Greater than 10.00 and less than 10.2397	32,001 ... 32,766 (7D01 ... 7FFE hex) with the channel warning bit set in the status word	High warning range
Greater than 10.2397	32,767 (7FFF hex) with the channel out-of-range and warning bits set in the status word	Over range

MIC146 Analog Input Signals (with 16-bit resolution)		
Voltage	**Data Count (Decimal)**	**Operating Results**
Less than or equal to −10.24	0 with the channel out-of-range and warning bits set in the status word	Under range
Less than −10.00 and greater than −10.24	1 ... 767 with the channel warning bit set in the status word	Low warning range
−10 . . . 0 . . . +10	768 . . . 32,768 . . . 64,768	Recommended operating range
Greater than +10.00 and less than +10.23970	64,769 ... 65,534 (FD01 ... FFFE hex) with the channel warning bit set in the status word	High warning range
Greater than +10.23970	65,535 with the channel's over-range bit set in the status word	Over range

MIC145 Analog Input Signals (with 14-bit resolution)			
Voltage	**Current (mA)**	**Data Count (Decimal)**	**Operating Results**
Less than .52	2.08	0 with the channel under-range and warning bits set in the status word	Under range
Greater than or equal to .52 and less than 1.00	Greater than or equal to 2.08 and less than 4.00	0 with the channel warning bit set in the status word	Low warning range
1 . . . 3 . . . 5	4 . . . 12 . . . 20	0 . . . 6400 (1900 hex) . . . 12,800 (3200 hex)	Recommended operating range
Greater than 5.00 and less than 5.12	Greater than 20.00 and less than 20.48	12,801 ... 13,183 (3201 ... 337F hex) with the channel out-of-range bit set in the status word	High warning range
Greater than or equal to 5.12	Greater than or equal to 20.48	13,184 (3380 hex) with the channel out-of-range and warning bits set in the status word	Over range

Each fixed analog input is available for reading approximately once every 50 ms. Therefore reading all four channels requires approximately 200 ms.

The Analog Output Channels

Each of the two output channels is also addressed to a word in the PLC's user data memory. The resolution of the fixed analog outputs is always 12 bits.

Analog Output Signals (with 12-bit resolution)			
Voltage	**Current (mA)**	**Data Count (Decimal)**	**Operating Results**
0	**4**	0	
.	.	.	
.	.	.	
.	.	.	
5	12	2047	Operating Range
.	.	.	
.	.	.	
.	.	.	
9.9975	19.996	4095	

Addressing A120 I/O

110CPU512 or 110CPU612 models may use an optional A120 I/O expansion capability. When A120 I/O is used, it also needs to be I/O mapped in that PLC's system configuration memory.

You must edit the I/O map via panel software to address A120 I/O. Each A120 I/O module is assigned a location in the rack where it is housed.

Each physical rack connected to the PLC—*racks #2*, *#3* and *#4*—can have up to five I/O locations in it. As many as 20 A120 I/O modules (locations) can be addressed in a Micro PLC's I/O map. The first five locations are reserved for fixed I/O capabilities, and locations 06 ... 20 are for A120 I/O modules. The PLC reserves the following references for expanded I/O addressing:

- References 00017 ... 00080 for addressing discrete A120 output points
- References 10017 ... 10080 for addressing discrete A120 input points
- References 30002 ... 30005 and 30011 ... 30030 for addressing register/analog inputs from A120 I/O
- References 40003 ... 40010 are reserved for mapping register outputs from A120 I/O

☞ **Note** These reserved references may be used for addressing fixed I/O resources in other PLCs on an I/O expansion link if they are not used for A120 I/O addressing.

An Example: A Micro PLC with One Rack of A120 I/O

The following example uses two I/O map screens from MODSOFT Lite. The system being I/O mapped comprises a 110CPU51200 PLC and one rack of five A120 I/O modules—two BDAP212s and three BDAP216s.

The PLC uses only one of its discrete I/O points for this application. Therefore, a total of six I/O locations are used in this configuration—**MIC128** for the fixed I/O points, and five locations for the A120 I/O modules.

Screen 1 shows the I/O map for fixed I/O resources of the 110CPU51200 PLC. This PLC is considered rack #1 with respect to A120 I/O expansion. Note that only locations 1, 2, and 3 in rack #1 can be accessed.

☞ **Note** In MODSOFT Lite, each rack is I/O mapped on a separate screen. You can move forward and backward through the screens—i.e., through the racks—by pushing <**PgUp**> and <**PgDn**>.

The A120 I/O in rack 2 is I/O addressed in the I/O map shown in screen 2. The A120 input points have been mapped to references 10017 ... 10032 and the output points to 00017 ... 00080 in the PLC's user data memory.

Altogether, this configuration uses 56 discrete inputs, 80 discrete outputs, and one counter/timer register input.

```
Utility           Del I/O  HoldTme  Get I/O                                Quit
F1———————F2———————F3———————F4———————F5———————F6———————F7—Lev 8—F8—OFF——F9———
                                   MICRO I/O MAP

                                     Parent I/O

  PLC         :  MICRO 512/00             Holdup Time  :  N/A
  Used Inputs :  040 of 512 Points        Used Outputs :  016 of 512 Points
  Next Input  :  10017_                   Next Output  :  00017
 ═══════════════════════════════════════════════════════════════════════════
 Location      Type        Reference Numbers       Data     Description
                         Inputs        Outputs     Type

 DISC   I/O:  MIC128   10001-10016   00001-00016   BIN   16@24V  I/O 12 RY
 INT/CTR IN:  MIC140   10081-10088        -        BIN   8 INTPT/CNTR INP
 TMR/CTR IN:  MIC147   30001-30001        -        BIN   16BIT TMR/CNTR VAL
 ANALOG I/O:  NotAvl        -             -
 DATA TRANS:  NotAvl        -             -
```

Screen 1. I/O Map for the Fixed I/O Locations (Rack 1)

```
Utility           DelDrop  Get I/O                                         Quit
F1———————F2———————F3———————F4———————F5———————F6———————F7—Lev 8—F8—OFF——F9———
                                   I/O MAP
                             A120 EXPANSION I/O

   PLC           :      MICRO 512/00   Rack          :             2
   Number Inputs :      56             Number Outputs:             80

  ═══════════════════════════════════════════════════════════════════
          Module     Reference Numbers           Data       Module
   Slot    Type     Input         Output         type     Description

   201     DAP212 10017 -10024   00017 -00024           Help Alt-H
   202     DAP212 10025 -10032   00025 -00032           Help Alt-H
   203     DAP216                00033 -00048           16-O 24V
   204     DAP216                00049 -00064           16-O 24V
   205     DAP216                00065 -00080           16-O 24V

                    _
```

Screen 2. I/O Map for A120 I/O Locations (Rack 2)

Addressing I/O on an Expansion Link

An I/O expansion link is created by daisy chaining up to five Micro PLCs together via cable connections at their RS-485 ports. One PLC must be configured as the parent, and the remaining units must be configured as child PLCs.

The Parent PLC

The parent PLC can address all its own fixed I/O resources as well as any fixed I/O resources residing in the child PLCs.

The fixed I/O locations of the parent PLC are automatically addressed for you. References for mapping additional I/O points from the parent are available as follows:

Physical Inputs	References (in User Data Memory)	Physical Outputs
	00001 ... 00012	Local fixed discrete outputs (12)
	00017 ... 00080	Reserved (A120 or child-based discretes)
	00081	Battery OK coil
Local fixed discrete inputs (16)	10001 ... 10016	
Reserved (A120 or child-based discretes)	10017 ... 10080	
Local interrupt/ counter inputs (8)	10081 ... 10088	
Reserved (child-based interrupt/timers)	10089 ... 10120	
Local timer/ counter input (1)	30001	
Reserved (child-based timers/counters)	30002 ... 30005	
Local fixed analog inputs (4)	30006 ... 30010	
Reserved (child-based analog inputs)	30011 ... 30030	
	40001 ... 40002	Local fixed analog outputs (2)
	40003 ... 40010	Reserved (child-based analog outputs)
	40011	10 ms timer
	40012 ... 40019	Time-of-day clock

A Child PLC

When you select child operating mode for a PLC, the ladder logic operating system assumes by default that all the fixed I/O points available on that PLC will be controlled by the parent on the network. Therefore, no values are assigned to the I/O map of a child PLC in its default state.

The fixed I/O locations in the child can be mapped in a screen associated with the parent's I/O map.

☞ **Note** Any A120 I/O connected to a child PLC must be addressed by the child. A120 I/O in a child cannot be accessed or controlled by the parent over the I/O expansion link.

An Example: An Expansion Link with all Fixed I/O Controlled by the Parent

The system being configured in the following example consists of two 110CPU51200 PLCs, a parent and one child, on an I/O expansion link. The example shows three I/O map screens from MODSOFT Lite.

When you configure the parent, make sure that it is set for at least one child. The operating system will not allow the parent to access any of the child's I/O resources unless you have specified the existence of that child in the parent's configuration.

Screens 1 and 2 show the I/O maps for the fixed I/O locations in the parent and child that will be controlled by references in the parent's memory. Both I/O map screens are accessed while the programming panel is connected to the parent.

```
Utility          Del I/O  HoldTme  Get I/O                                Quit
F1———————F2———————F3———————F4———————F5———————F6———————F7—Lev 8—F8—OFF——F9———
                                  MICRO I/O MAP

                                    Parent I/O

 PLC         :  MICRO 512/00            Holdup Time  :  N/A
 Used Inputs :  040 of 512 Points       Used Outputs :  016 of 512 Points
 Next Input  :  10017_                  Next Output  :  00017
 ═══════════════════════════════════════════════════════════════════════════
 Location      Type        Reference Numbers       Data     Description
                         Inputs        Outputs     Type

 DISC   I/O:  MIC128   10001-10016   00001-00016   BIN   16@24V  I/O 12 RY
 INT/CTR IN:  MIC140   10081-10088        -        BIN   8 INTPT/CNTR INP
 TMR/CTR IN:  MIC147   30001-30001        -        BIN   16BIT TMR/CNTR VAL
 ANALOG I/O:  NotAvl        -             -
 DATA TRANS:  NotAvl        -             -
```

Screen 1. I/O Map for the Fixed I/O Points in the Parent

```
F1———F2———F3———F4———F5———F6———F7-Lev 8-F8-OFF——F9——
                          MICRO I/O MAP

                  Parent's I/O Sharing with Child #1

 PLC         :  MICRO                    Holdup Time  :  3     x100ms
 Used Inputs :  080 of 512 Points        Used Outputs :  096 of 512 Points
 Next Input  :  10041                    Next Output  :  00033
 ═══════════════════════════════════════════════════════════════════════
 Location      Type        Reference Numbers       Data     Description
                         Inputs        Outputs     Type

 DISC   I/O:  MIC128   10017-10032   00017-00032   BIN   16@24V  I/O 12 RY
 INT/CTR IN:  MIC140   10033-10040        -        BIN   8 INTPT/CNTR INP
 TMR/CTR IN:                -             -
 ANALOG I/O:                -             -
 DATA TRANS:                -             -

                  _
```

Screen 2. I/O Map for the Fixed I/O Points in the Child Accessed by the Parent

Notice that the location types used in the I/O map for the child place all the available fixed discrete input and relay output locations of the child under the control of the parent. **MIC128** maps all 16 of the child's 24 VDC inputs to references 10017 ... 10032 in the parent's memory and the 12 relay outputs to references 00017 ... 00032 in the parent's user data memory; **MIC140** maps the high-speed inputs to references 10033 ... 10040 in the parent's user data memory.

As a result, the I/O map screen that appears when the programming panel (see screen 3 below) is attached to the child shows no location types in it:

```
Utility           Del I/O  HoldTme  Get I/O                          Quit
F1———F2———F3———F4———F5———F6———F7-Lev 8-F8-OFF——F9——
                          MICRO I/O MAP

                            Child I/O

 PLC         :  MICRO 512/00             Holdup Time  :  3     x100ms
 Used Inputs :  000 of 512 Points        Used Outputs :  000 of 512 Points
 Next Input  :  10001                    Next Output  :  00001
 ═══════════════════════════════════════════════════════════════════════
 Location      Type        Reference Numbers       Data     Description
                         Inputs        Outputs     Type

 DISC   I/O:                -             -
 INT/CTR IN:                -             -
 TMR/CTR IN:                -             -
 ANALOG I/O:  NotAvl        -             -
 DATA TRANS:                -             -

                  _
```

Screen 3. I/O Map for the Fixed I/O Points in the Child

Splitting Fixed I/O between Parent and Child PLCs

A child PLC has the option of splitting its fixed I/O resources with the parent—i.e., the child retains control over some of its own fixed I/O while the parent controls the rest. When fixed I/O resources are split, the I/O points controlled by the child must be addressed in the child's I/O map, and the I/O points controlled by the parent must be addressed in the parent's I/O map.

The key to splitting I/O is choosing the proper location types (see the table below) and placing them in the I/O map screens of the parent and child.

For example, if a child has 12 fixed FET outputs, you can I/O address one PLC's I/O map with a location type of **MIC138** (putting 8 FET outputs under its control) and the other I/O map with a location type of **MIC139** (putting the remaining four FET outputs under the other PLC's control).

I/O Map Location Types for Fixed I/O

I/O Type		Location Type	110CPU Models
Discrete	16 ... 24 VDC in / 12 relay out	MIC128	31100, 41100, 51200, 61200, 61204
	16 ... 24 VDC in / 8 relay out	MIC129	
	16 ... 24 VDC in / 4 relay out	MIC130	
	16 ... 115 VAC in / 8 triac out 4 relay out	MIC131	31101, 41101, 51201
	16 ... 115 VAC in / 8 triac out	MIC132	
	16 ... 115 VAC in / 4 relay out	MIC133	
	16 ... 230 VAC in / 8 triac out 4 relay out	MIC134	31102, 41102, 51202
	16 ... 230 VAC in / 8 triac out	MIC135	
	16 ... 230 VAC in / 4 relay out	MIC136	
	16 ... 24 VDC in / 12 FET out	MIC137	31103, 41103, 51203, 61203
	16 ... 24 VDC in / 8 FET out	MIC138	
	16 ... 24 VDC in / 4 FET out	MIC139	
Counter / Interrupt	8-bit counter/interrupt in	MIC140	All 512 & 612 Models
Analog (for 612 Models only) All output channels have 12-bit resolution	4 in (0 ... 10, 12-bit), 2 out	MIC141	61200, 61203, 61204
	4 in (1 ... 5, 12-bit), 2 out	MIC142	
	4 in (± 10, 12-bit), 2 out	MIC143	
	4 in (0 ... 10, 15-bit), 2 out	MIC144	
	4 in (1 ... 5, 14-bit), 2 out	MIC145	
	4 in (± 10), 2 out	MIC146	
Timer / Counter	16-bit timer/Current count value	MIC147	Default is NONE for all models
Generalized Data Transfer	1 word in, 1 word out	MIC148	
	2 words in, 2 words out	MIC149	
	4 words in, 4 words out	MIC150	
	8 words in, 8 words out	MIC151	

Both PLCs will read the same input data. Shared input data will not cause conflicts between the parent and child, and, therefore, the same fixed *inputs* can be mapped in both the parent and the child.

However, having both PLCs write the same output data can introduce errors. If the same outputs are mapped in both PLCs, the system will log an error against the parent, and it will be marked *unhealthy* in the PLC status table.

An Example: Splitting I/O

The following example shows two I/O map screens from MODSOFT Lite. They show how the 12 fixed relay outputs of a 110CPU51200 PLC configured as a child can be split between it and its parent.

Screen 1 below is the map of the child I/O to be accessed by the parent. This I/O map screen is created while the programming panel is connected to the parent PLC. The location type for the discrete I/O is **MIC129**, indicating that the parent can access eight of the child's fixed relay outputs.

```
Utility          Del I/O  HoldTme  Get I/O                              Quit
F1———F2———F3———F4———F5———F6———F7—Lev 8—F8—OFF——F9——
                                MICRO I/O MAP

                        Parent's I/O Sharing with Child #1

 PLC         :  MICRO                    Holdup Time  :  3     x100ms
 Used Inputs :  056 of 512 Points        Used Outputs :  024 of 512 Points
 Next Input  :  10033                    Next Output  :  00025
 ═══════════════════════════════════════════════════════════════════════
 Location      Type        Reference Numbers       Data     Description
                         Inputs        Outputs     Type

 DISC   I/O:   MIC129   10017-10032   00017-00024   BIN   16@24V  I/O 8 RY
 INT/CTR IN:              -             -
 TMR/CTR IN:              -             -
 ANALOG I/O:              -             -
 DATA TRANS:              -             -
```

8 relay outputs accessed by the parent and mapped to references 00017 ... 00024

Screen 1: Child I/O accessed by the parent

Screen 2 is the map of the child I/O that remains under the control of the child. This I/O map is created while the programming panel is connected to the child PLC. The location type for the discrete I/O is **MIC130**, indicating that the child maintains control over four of its fixed relay outputs.

```
Utility          Del I/O  HoldTme  Get I/O                                Quit
F1———F2———F3———F4———F5———F6———F7—Lev 8—F8—OFF——F9——
                                MICRO I/O MAP

                                  Child I/O

 PLC         :  MICRO 512/00            Holdup Time  :  3     x100ms
 Used Inputs :  016 of 512 Points       Used Outputs :  008 of 512 Points
 Next Input  :  10033                   Next Output  :  00033
 ═══════════════════════════════════════════════════════════════════════
 Location      Type        Reference Numbers       Data     Description
                         Inputs        Outputs     Type

 DISC   I/O:   MIC130   10017-10032   00025-00032   BIN   16@24V  I/O 4 RY
 INT/CTR IN:                -             -
 TMR/CTR IN:                -             -
 ANALOG I/O:   NotAvl       -             -
 DATA TRANS:                -             -
```

4 relay outputs controlled by the parent and mapped to references 00025 ... 00032

Screen 2: Fixed I/O resources controlled by the child

Generalized Data Transfer

The I/O expansion link is fundamentally a capability for accessing systemwide I/O resources for a logic program running in a single PLC—the parent. However, because each child PLC on the link has the ability to store its own user logic program and service its own I/O and communication ports, a certain amount of *coprocessing* can occur in the various CPUs on the link.

Generalized data transfer is a tool that allows the parent and child PLCs on the link to share non-control data. It utilizes the cable connections of the expansion link to pass data to and from each other.

The parent can share generalized data with any and all child PLCs; a child can share generalized data only with the parent.

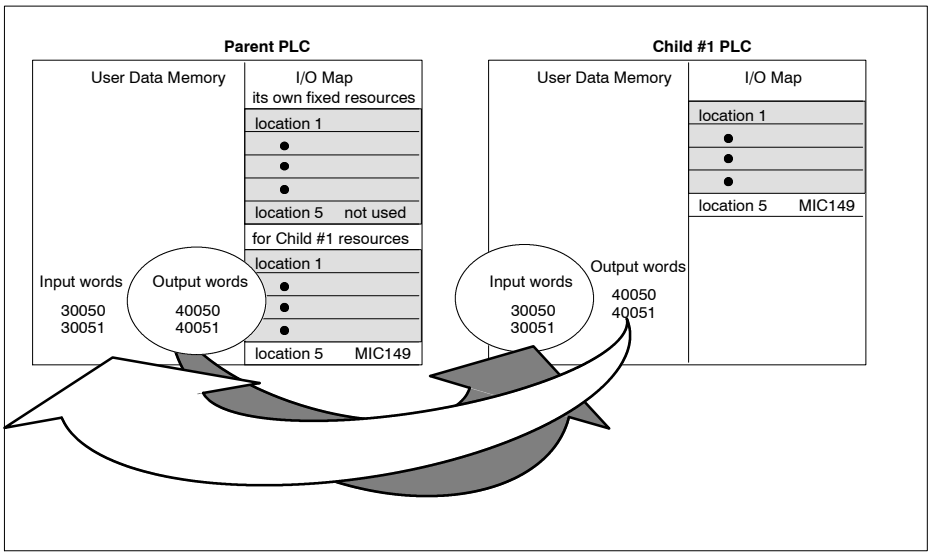
The parent and child PLCs on an I/O expansion network can bidirectionally transfer a selectable number of non-control data words over the I/O expansion network. Fixed I/O location #5 on all Modicon Micro PLCs is reserved for this generalized data transfer capability.

You can select either the input/output words to be reserved in the User Data Memory of the parent and child PLCs by specifying one of the following location types in the I/O maps of the parent and child:

- ❒ **MIC148**, specifying one input word (of 1*x* or 3*x* references) and one output word (of 0*x* or 4*x* references)
- ❒ **MIC149**, specifying two input words (of 1*x* or 3*x* references) and two output words (of 0*x* or 4*x* references)
- ❒ **MIC150**, specifying four input words (of 1*x* or 3*x* references) and four output words (of 0*x* or 4*x* references)
- ❒ **MIC151**, specifying eight input words (of 1*x* or 3*x* references) and eight output words (of 0*x* or 4*x* references)

To set up a generalized data transfer between a parent and child PLC, you must specify the same location type in the I/O maps of the child and of the parent. When the programming panel is connected to the parent PLC, specify the generalized data transfer location type in the I/O map that describes the fixed I/O resources of the child, not in the I/O map that describes the fixed I/O resources of the parent.

Here is an illustration of the generalized data transfer process:



Programming Note for 512XX and 612XX Controllers

In very small user logic test situations (e.g., using a contact to switch a coil as a fast oscillator), in Single or Child mode operation, the fast scantime [2.5 milliseconds per 1000 nodes programmed in a 512/612 Micro] may inhibit correct operation of the internal hardware output LED circuit and the internal output device circuit.

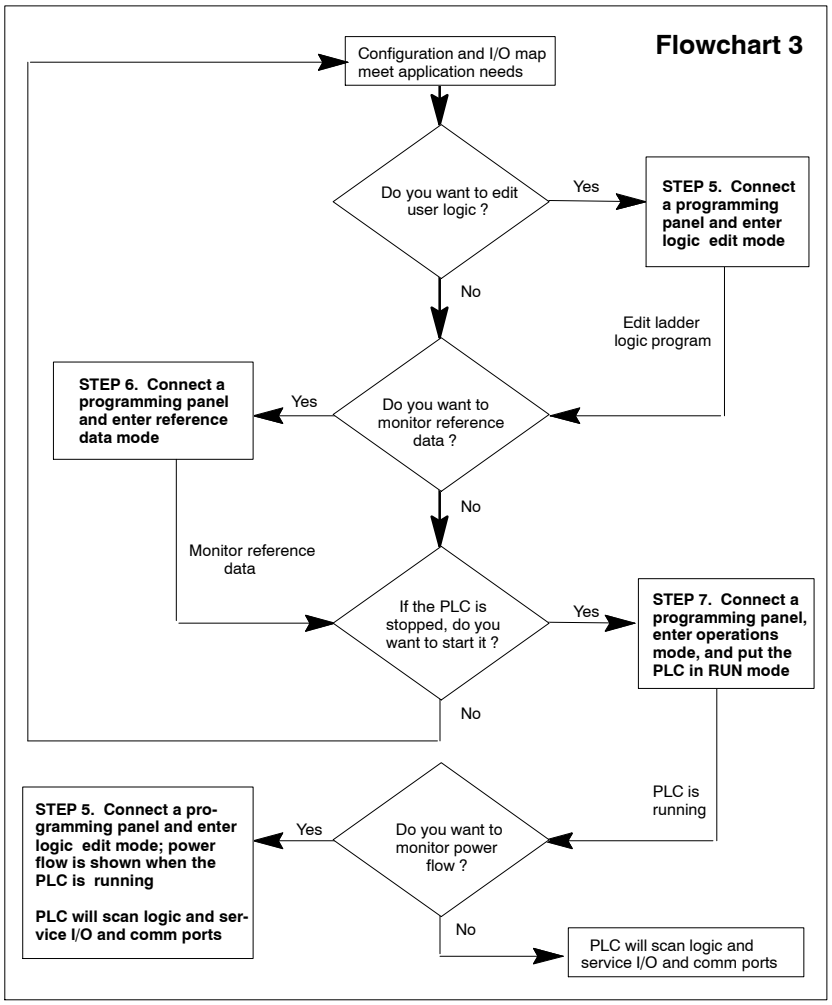
Both circuits react independently to user logic, so the LED may not reflect actual output operation.

The more logic that is programmed, the longer the scantime will be; and both LEDs and output circuits will then show the correct programmed response.

Consult the hardware manual provided with your unit to determine the response or switching time of the output device. [For example, the internal output relay has a maximum switching rate of 5 Hz.]

When the Micro is set up as a parent, this hardware restriction should not be seen, since each added Child Micro in the Parent configuration adds 3 milliseconds to the scantime.

PLC Operations



Once the PLC has been configured for its desired operating mode and the I/O locations have been addressed in the I/O map, you can:

- ❒ Create or edit your ladder logic program
- ❒ Monitor and edit reference data
- ❒ Start and stop the PLC
- ❒ Monitor power flow in a running application program

In the next chapters, we will look closely at the ladder logic instruction set and how it can be used to create application programs.

Chapter 3
Essentials of Ladder Logic Programming

- Segments and Networks
- Standard Ladder Logic Elements
- Application Example: A Motor Start/Stop Circuit
- Standard Modicon Micro PLC Instructions
- Instructions Available on Select Models of the Modicon Micro PLCs

Segments and Networks

Ladder Logic Segments

All the ladder logic required to control your application is stored in a logic *segment* in user memory. If you are calling subroutines as part of your application, the subroutine logic must be placed in a separate segment.

The Modicon Micro PLCs give you a configuration with two segments in it. Segment 1 is where all normally scheduled ladder logic used to control the application is stored. Segment 2 is where all subroutine logic is stored. Subroutines logic is scanned only when it is called, either from the ladder logic or from an external event that triggers an interrupt. Therefore segment 2 is not solved as part of the regular logic scan.

Ladder Logic Networks

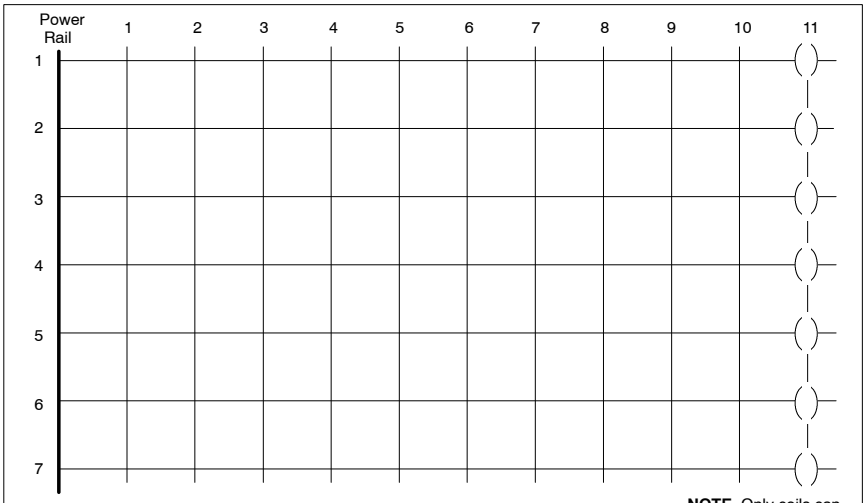
Each segment is composed of a group of contiguous *networks*. Each network is a small, clearly defined ladder diagram bounded on the left by a power rail and on the right by a rail that, by convention, is not displayed. The ladder is seven rungs high by eleven columns wide.

The intersection of each rung and column in the network is called a node—each network contains 77 nodes.

There is no prescribed limit on the number of networks that can be put in a segment—overall program size is limited by the amount of *user program memory* available in the CPU and by the time it takes for the CPU to scan the ladder logic program.

Placing Relay Logic and Instructions in a Network

Each time you use an *relay logic element*—e.g., a contact, a coil, a horizontal short—in ladder logic, the element consumes one node in the logic network.



NOTE Only coils can be shown in column 11

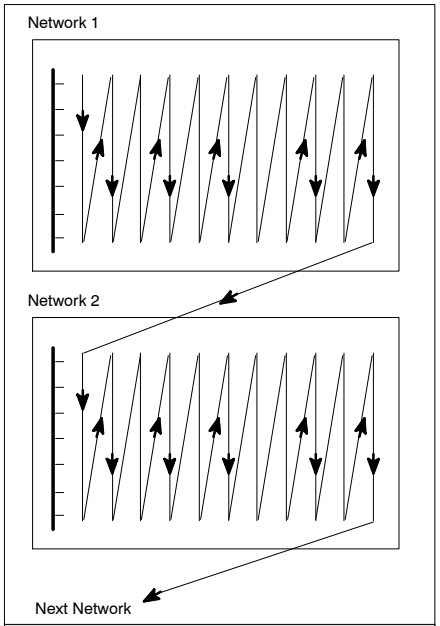
Ladder Logic Network Structure

An *instruction* in ladder logic may consume one, two, or three nodes in a network, depending on the instruction type. A counter instruction, for example, is a two-high nodal instruction—it consumes two contiguous nodes that must be one over the other. An ADD instruction, on the other hand, is a three-high nodal instruction consuming three contiguous nodes stacked over each other.

How Ladder Logic Is Solved

A Modicon Micro PLC scans the ladder logic program sequentially in the following order:

- Segment by segment
- Network 1 through network *n* sequentially within each segment
- Node by node within each network, starting in the upper left node of the ladder and moving top to bottom, then left to right



Power Flow in and between Ladder Logic Networks

Relay Logic Elements

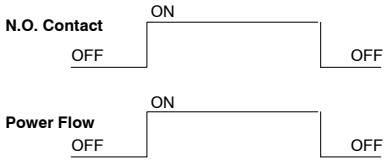
There are three general types of relay logic elements used in ladder logic programming—contacts, coils, and shorts.

Each relay logic element consumes one node in a ladder network.

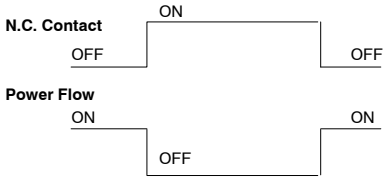
Relay Contacts

Contacts are used to pass or inhibit power flow in a ladder logic program. Four kinds of contacts may be used:

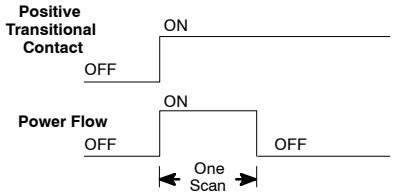
- ❒ The normally open (N.O.) contact, which passes power when its referenced coil or input is ON:



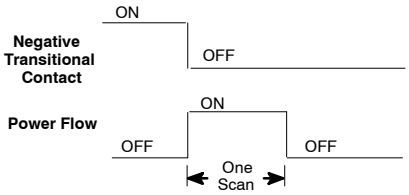
- ❒ The normally closed (N.C.) contact, which passes power when its referenced coil or input is OFF:



- ❒ The positive transitional contact, which passes power for only one scan as the contact or coil transitions from OFF to ON:



- ❒ The negative transitional contact, which passes power for only one scan as the contact or coil transitions from ON to OFF:



The symbols used in ladder logic to represent contact types are shown in the table below.

Element	Symbol	Function	Memory Utilization
N.O. Contact		Passes power when its referenced coil or input is ON	Can be referenced to a logic coil in a 0*x* register or to a discrete input in a 1*x* register
N.C. Contact		Passes power when its referenced coil or input is OFF	Can be referenced to a logic coil in a 0*x* register or to a discrete input in a 1*x* register
Positive Transitional Contact		Passes power for one scan as the contact or coil transitions from OFF to ON	Can be referenced to a logic coil in a 0*x* register or to a discrete input in a 1*x* register
Negative Transitional Contact		Passes power for one scan as the contact or coil transitions from ON to OFF	Can be referenced to a logic coil in a 0*x* register or to a discrete input in a 1*x* register

Normal and Memory-retentive Coils

Element	Symbol	Function	Memory Utilization
Normal Coil	–( )–	Turns OFF when power is removed	A discrete output value represented by a 0*x* reference number; may be used internally in the logic program or externally to a discrete output
Memory-retentive Coil	–(M)–	Coil comes back in the same state when power is restored for one scan	A discrete output value represented by a 0*x* reference number; may be used internally in the logic program or externally to a discrete output

A coil is a discrete output value represented by a 0*x* reference bit. Because output values are updated in state RAM by the CPU, 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.

A coil is either ON or OFF, depending on power flow. 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—*normal* coils and *memory-retentive* coils. When power is applied or restored to a normal coil, any value previously held by the coil is cleared prior to the first logic scan of the PLC. With a memory-retentive coil, the value previously held by the coil is retained for one scan, then the logic takes control.

Displaying Coils in a Network

A ladder network can contain a maximum of seven coils. No logic elements except coils are allowed in the eleventh column. If a coil appears on a rung in a column other than 11, no other logic element can be placed to the right of the coil on that rung.

Vertical and Horizontal Shorts

Shorts are simply straight-line connections between instruction blocks and/or contacts in a ladder logic network.

A *vertical* short connects contacts or instruction blocks one above the other in a network column. Vertical shorts can also be used to connect inputs or outputs to create either/or conditions such as the one illustrated below. When two contacts are connected by a vertical short, power is passed when one or both contact(s) receive power. A vertical short does not consume any user memory.

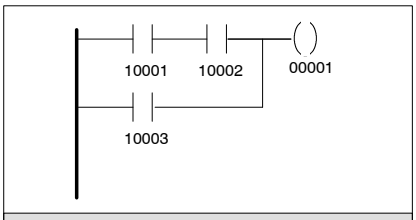
Horizontal shorts are used to expand a rung in a ladder logic network without breaking the power flow. Each horizontal short used in a program consumes one word of user logic memory.

On the following page are two examples of how horizontal and vertical shorts can be used together with relay contacts to create ladder logic.

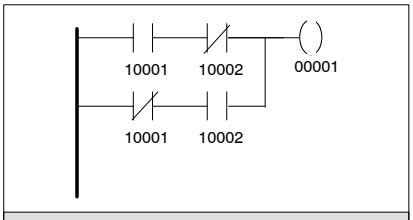
The first example is a simple either/or condition—the top rung of ladder contains two N.O. contacts (10001 and 10002), and the lower rung contains a single contact (10003) followed by a horizontal short. A vertical short connects the two rungs after the second column. Power can pass through the network to energize coil 00001 when *either* contacts 10001 and 10002 are energized *or* when contact 10003 is energized.

The second example shows an Exclusive-OR circuit built with similar contacts and shorts. This circuit can be used to prevent coil 00001 from energizing when two conditions, represented by contact 10001 and contact 10002, are activated simultaneously.

In both examples, the vertical shorts, which do not consume any user program memory, are treated as part of the node in which contact 10002 is programmed.

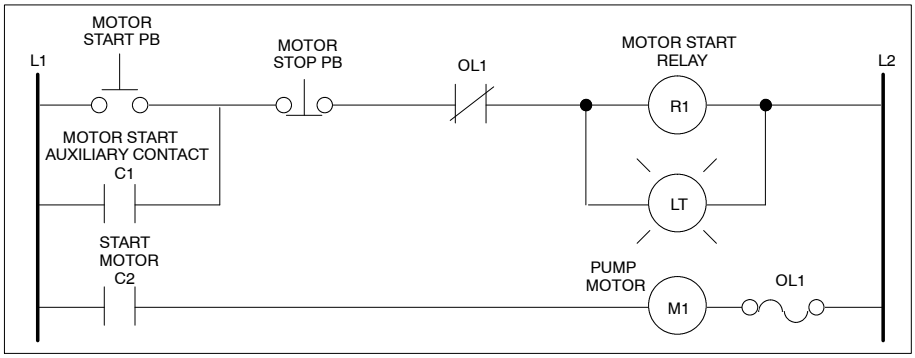


Example 1: Either/Or Relay Logic



Example 2: Exclusive-OR Relay Logic

Application Example: A Motor Start/Stop Circuit



Above is an example of a standard across-the-line electrical diagram for a pushbutton-activated motor start/stop circuit.

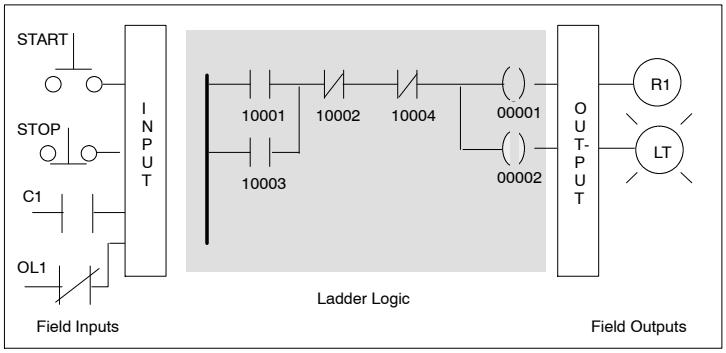
Pushing the *motor start* pb energizes motor control relay R1 and closes contact C2 to start motor M1. The auxiliary contacts on motor control relay C1 also close, allowing the motor start/stop circuit to be latched ON. Two things can cause relay R1 to drop out:

- An overload (OL1) on motor M1
- The *motor stop* pb is pushed

Now let's look at an implementation of the same circuit using contacts, coils, and shorts in a ladder logic network.

We see in the illustration below that the sequence of operation remains essentially the same when the motor start/stop circuit is designed for the PLC. The big difference is that all the I/O points are wired directly to input/output units built into the PLC system and the actual control is programmed in ladder logic in the PLC.

The ladder logic implementation allows greater flexibility of control and decreased development time, since all the *hard-wiring* between points of control is done electronically.



Chapter 4
Counters and Timers

- Counter Instructions
- Timer Instructions
- Application Example: A Real-time Clock with a millisecond Timer

Counter Instructions

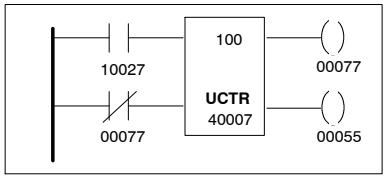
Two counter instructions are provided. The up-counter (UCTR) counts up from 0 to a preset value, and the down-counter (DCTR) counts down from a preset value to 0. Both are two-high nodal instructions.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Up-counter	I — 3*x*, 4*x*, or K* — O I — **UCTR** 4*x* — O	*Top:* ON initiates counter *Bottom:* 0 = reset 1 = enabled	*Top:* counter preset *Bottom:* accumulated count	*Top:* count = preset *Bottom:* count < preset	Counts up from 0 to a preset value
Down-counter	I — 3*x*, 4*x*, or K* — O I — **DCTR** 4*x* — O	*Top:* ON initiates counter *Bottom:* 0 = reset 1 = enabled	*Top:* counter preset *Bottom:* accumulated count	*Top:* count = 0 *Bottom:* count > preset	Counts down from a preset value to 0

*K is an integer constant in the range 1 ... 999.

A Simple Up-counter Example

When contact 10027 is energized, the top input to UCTR receives power; since contact 00077 is also receiving power, the instruction is enabled. Each time contact 10027 transitions from OFF to ON, the accumulated count increments by 1. When the value reaches 100, the top output passes power—coil 00077 is energized, and coil 00055 is de-energized. Contact 00077 loses power when coil 00077 is energized, and the accumulated count is reset to 0 on the next scan. On the next scan, coil 00077 is de-energized; contact 00077 is re-energized and the UCTR is enabled.



Timer Instructions

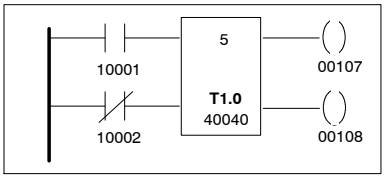
The four timer instructions can be used to time events or create delays in an application. The first three are two-high nodal instructions, and the millisecond timer is a three-high instruction.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
One-second timer	I — 3*x*, 4*x*, or K* — O I — **T1.0** 4*x* — O	*Top:* ON when bottom input = 1 *Bottom:* 0 = reset 1 = enabled	*Top:* timer preset *Bottom:* accumulated time	*Top:* time = preset *Bottom:* time < preset	Timer increments at intervals of one second
Tenth-of-a second timer	I — 3*x*, 4*x*, or K* — O I — **T0.1** 4*x* — O	*Top:* ON when bottom input = 1 *Bottom:* 0 = reset 1 = enabled	*Top:* timer preset *Bottom:* accumulated time	*Top:* time = preset *Bottom:* time < preset	Timer increments at intervals of 0.1 s
Hundredth-of a-second timer	I — 3*x*, 4*x*, or K* — O I — **T.01** 4*x* — O	*Top:* ON when bottom input = 1 *Bottom:* 0 = reset 1 = enabled	*Top:* timer preset *Bottom:* accumulated time	*Top:* time = preset *Bottom:* time < preset	Timer increments at intervals of 0.01 s
Millisecond timer	I — 3*x*, 4*x*, or K* — O I — 4*x* — O **T1MS** 0001	*Top:* ON when middle input = 1 *Middle:* 0 = reset 1 = enabled	*Top:* timer preset *Middle:* accumulated time *Bottom:* Always set to a constant value of 1	*Top:* time = preset *Middle:* time < preset	Timer increments at intervals of 1 ms

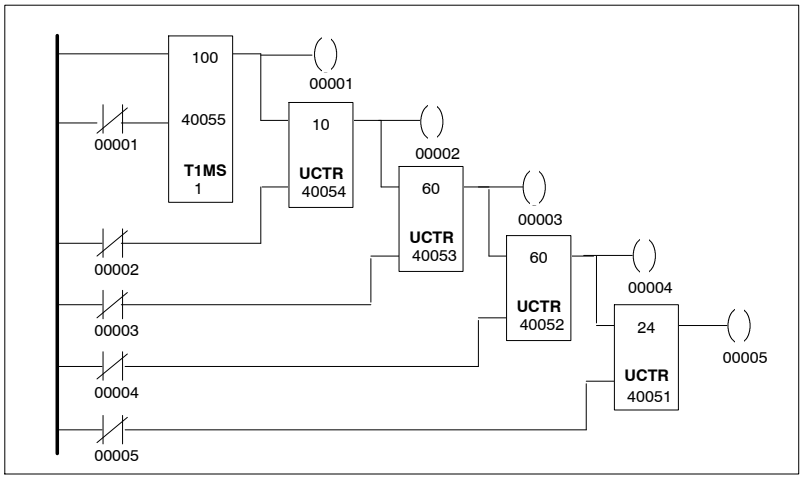
*K is an integer constant in the range 1 ... 999.

A One-second Timer Example

Here contact 10002 is closed—i.e., the timer is enabled—and the value contained in register 40040 is 0. Coil 00108 is ON and 00107 is OFF. When contact 10001 is closed, the count accumulates in register 40040 at one-second intervals until 5 is reached; coil 00107 goes ON and 00108 goes OFF. When contact 10002 is opened, the value in register 40040 is reset to 0, coil 00107 goes OFF, and 00108 goes ON.



Application Example: A Real-time Clock with a millisecond Timer



This example shows the ladder logic for a real-time clock with millisecond accuracy. The T1MS instruction is programmed to pass power at 100 ms intervals; it is followed by a cascade of four up-counters that store the the time respectively in hundredth-of-a-second units, tenth-of-a-second units, one-second units, one-minute units, and one-hour units.

When logic solving begins, the accumulated time value begins incrementing in register 40055 of the T1MS block. After ten one-millisecond increments, the top output passes power and energizes coil 00001.

At this point, the value in register 40053 in the timer is reset to 0. The accumulated count value in register 40054 in the first UCTR block increments by 1, indicating that 10 ms have passed.

Because the accumulated time count in T1MS no longer equals the timer preset, the timer begins to re-accumulate time in ms.

When the accumulated count in register 40054 of the first UCTR instruction increments to 10, the top output from that instruction block passes power and energizes coil 00002. The value in register 40054 then resets to 0, and the accumulated count in register 40051 of the second UCTR block increments by 1.

As the times accumulate in each counter, the time of day can be read in five holding registers as follows:

Register	Unit of Time
40055	hundredths-of-a-second (0 ... 10)
40054	tenths-of-a-second (0 ... 10)
40053	seconds (0 ... 60)
40052	minutes (0 ... 60)
40051	hours (0 ... 24)

Chapter 5
Basic Math Instructions

- ❒ Integer Math Instructions
- ❒ Application Example: Fahrenheit-to-Centigrade Conversion

Integer Math Instructions

Standard addition, subtraction, multiplication, and division instructions are provided for calculating integer math operations. Each of the four instructions is a three-high nodal instruction.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Integer Addition	I — 3*x*, 4*x*, or K* — O 3*x*, 4*x*, or K* **ADD** 4*x*	*Top:* ON enables a (val 1) + (val 2) operation	*Top:* value 1 *Middle:* value 2 *Bottom:* sum	*Top:* sum > 9999	Adds the values in the top and middle nodes, then stores the result in a 4*x* register in the bottom node
Absolute (no signs in the values) Integer Subtraction	I — 3*x*, 4*x*, or K* — O 3*x*, 4*x*, or K* — O **SUB** 4*x* — O	*Top:* ON enables a (val 1) – (val 2) operation	*Top:* value 1 *Middle:* value 2 *Bottom:* difference	*Top:* val 1 > val 2 *Middle:* val 1 = val 2 *Bottom:* val 1 < val 2	Subtracts the middle node value from the top node value and stores the difference in a 4*x* register in the bottom node
Integer Multiplication	I — 3*x*, 4*x*, or K* — O 3*x*, 4*x*, or K* **MUL** 4*x*	*Top:* ON enables a (val 1) x (val 2) operation	*Top:* value 1 *Middle:* value 2 *Bottom:* product (high order digits)	*Top:* echos the top input	Multiplies the values in the top and middle nodes, then stores the product in two contiguous 4*x* registers
Integer Division with remainder	I — 3*x*, 4*x*, or K* — O I — 3*x*, 4*x*, or K* — O **DIV** 4*x* — O	*Top:* ON enables a (val 1) / (val 2) operation Middle: 0 = fractional remainder 1 = decimal remainder	*Top:* value 1** *Middle:* value 2 *Bottom:* result (remainder in reg 4*x* + *1*)	*Top:* division successful *Middle:* if result > 9999 a value of 0 is returned *Bottom:* value 2 = 0	Divides the top node value by the middle node value, then stores the result in the 4*x* register in the bottom node and the remainder in register 4*x* + 1

*K is an integer constant in the range 1 ... 999.

** If value 1 of the DIV instruction is stored 3*x* or 4*x* registers, then the register shown in the top node is the first of two contiguous registers. The high-order half of value 1 is stored in the displayed register (3*x* or 4*x*) and the low-order half of value 1 is stored in the next contiguous register (3*x* + 1 or 4*x* + 1).

The MUL and DIV blocks require that two contiguous registers be used in the bottom node. The first of the two registers is seen in the block, and the presence of the second register is implicit.

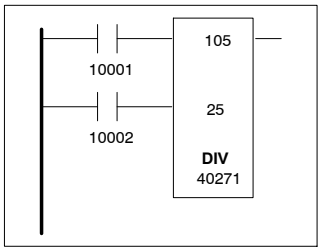
In the MUL instruction block, the high-order portion of the calculated product is stored in the first bottom-node register and the low-order portion of the product is stored in the second bottom-node register.

In the DIV instruction block, the quotient is stored in the first bottom-node register and the remainder is stored in the second bottom-node register. If you do not use a constant as the top-node value in a DIV instruction, then it the value must be placed in two contiguous 3*x* or 4*x* registers. The high-order half of the value is stored in the displayed register, and the low-order half of the value is stored in the implied register.

For example, if the top-node value is 105 and it were to be placed in two contiguous registers, 40025 and 40026, instead of being given as a constant, then register 40025 would contain all zeros and register 40026 would contain the value 105.

A DIV Example

Here is an example of a DIV operation where the top-node value, 105, is divided by the middle-node value, 25. The quotient (4) is stored in register 40271, and the remainder (5) is stored in register 40272.



When the middle input—contact 10002—is open, the remainder is expressed as a fraction (0005); when contact 10002 is closed, the remainder is expressed as a decimal (2000).

Application Example: Fahrenheit-to-Centigrade Conversion

This example implements the formula

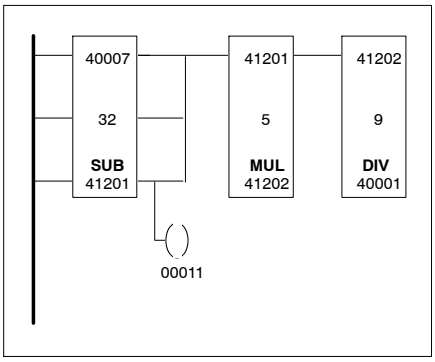
$$°C = (°F - 32) \times {}^{5}/_{9}$$

When the top input to the SUB instruction block receives power, the value in the middle node, 32, is subtracted from the value stored in register 40007, some number of degrees Fahrenheit. The difference is placed in register 41201.

The top input to the MUL instruction block then receives power, regardless of whether the subtraction result is positive, negative, or 0. In the case where the subtraction result is negative, coil 00011 is energized to indicate a negative value.

The value in the top-node register of the MUL block—register 41201—is then multiplied by 5 and the product is placed in register 41202 and implicit register 41203.

The top node in the DIV instruction block is then energized, and the value in registers 41202 and 41203 is divided by 9. The quotient, which is the temperature conversion in degrees Centigrade, is stored in register 40001 (and the remainder in implicit register 40002).



Note: The vertical short to coil 00011 (indicating a negative value) must be placed to the left of the vertical shorts that link the three SUB block output.

Chapter 6
Data Management Instructions

- Moving Register and Table Data
- Building a FIFO Stack
- Searching a Table
- Moving a Block of Data

Moving Register and Table Data

Three standard instruction blocks are provided for moving the data stored in registers and in tables of registers:

- ❒ A register-to-table (R→T) DX move
- ❒ A table-to-register (T→R) DX move
- ❒ A table-to-table (T→T) DX move

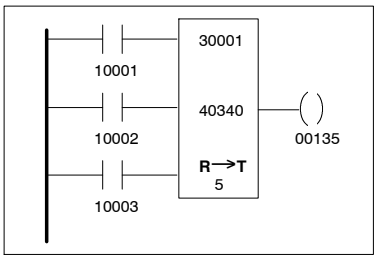
A Modicon Micro PLC system can accommodate the transfer of one register per scan for each instruction in a ladder logic program.

Each is a three-high nodal instruction.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Register-to-table move	I — 0*x*, 1*x*, * 3*x*, or 4*x* — O I — 4*x* — O I — R→T K**	*Top:* ON moves data and increments pointer *Middle:* ON freezes the pointer *Bottom:* ON resets the pointer to 0	*Top:* source register *Middle:* pointer to the target register (4*x* + 1) in the destination table *Bottom:* Table length*	*Top:* echos the top input *Middle:* pointer = table length	Copies a 16-bit pattern in a source register to a register in the destination table; the destination register is pointed to by the 4*x* register in the middle node
Table-to-register move	I — 0*x*, 1*x*, * 3*x*, or 4*x* — O I — 4*x* — O I — T→R K**	*Top:* ON moves data and increments pointer *Middle:* ON freezes the pointer *Bottom:* ON resets the pointer to 0	*Top:* source table *Middle:* pointer to the destination register (4*x* + 1) *Bottom:* Table length*	*Top:* echos the top input *Middle:* pointer = table length	Copies the bit pattern of a register in the source table to a destination register (register 4*x* + 1 in the middle node)
Table-to-table move	I — 0*x*, 1*x*, * 3*x*, or 4*x* — O I — 4*x* — O I — T→T K**	*Top:* ON moves data and increments pointer *Middle:* ON freezes the pointer *Bottom:* ON resets the pointer to 0	*Top:* source table *Middle:* pointer to the target register (4*x* + 1) in the destination table *Bottom:* Table length*	*Top:* echos the top input *Middle:* pointer = table length	Copies the bit pattern of a register in the source table to a register in the same position in a destination table; the destination register is pointed to by the 4*x* register in the middle node

* If you use a 0*x* or 1*x* reference, it must be given as a multiple of 16 + 1 (1, 17, 33, etc.), and it implies the use of 16 discrete bits (1 ... 16, 17 ... 32, 33 ... 48, etc.).

** K is an integer constant in the range 1 ... 255.



The ladder logic example shown above moves the value stored in register 30001 into a destination table of five holding registers, 40341 ... 40345. One 30001 register value is moved into one of the table registers in every scan.

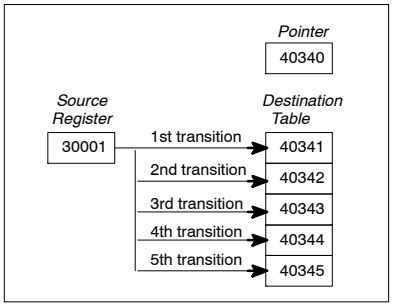
The pointer to the destination table—register 40340—is specified in the middle node of the register-to-table instruction block, and the number of holding registers in the table, 5, is specified in the bottom node.

When contact 10001 transitions ON for the first time, the current contents of register 30001 are copied to register 40341, the first of five contiguous registers in the destination table. The first table in the destination register is always the next contiguous register after the pointer reference number given in the middle node of the instruction block. When this DX move takes place, the value in the pointer register increments from 0 to 1.

In the next scan of contact 10001, the contents of register 30001 are copied into register 40432, the second register in the destination table; the value in the pointer register increments from 1 to 2.

This process continues until the contents of register 30001 are copied into register 40345 in the table and the pointer value has incremented to 5. At this point, the middle output from the block passes power and energizes coil 00135.

No further register-to-table moves are possible while the value of the pointer equals the table length specified in the bottom node of the block.



If, after the second transition of contact 10001, contact 10002 were to become energized, the pointer value would be frozen-i.e., it could not be incremented or decremented—and subsequent transitions of contact 10001 would cause the current value in register 30001 to be copied to register 40343.

If contact 10003 is energized, the value of the pointer is reset to 0.

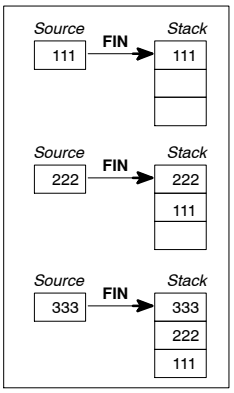
Building a FIFO Stack

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
First-in to a queue stack	I — 0*x*, 1*x*, * 3*x*, or 4*x* — O 4*x* — O **FIN** K** — O	*Top:* ON inserts a bit pattern in the top of the stack	*Top:* The source register in the stack *Middle:* pointer to the register in the stack where the source bits will be inserted *Bottom:* stack length*	*Top:* echos the top input *Middle:* stack is full *Bottom:* stack is empty	Copies a 16-bit pattern into a register at the top of a stack; the table begins at register 4*x* + 1 of the middle node
First-out of a queue stack	I — 4*x* — O 0*x* or 4*x* — O **FOUT** K** — O	*Top:* ON removes the bit pattern from the bottom of the stack	*Top:* pointer to the source register in the stack *Middle:* destination register where source bits will be moved *Bottom:* stack length*	*Top:* echos the top input *Middle:* stack is full *Bottom:* stack is empty	Moves the bit pattern in the bottom register of the stack to a destination register out of the stack

* If you use a 0*x* or 1*x* reference, it must be given as a multiple of 16 + 1 (1, 17, 33, etc.), and it implies the use of 16 discrete bits (1 ... 16, 17 ... 32, 33 ... 48, etc.).

** K is an integer constant in the range 1 ... 255.

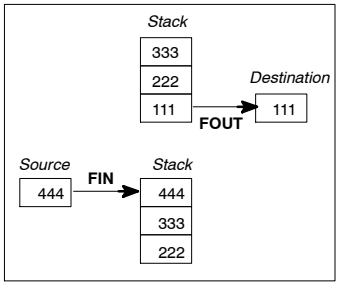
The two instructions above let you queue data into a first-in/first-out stack. The FIN instruction copies the bit pattern of a register or of 16 discretes into a register at the top of a table (or stack) of holding registers.



The FOUT instruction moves the bit pattern down through the stack, then out of the stack and into a destination table.

Warning FOUT will override any disabled coils in a destination table without enabling them. If a coil has been disabled for repair or maintenance, there is the potential for injury, since that coil's state can change as a result of the FOUT operation.

When you are running a FIFO stack in ladder logic, the FOUT instruction should be executed in each scan before the FIN instruction so that the oldest data in the stack can be cleared to the destination table before the newest data is queued into the stack. If the FIN block is executed first, an attempt to enter data into a filled stack is ignored.



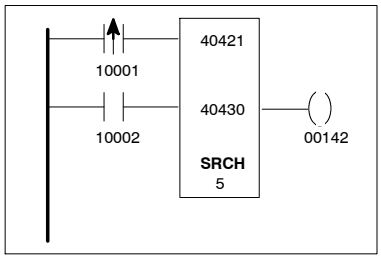
Searching a Table

The SRCH instruction allows you to search a table of registers for a specific bit pattern contained in one of the table registers. SRCH is a three-high nodal instruction.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Table search	I — 3*x* or 4*x* — O I — 4*x* — O **SRCH** K*	*Top:* ON initiates a search *Middle:* 0 = search from the beginning 1 = search from last match	*Top:* first register in the source table *Middle:* 4*x* pointer to the location in the table of the register holding the value searched for; the next register, 4*x* + 1, contains the value being searched for *Bottom:* Table length*	*Top:* echos the top input *Middle:* match found	Searches a table of registers for the bit pattern specified in the register immediately following the pointer in the middle node

* K is an integer constant in the range 1 ... 255.

An Example of a SRCH Operation



The source table to be searched is five registers long starting at holding register 40421, and the content of the table registers is as follows:

Source Table Registers		*Register Content*
40421	=	1111
40422	=	2222
40423	=	3333
40424	=	4444
40425	=	5555

The bit pattern to be searched for is 3333, which is the value that gets entered into register 40431 (the register immediately following the pointer register in the middle node).

When contact 10001 transitions from OFF to ON, the logic searches the source table for the register that contains 3333. When that value is found (in register 40423), the pointer value in register 40430 is set to 3, indicating that the third register in the source table contains the searched-for value; coil 00142 is also energized for one scan.

Moving a Block of Data

The block move (BLKM) instruction copies the entire contents of a source table of registers to a destination table in one logic scan. BLKM is a three-high nodal instruction.

Warning BLKM will override any disabled coils in a destination table without enabling them. If a coil has been disabled for repair or maintenance, there is the potential for injury, since that coil's state can change as a result of the BLKM operation.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Block move	I — 0*x*, 1*x*, * 3*x*, or 4*x* — O 0x** or 4*x* **BLKM** K***	*Top:* ON initiates a block move	*Top:* source table *Middle:* destination table *Bottom:* Table length*	*Top:* echos the top input	Copies the entire contents of one table to another table of outputs or holding registers

* If you use a 0*x* or 1*x* reference, it must be given as a multiple of 16 + 1 (1, 17, 33, etc.), and it implies the use of 16 discrete bits (1 ... 16, 17 ... 32, 33 ... 48, etc.).

** If 0*x* references are used as the destination, they cannot be programmed as coils, only as contacts referencing those coil numbers

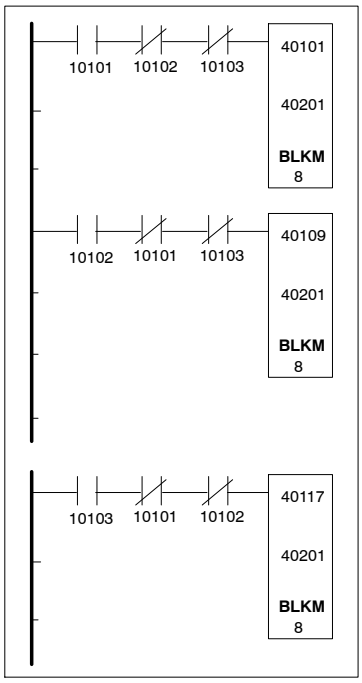
*** K is an integer constant in the range 1 ... 100.

Application Example: A Recipe Loading Routine Using Block Moves

A ladder logic program can store a collection of specific process recipes, each in a unique storage table and loadable on demand to a working table where a generic process is being run. The recipes must be structured with similar types of information in corresponding registers—if heating temperature information is kept in the third register of one recipe, similar information should be kept in the third register of all the other recipes as well.

Specific recipes can be loaded to and removed from the generic process via BLKM instructions.

The logic example shown on the next page contains an eight-register working table (registers 40201 ... 40208) in which three different recipes can be run. Recipe selection is handled by three input switches, contacts 10101, 10102, and 10103.



To run process A, for example, turn contact 10101 ON and leave contacts 10102 and 10103 OFF. When input 10101 is energized, it passes power through N.C. contacts 10102 and 10103, and the first BLKM block moves the recipe for process A from registers 40101 ... 40108 to registers 40201 ... 40208.

Chapter 7
Data Manipulation Instructions

- Boolean Logic Instructions
- An Application Example: Simple Table Averaging
- Bit Complementing in a Data Matrix
- Bit Comparison in a Data Matrix
- Sensing and Manipulating Bits in a Data Matrix

Boolean Logic Instructions

Three instructions are available to perform ANDing, ORing, and XORing logic operations.

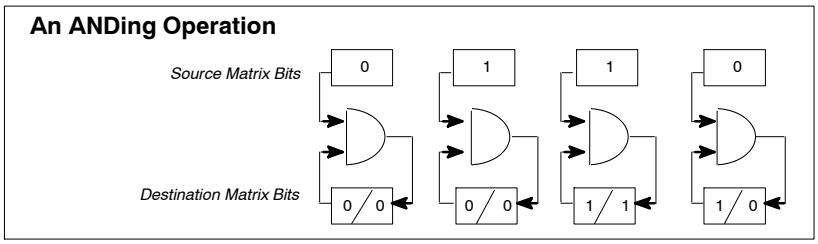
Warning These Boolean instructions will override any disabled coils in the destination matrix without enabling them. If a coil has been disabled for repair or maintenance, there is the potential for injury, since that coil's state can change as a result of the logic operation.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Boolean AND	I — 0*x*, 1*x*, * 3*x*, or 4*x* — O 0*x*** or 4*x* **AND** K****	*Top:* Initiates a logical AND operation	*Top:* source matrix *Middle:* destination matrix *Bottom:* matrix length*	*Top:* echos the top input	ANDs the bits in the source matrix with the equivalently positioned bits in the destination matrix, then places the results in the destination matrix, overwriting the original bit pattern
Boolean OR	I — 0*x*, 1*x*, * 3*x*, or 4*x* — O 0*x*** or 4*x* **OR** K***	*Top:* Initiates a logical OR operation	*Top:* source matrix *Middle:* destination matrix *Bottom:* matrix length*	*Top:* echos the top input	ORs the bits in the source matrix with the equivalently positioned bits in the destination matrix, then places the results in the destination matrix, overwriting the original bit pattern
Boolean exclusive OR	I — 0*x*, 1*x*, * 3*x*, or 4*x* — O 0*x*** or 4*x* **XOR** K***	*Top:* Initiates a logical XOR operation	*Top:* source matrix *Middle:* destination matrix *Bottom:* matrix length*	*Top:* echos the top input	XORs the bits in the source matrix with the equivalently positioned bits in the destination matrix, then places the results in the destination matrix, overwriting the original bit pattern

* If you use a 0*x* or 1*x* reference, it must be given as a multiple of 16 + 1 (1, 17, 33, etc.), and it implies the use of 16 discrete bits (1 ... 16, 17 ... 32, 33 ... 48, etc.).

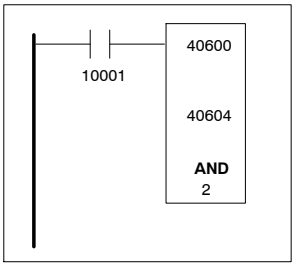
** If 0*x* references are used as the destination, they cannot be programmed as coils, only as contacts referencing those coil numbers

*** K is an integer constant in the range 1 ... 100

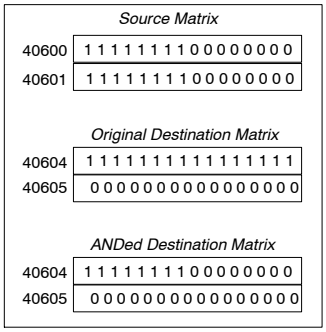


An AND instruction logically ANDs each bit in a source matrix with the corresponding bits in a destination matrix, then posts the results in the destination matrix—overwriting the previous bit pattern in the destination matrix.

For example, when contact 10001 passes power in the network below, the bit matrix comprising registers 40600 and 40601 are ANDed with the bit matrix comprising registers 40604 and 40605.



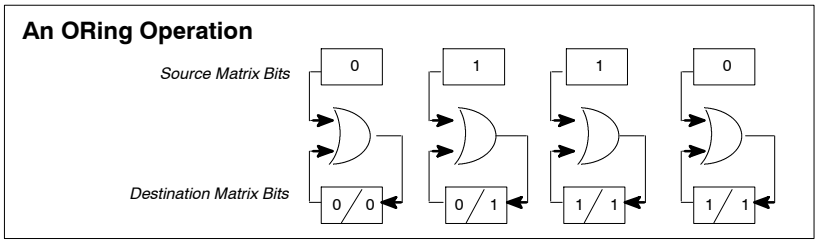
The result is then copied into registers 40604 and 40605, overwriting the previous bit pattern.



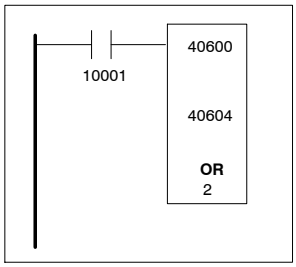
OR

Likewise, an OR instruction logically ORs the bits in a source matrix with the corresponding bits in a destination matrix, then overwrites the destination matrix with the results of the operation.

Note Outputs and coils cannot be turned OFF with the OR instruction.



For example, if we were to OR the same two matrixes as in the example shown above:



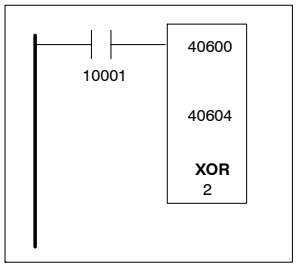
the result would be:

Source Matrix	
40600	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
40601	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
Original Destination Matrix	
40604	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
40605	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ORed Destination Matrix	
40604	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
40605	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0

XOR

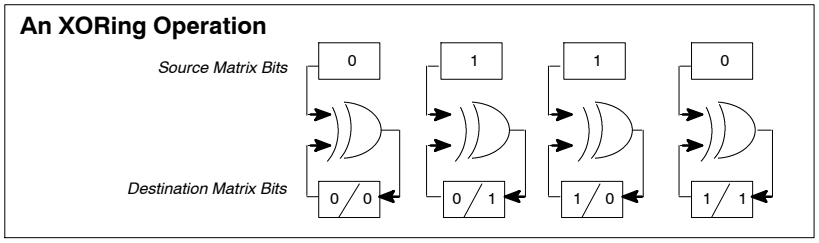
The exclusive OR instruction logically XORs the bits in a source matrix with the corresponding bits in a destination matrix, then overwrites the destination matrix with the results of the operation.

For example, if we were to XOR the same two matrixes as in the example shown above:



the result would be:

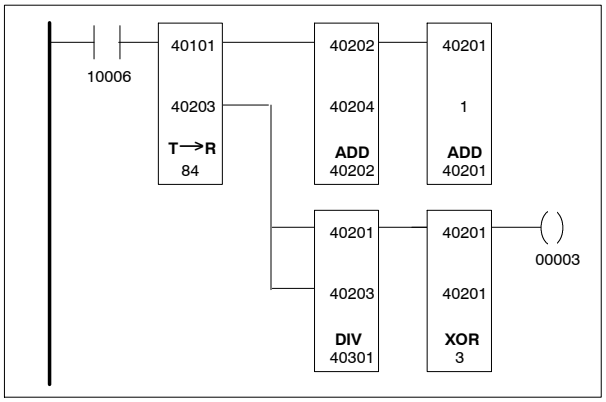
Source Matrix	
40600	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
40601	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
Original Destination Matrix	
40604	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
40605	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
XORed Destination Matrix	
40604	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
40605	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0



Archiving the Original Destination Matrix Values

If you want to save the original bit pattern from the registers in the destination matrix, use the BLKM instruction to copy the information into another table before running the Boolean logic operation.

An Application Example: Simple Table Averaging



Here is an application routine that combines three integer math calculations with a data transfer and an XOR instruction. It calculates the average value of the 84 values stored in the table of registers 40101 ... 40184.

When contact 10006 closes, the top node in the table-to-register instruction receives power, initiating the data transfer. The value in the first register of the table is copied into the middle node of the first ADD instruction, and the table pointer value increments register 40203 in the middle node of both the table-to-register instruction and the DIV instruction. Because the top output from the table-to-register instruction passes power, the first ADD block receives power and adds the value in register 40204 to the value in register 40202 (which is initially 0); then the sum of this addition overwrites the previous value in register 40202.

The routine continues to run this way until all the values in the table of 84 registers have been added together. At this point, the pointer value in the middle node of the table-to-register instruction is 84, and the middle output from that block passes power and enables the DIV instruction.

The values in registers 40201 (all 0s, representing the high-order portion of the sum of all the register values in the table) and 40202 (the low-order portion of the sum) are divided by 84. The result is placed in register 40301, and the remainder is placed in register 40302. (Because there is power to the middle input of the DIV instruction, the remainder is expressed as a decimal.) The result of the DIV operation is the average value of the current values stored in all 84 registers in the table.

When the top output from the DIV instruction passes power, the XOR instruction becomes empowered. It exclusively ORs the values in registers 40201 ... 40203 with themselves, clearing the matrix to 0s and indicating that the current table averaging operation is complete and that a new one should start.

Bit Complementing in a Data Matrix

The COMP instruction complements the bit pattern in a matrix—i.e., changes all the 0s to 1s and all the 1s to 0s—then copies the result in a second matrix. A matrix can be complemented in one scan.

COMP is a three-high nodal instruction.

Warning COMP will override any disabled coils in a destination matrix without enabling them. If a coil has been disabled for repair or maintenance, there is the potential for injury, since that coil's state can change as a result of the COMP operation.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Bit complement	I — 0*x*, 1*x*, * 3*x*, or 4*x* — O 0*x*** or 4*x* **COMP** K***	*Top:* ON initiates the bit complement operation	*Top:* source matrix *Middle:* destination matrix *Bottom:* matrix length*	*Top:* echos the top input	Complements the bit values in the source matrix and places the results in the destination matrix

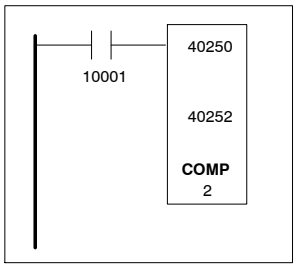
* If you use a 0*x* or 1*x* reference, it must be given as a multiple of 16 + 1 (1, 17, 33, etc.), and it implies the use of 16 discrete bits (1 ... 16, 17 ... 32, 33 ... 48, etc.).

** If 0*x* references are used as the destination, they cannot be programmed as coils, only as contacts referencing those coil numbers

*** K is an integer constant in the range 1 ... 100

A Bit Complement Example

The ladder logic below shows a COMP block with a source matrix composed of two registers—40250 and 40251—and a destination matrix composed of registers 40252 and 40253.



When contact 10001 passes power the block complements the bit values in the source register and places the results in the destination register.

Source Matrix

40250	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
40251	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0

Complemented Destination Matrix

40252	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
40253	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

All values stored in the destination register before the COMP instruction is enabled will be overwritten by the complemented source values as a result of the COMP operation.

Bit Comparison in a Data Matrix

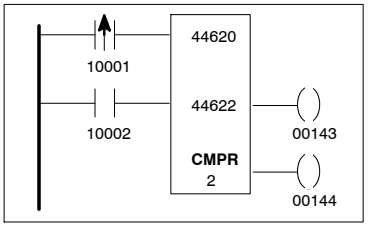
The CMPR instruction compares the bit pattern in one register matrix with the bit pattern in another matrix. When a bit value in one matrix miscompares with the correspondingly positioned bit value in the other matrix, a value indicating that matrix location is posted in the middle node.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Bit compare	I — 0*x*, 1*x*, * 3*x*, or 4*x* — O I — 4*x* — O CMPR K** — O	*Top:* ON initiates the bit compare *Middle:* 0 = restart at last miscompare 1 = restart at the beginning	*Top:* matrix a *Middle:* posts the bit position of the currently detected miscompared bit and points to matrix b, which begins at 4*x* + 1 *Bottom:* matrix length*	*Top:* echos the top input *Middle:* miscompare detected *Bottom:* state of miscompared bit in matrix a	Compares bit patterns in matrixes a and b, and reports miscompares

* If you use a 0*x* or 1*x* reference, it must be given as a multiple of 16 + 1 (1, 17, 33, etc.), and it implies the use of 16 discrete bits (1 ... 16, 17 ... 32, 33 ... 48, etc.).

** K is an integer constant in the range 1 ... 100

A Bit Comparison Example



This example shows a bit comparison between two two-register matrixes. Matrix a comprises registers 44620 and 44621; matrix b comprises registers 44623 and 44624:

	Matrix a
40600	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
40601	1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

	Matrix b
40604	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
40605	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Matrix a is compared against matrix b bit by bit on every scan that contact 10001 transitions from OFF to ON until one miscompare is found.

In the first transition of contact 10001, the matrix bits are compared until bit 17, where the value in matrix a = 1 and the value in matrix b = 0. At this point, a value of 17 is posted in register 44622, the comparison stops, and coils 00143 and 00144 energize for one scan.

If contact 10002 is energized, the function will begin to compare at matrix position 1 in the next transition of 10001 and stop again when the value in register 44622 = 17. If contact 10002 is not energized, the function will begin to compare at matrix position 18 in the next transition of 10001 and stop when the value in register 44622 = 25.

Sensing and Manipulating Bits in a Data Matrix

Three instructions are provided to let you examine and manipulate the bit patterns in a data matrix:

- ❒ The bit-sense (SENS) instruction examines and reports the sense—1 or 0—of specific bits in the matrix
- ❒ The bit-modify (MBIT) instruction modifies the sense of a specific bit in a matrix—i.e., changes a 0 bit to 1 and clears a 1 bit to 0
- ❒ The bit-rotate (BROT) instruction shifts the bit pattern in a matrix to the left or right, forcing the exiting bit to either fall out of the matrix or wrap onto the other end of the register

One bit per scan may be sensed, modified, or rotated via these instructions. Each is a three-high nodal instruction.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Bit rotation	I — 0*x*, 1*x*, * 3*x*, or 4*x* — O I — 0*x*** or 4*x* — O I — **BROT** K***	*Top:* ON initiates the bit rotation *Middle:* 0 = start left 1 = start right *Bottom:* 0 = bit falls out of the register 1 = bit wraps to start of register	*Top:* source matrix *Middle:* destination matrix *Bottom:* matrix length*	*Top:* echos the top input *Middle:* sense of the bit rotating out of the matrix	Rotates or shifts the bit pattern in a matrix, shifting the bits one position per scan
Bit sensing	I — 3*x*, 4*x*, or K_1*** — O I — 0*x*** or 4*x* — O I — **SENS** K_1** — O	*Top:* ON reports the sense of the matrix bits *Middle:* increments the pointer after a bit sense *Bottom:* resets the pointer to 1	*Top:* pointer to the matrix *Middle:* address of first register in the matrix *Bottom:* matrix length**	*Top:* echos the top input *Middle:* copies the sensed bit *Bottom:* pointer > matrix length	Examines and reports the sense of specific bits—i.e., 1 or 0—in a matrix; one bit per scan can be sensed
Bit modification	I — 3*x*, 4*x*, or K_1*** — O I — 0*x*** or 4*x* — O I — **MBIT** K_1*** — O	*Top:* ON changes the sense of the matrix bits *Middle:* 0 = clear bit 1 = set bit *Bottom:* increments the pointer after bit modification	*Top:* pointer to the matrix *Middle:* address of first register in the matrix *Bottom:* matrix length**	*Top:* echos the top input *Middle:* echos the middle input *Bottom:* pointer > matrix length	Changes the value of a bit in the matrix from 0 to 1 or from 1 to 0; one bit per scan can be modified

* If you use a 0*x* or 1*x* reference, it must be given as a multiple of 16 + 1 (1, 17, 33, etc.), and it implies the use of 16 discrete bits (1 ... 16, 17 ... 32, 33 ... 48, etc.).

** If 0*x* references are used as the destination, they cannot be programmed as coils, only as contacts referencing those coil numbers

*** K is an integer constant in the range 1 ... 100 ; K_1 is an integer constant in the range 1 ... 255

Warning MBIT and BROT will override any disabled coils in the matrix without enabling them. If a coil has been disabled for repair or maintenance, there is the potential for injury, since that coil's state can change as a result of bit manipulation.

Chapter 8
Simple ASCII Communications

- ASCII Communication via Ladder Logic
- The COMM Instruction
- Data Formats
- ASCII Character Codes
- Application Example: Using the HHP an an ASCII Display Terminal

ASCII Communication via Ladder Logic

The COMM instruction gives you the ability to read and write ASCII character devices—e.g., keyboards, displays, bar-code readers—via one of the PLC's built-in communication ports or, if the PLC is a parent, via a comm port on one of the child PLCs on the expansion link.

Canned Message Formats

The ASCII communications capability offered with the Modicon Micros PLCs provides simple canned message formats. In this way, you can use the low-cost 520VPU19200 Hand-held Programmer (HHP) as an ASCII device; the HHP itself does not support full ASCII message/format editing.

The table below shows the formats—i.e., operation types—available for use in the COMM instruction.

The difference between CR/LF and no CR/LF formats is the way in which they handle carriages and linefeeds:

- For a write operation with CR/LF, the COMM instruction automatically sends a carriage return/linefeed after the selected number of items is sent
- For a write operation with no CR/LF, the COMM instruction does not automatically send any carriage returns or linefeeds
- For a read operation with CR/LF, the format is satisfied when either the selected number of items is input—i.e., taken out of the output buffer—or when you input a carriage return or linefeed; in the second case, the CR/LF is not put into any register
- For a read operation with no CR/LF, inputting the selected number of items is the only way to satisfy the format

Canned Message Formats

Format	Decimal Format Indicator
Flush input buffer	1000
Flush input byte, no CR/LF	1001
Read ASCII character, no CR/LF	1010
Write ASCII character, no CR/LF	1110
Read ASCII character, CR/LF	1020
Write ASCII character, CR/LF	1120
Read integer (1 ... 4), no CR/LF	1031 ... 1034
Write integer (1 ... 4), no CR/LF	1131 ... 1134
Read integer (1 ... 4), CR/LF	1041 ... 1044
Write integer (1 ... 4), CR/LF	1141 ... 1144
Read hex (1 ... 4), no CR/LF	1051 ... 1054
Write hex (1 ... 4), no CR/LF	1151 ... 1154
Read hex (1 ... 4), CR/LF	1061 ... 1064
Write hex (1 ... 4), CR/LF	1161 ... 1164

The COMM Instruction

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Simple ASCII Read/Write	I — 4*x* — O 4*x* — O I — **COMM** K* — O	*Top:* ON starts the comm function *Bottom:* aborts the active function and sets the middle output	*Top:* Beginning of the control block *Middle:* Write function source or Read function destination *Bottom:* size of the source/destination table	*Top:* ACTIVE output *Middle:* turned ON for one scan when an error is detected *Bottom:* turned ON for one scan when function completes	Performs the ASCII communication function defined in the first register of the control block (register 4*x* in the top node)

*K is an integer constant in the range 1 ... 255

COMM Control Block (pointed to by the register in the top node of the instruction)

Register Number	Register Content
4*x*	Canned message format (One of the decimal format indicators from the table on the previous page)
4*x* + 1	**COMM error status**

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

No error	0 0 0 0	00
Unconfigured child selected in register 4*x* + 5	0 0 0 1	01
COMM instruction active longer than the time specified in register 4*x* + 9	0 0 1 0	02
Invalid operation type (format) selected in register 4*x*	0 0 1 1	03
Number of data fields specified in register 4*x* + 2 bigger than the constant in the bottom node of the COMM instruction	0 1 0 0	04
Receiver buffer error detected	0 1 0 1	05
Bad integer value detected in incoming or outgoing data	0 1 1 0	06
Bad hex value detected in incoming or outgoing data	0 1 1 1	07
Number of bytes to be transmitted exceeds transmit buffer size—256 bytes for the local ASCII port, 64 bytes for each child	1 0 0 0	08
No local port configured for ASCII	1 0 0 1	09
Port in use by parent/child	1 0 1 0	10
Child is unhealthy	1 0 1 1	11
DSR line is active	1 1 0 0	12

Note See the table on the next page for more details and actions to take when an error is received

4*x* + 2	number of data fields provided/expected
4*x* + 3	number of data fields processed (This register is maintained by the instruction)
4*x* + 4	reserved for Modicon use
4*x* + 5	port number (1 for a port on the local PLC, 2 ... 5 if the local PLC is a parent using a port on a child)
4*x* + 6	reserved for Modicon use
4*x* + 7	reserved for Modicon use
4*x* + 8	reserved for Modicon use
4*x* + 9	active status timer

COMM Instruction Error Codes (Returned to the second word in the Control Block)		
Code	**Error**	**Considerations**
01	Unconfigured child selected in register 4*x* + 5	The value in register 4*x* + 5 specifies which PLC the COMM instruction is to communicate with. A value of 1 selects the local PLC; a value of 2 selects child #1; a value of 3 selects child #2; etc. **Note** The physical port on the selected PLC to be used for ASCII communication—**comm1**, **comm2**, or **exp link**—is selected at configuration time. It is not dynamically selectable from the COMM instruction.
02	COMM instruction active longer than the time specified in register 4*x* + 9	
03	Invalid operation type (format) selected in register 4*x*	
04	Number of data fields specified in register 4*x* + 2 bigger than the constant in the bottom node of the COMM instruction	For ASCII formats, each register can hold two fields (ASCII characters). Thus, with a size of 1, the number of fields must be ≤ 2; with a size of 2, the number of fields must be ≤ 4; etc. For integer formats (1 ... 4), each register can hold one field (one integer) with a width of from 1 ... 4 digits. For hex formats (1 ... 4), each register can hold one field (one hex number) with a width of from 1 ... 4 nibbles. For all formats that append a return or line feed, the return/line feed does not require any register storage.
05	Receiver buffer error detected	This error can be one of parity, overrun, or framing. To clear the error, you must issue a flush buffer.
06	Bad integer value detected in incoming or outgoing data	Valid values (in decimal): For I1 0 ... 9 For I2 0 ... 99 For I3 0 ... 999 For I4 0 ... 9999
07	Bad hex value detected in incoming or outgoing data	Valid hex values: For H1 0 ... F For H2 0 ... FF For H3 0 ... FFF For H4 0 ... FFFF
08	Number of bytes to be transmitted exceeds transmit buffer size—256 bytes for the local ASCII port, 64 bytes for each child	The number of bytes to be sent depends on the format selected and the number of fields to be processed. For ASCII format, the number of bytes = the number of formats to be processed. For integer and hex formats, the number of bytes = the number of formats to be processed x the format specifier (1 ... 4). For example, if the number of fields to be processed is 2 and the format specifier is I3, the number of bytes to be sent is 6 (2 x 3). For all formats, you must add 2 to the above numbers if return/line feed is selected—returns and line feeds are stored in the TX buffer.
09	No local port configured for ASCII	Reconfigure the PLC and assign the desired port to ASCII
10	Port in use by parent/child	In a parent, this error indicates that the unit is trying to access a child's ASCII port when that port has been configured for use by the child itself. Reconfigure the child and assign the desired port to the parent. In a child, this error indicates that the unit is trying to access the local ASCII port when that port has been configured for use by the parent. Reconfigure the child and assign the desired port to the child.
11	Child is unhealthy	The parent is unable to communicate with the child over the expansion link
12	DSR line is active	When a comm port is configured for ASCII, it may actually be in Modbus/ASCII toggle mode, where the DSR line is used to toggle between the two communication protocols. When the PLC is stopped or when the DSR line is asserted while the PLC is running, the port reconfigures with Modbus parameters set in the configuration. When the PLC is running and the DSR line is not asserted while, the port reconfigures with ASCII parameters set in the configuration.

Data Formats

ASCII Character Format

Format numbers	1010, 1110, 1020, 1120
General Usage	Sending/receiving ASCII characters or 8-bit data. The data is packed two characters per 4*x* register, the first character in the most significant eight bits of the register and the second character in the eight least significant bits
Usage in a write operation	
No auto CR/LF	Format satisfied after *n* characters output from registers
Auto CR/LF	Format satisfied after *n* characters output from registers and CR/LF output
Usage in a read operation	
No auto CR/LF	Format satisfied after *n* characters input to registers
Auto CR/LF	Format satisfied after *n* characters input to registers or CR/LF received in buffer

Integer (1 ... 4) Format

Format numbers	1031 ... 1034, 1131 ... 1134, 1041 ... 1044, 1141 ... 1144
General Usage	Sending/receiving integer data fields. The data is packed as 1 ... 4 digits (depending on format number selected) per 4*x* register and is right-justified with the first digit in the data field in the leftmost position
Usage in a write operation	
No auto CR/LF	Format satisfied after *n* data fields output from registers
Auto CR/LF	Format satisfied after *n* data fields output from registers and CR/LF output
Usage in a read operation	
No auto CR/LF	Format satisfied after *n* integers input to registers
Auto CR/LF	Format satisfied after *n* integers input to registers or CR/LF received in buffer

Hex (1 ... 4) Format

Format numbers	1051 ... 1054, 1151 ... 1154, 1061 ... 1064, 1161 ... 1164
General Usage	Sending/receiving hex data fields. The data is packed as 1 ... 4 digits (depending on format number selected) per 4*x* register and is right-justified with the first digit in the data field in the leftmost position
Usage in a write operation	
No auto CR/LF	Format satisfied after *n* data fields output from registers
Auto CR/LF	Format satisfied after *n* data fields output from registers and CR/LF output
Usage in a read operation	
No auto CR/LF	Format satisfied after *n* integers input to registers
Auto CR/LF	Format satisfied after *n* integers input to registers or CR/LF received in buffer

Flush Input Buffer Format

Format number	1000
General Usage	Flushing the input buffer. In the local PLC, the buffer is flushed immediately—i.e., at logic solve time. If a parent is using the comm port of a child for the ASCII operation, the flush is done when the child receives the request from the parent—the parent will send this request at the end of scan
Usage in a write operation	Not applicable
Usage in a read operation	All bytes in the input buffer will be discarded

Flush Input Byte Format

Format number	1001
General Usage	Flushing a number of bytes from the input buffer. In the local PLC, the bytes are flushed immediately. If a parent is using the comm port of child for the ASCII operation, the flush is done when the child receives the request from the parent—the parent will send this request at the end of scan
Usage in a write operation	Not applicable
Usage in a read operation	The specified number of bytes in the input buffer will be discarded

ASCII Character Codes

Here is a list of the ASCII characters, along with their decimal and hexidecimal representations, that can be supported by the Hand-held Programmer for simple message displays:

ASCII Character	Dec Value	Hex Value
Bell	7	07
Linefeed	10	0A
Formfeed	12	0C
Carriage return	13	0D
→	26	1A
←	27	1B
Space	32	20
!	33	21
”	34	22
#	35	23
$	36	24
%	37	25
&	38	26
’	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
’	44	2C
-	45	2D
.	46	2E
/	47	2F
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B

ASCII Character	Dec Value	Hex Value
<	60	3C
=	61	3D
>	62	3E
?	63	3F
@	64	40
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D
N	78	4E
O	79	4F
P	80	50
Q	81	51
R	82	52
S	83	53
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A
[	91	5B
]	93	5D
^	94	5E
_	95	5F
a	97	61
b	98	62
c	99	63

ASCII Character	Dec Value	Hex Value
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A
{	123	7B
|	124	7C
}	125	7D
ü	129	81
ä	132	84
ö	148	94
¢	155	9B
£	156	9C
ñ	164	A4
▮	219	DB
α	224	E0
β	225	E1
Σ	228	E4
σ	229	E5
µ	230	E6
Ω	234	EA
∞	236	EC
ε	238	EE
÷	246	F6

Application Example: Using the HHP as an ASCII Display Terminal

In this example, a 520VPU19200 Handheld Programmer is used as the ASCII display terminal where a part count and cycle time are printed out. The application uses four different COMM instructions:

- The first COMM writes the ASCII message `PART COUNT =` ; it uses the 1110 format, which writes ASCII characters followed by a carriage return and linefeed (CR/LF)
- The second COMM writes four integers that indicate the part count; it uses the 1144 format, which writes four integers followed by a CR/LF
- The third COMM writes the ASCII message `CYCLE TIME =` ; it uses the 1110 format
- The fourth COMM writes four integers representing the cycle time; it uses the 1144

The first and third COMM instructions use the same control block. The first ten registers of this control block, 40400 ... 40409, look like this:

Control Block for First and Third COMMs

Register Number	Register Value	Meaning
40400	1110	Data format is: Write ASCII character, CR/LF
40401	*nn*	PLC generates an error message where *nn* is in the range 00 ... 12 (00 indicates no problems)
40402	14	A maximum of 14 bytes of information
40403	*nn*	Number of data fields processed (*nn* is maintained by the PLC)
40404		Reserved
40405	1	ASCII communication being handled from the local PLC
40406		Reserved
40407		Reserved
40408		Reserved
40409	0	No timeout

The ASCII character strings are stored in registers 40410 ... 40426 of the control block. Here is a table showing the two ASCII characters in each register and the hex equivalent for each:

Register Number	LByte ASCII	HByte ASCII	LByte Hex	HByte Hex
40410	P	A	50	41
40411	R	T	52	54
40412		C	20	43
40413	O	U	4F	55
40414	N	T	4E	54
40415		=	20	3D
40416	^	^	00	00
40420	C	Y	43	59
40421	C	L	43	4C
40422	E	^	45	00
40423	T	I	54	49
40424	M	E	4D	45
40425		=	20	3D
40426	^	^	00	00

^ indicates an empty character space

The second and fourth COMM instructions also use the same control block. The first ten registers of this control block, 40430 ... 40439, look like this:

Control Block for Second and Fourth COMMs

Register Number	Register Value	Meaning
40430	1144	Data format is: Write four integers, CR/LF
40431	*nn*	PLC generates an error message where *nn* is in the range 00 ... 12 (00 indicates no problems)
40432	1	A maximum of 1 byte of information
40433	*nn*	Number of data fields processed (*nn* is maintained by the PLC)
40434		Reserved
40435	1	ASCII communication being handled from the local PLC
40436		Reserved
40437		Reserved
40438		Reserved
40439	0	No timeout

Register 40400 should be loaded with a part count number “223”, and register 40441 should be loaded with the cycle-time value “8”.

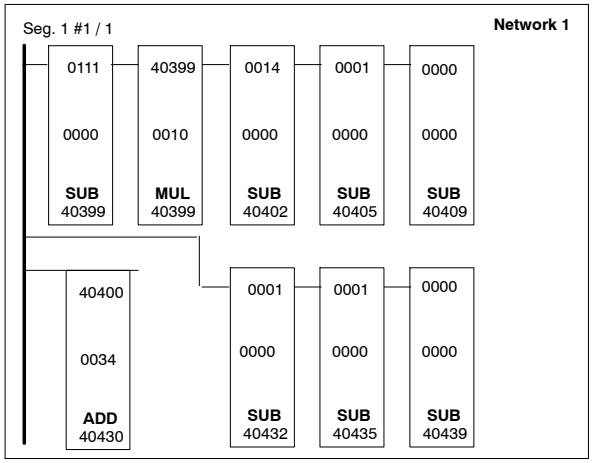
Network 1 on the following page shows a technique of using Math Function Blocks to preload registers with information – in this case, the COMM Control Blocks. These are the only registers you need to fill in other than the ASCII character registers. Another technique shown as part of this example is the shared register resources within function blocks, which saves using more registers than necessary.

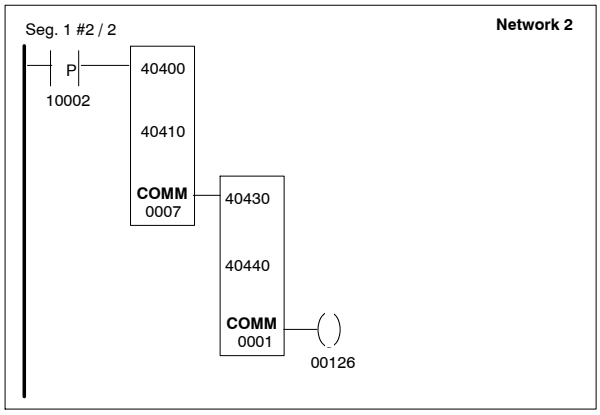
If you are using the HHP, it should be connected to (a) for 311/411 Micros, the port that was configured for ASCII after network setup, or (b) for 512/612 Micros, port 2. Follow the menu screens to set up the HHP to “Slave Mode, Simple Message.” The HHP will display a blank screen with blinking cursor.

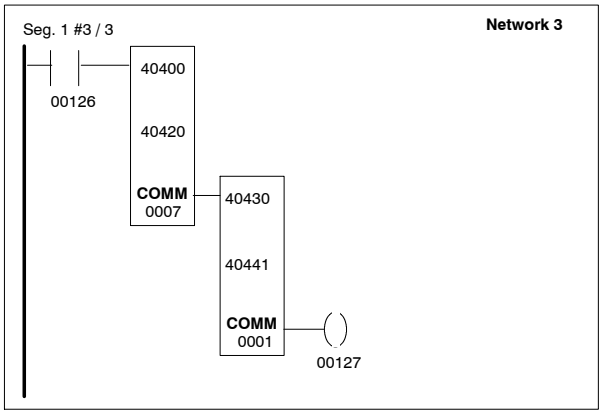
To activate this example, power should be applied to external input 10002. The result displayed on the HHP screen should be two lines of text:

```
PART COUNT = 0223
CYCLE TIME = 0008
```

To repeat, clear the HHP screen by pressing the red (‡) key and the EXIT key. Again apply power to external input 10002.


Seg. 1 #1 / 1
Network 1
0111
0000
SUB
40399
40399
0010
MUL
40399
0014
0000
SUB
40402
0001
0000
SUB
40405
0000
0000
SUB
40409
40400
0034
ADD
40430
0001
0000
SUB
40432
0001
0000
SUB
40435
0000
0000
SUB
40439


Seg. 1 #2 / 2
Network 2
P
10002
40400
40410
COMM
0007
40430
40440
COMM
0001
00126


Seg. 1 #3 / 3
Network 3
00126
40400
40420
COMM
0007
40430
40441
COMM
0001
00127

Chapter 9
The Sequence Control Interface Function

- SCIF Instruction
- Application Example: Time-stepping with SCIF Blocks

SCIF Instruction

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Sequential Control Interface	I — 4*x* — O I — 4*x* — O I — **SCIF** K* — O	*Top:* ON performs the drum or ICMP operation *Middle*: ON in drum mode increments the step pointer to the next step; ON in ICMP mode passes the compare status to the middle output *Bottom*: ON in drum mode resets the step pointer to 0; this input not used in ICMP mode	*Top:* The step pointer *Middle*: the first register in the step data table; the first six registers in the table are reserved as shown below *Bottom:* The number of application-specific step data registers in the step data table; the total number of registers in the table is K + 6	*Top:* echos the top input *Middle*: In drum mode, goes ON for the last step—i.e., when the step pointer = the maximum number of steps; ON in ICMP mode indicates a valid (1) or invalid (0) compare of the inputs (see Note below). *Bottom:* ON if an error is detected	Performs one of two functions as defined by the value in the first register in the step data table: 0 = drum mode, where the block controls outputs in the drum sequencing application 1 = input compare (ICMP) mode, where the block reads inputs to insure that limit switches, proximity switches, pushbuttons, etc. are properly positioned to allow drum outputs to be fired

*K is an integer in the range 1 ... 255.

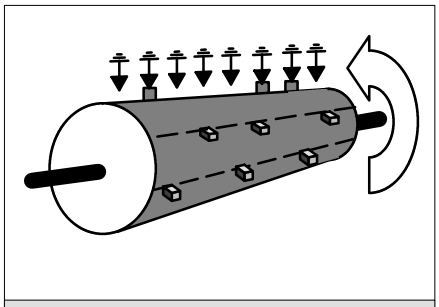
Registers in the Step Data Table (pointed to by the middle-node register)		
Reference	**Register Name**	**Description**
4*x*	*subfunction*	0 = drum mode functionality 1 = input comparison (ICMP) mode functionality (entry of any other value in this register will result in all outputs OFF)
4*x* + 1	*masked output data* (in drum mode) *raw input data* (in ICMP mode)	Loaded by SCIF each time the block is solved; the register contains the contents of the *current step data* register masked with the *output mask* register Loaded by the user from a group of sequential inputs to be used by the block in the current step
4*x* + 2	*current step data*	Loaded by SCIF each time the block is solved; the register contains data from the current step (pointed to by the step pointer)
4*x* + 3	*output mask* (in drum mode) *input mask* (in ICMP mode)	Loaded by the user before using the block, the contents will not be altered during logic solving; contains a mask to be applied to the data for each sequencer step Loaded by the user before using the block, it contains a mask to be ANDed with *raw input data* for each step—masked bits will not be compared; the masked data are put in the *masked input data* register
4*x* + 4	not used in drum mode *masked input data* (in ICMP mode)	Loaded by SCIF each time the block is solved, it contains the result of the ANDed *input mask* and *raw input data*
4*x* + 5	not used in drum mode *compare status* (in ICMP mode)	Loaded by SCIF each time the block is solved, it contains the result of an XOR of the *masked input data* and the *current step data*; unmasked inputs that are not in the correct logical state cause the associated register bit to go to 1—non-zero bits cause a miscompare and turn ON the middle output from the SCIF block
4*x* + 6	*start of data table**	First of K registers in the table containing the user-specified control data

*This and the rest of the registers represent application-specific step data in the process being controlled

Note When using the middle output, be aware that when integrating with other logic, if the step pointer is zero and the middle input is ON, then the middle output will also be ON. *This condition will cause the step pointer to be one step out of sequence.*

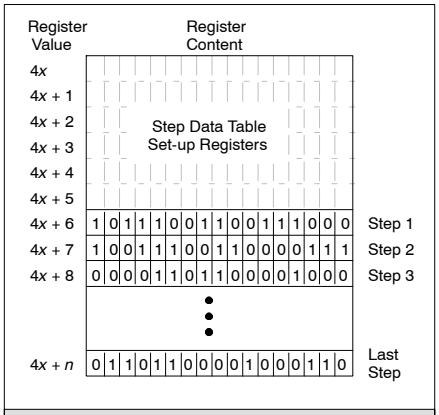
The drum and ICMP subfunctions work together to read inputs, trigger outputs, and sequence steps in the drum process. The SCIF instruction emulates electronically the mechanical tenor drum sequencer, introduced in the early 1900's and still used today in applications that require simultaneous control of multiple motors, valves, solenoids, etc. at different steps in a process.

The mechanical tenor drum works much like a piano roll. A cylinder consists of a series of rows of cams and flat surfaces. Each row represents a step in a process, and each cam represents a change of state for a mechanical device in the process. The cylinder rotates in a single direction so that each row passes a stationary string of contacts, one row at a time. As the cams in a given row meet the contacts, mechanical state changes take place for that step in the process.



A Mechanical Tenor Drum

With a SCIF block, a step data table is set up with a 16-bit register to represent each step in the process being controlled. The logic scans the table from top to bottom, treating each 1 value in a register like a cam and each 0 like a flat surface in a row on the mechanical tenor drum.



A Step Data Table for a SCIF Block

The SCIF instruction combines the concept of the mechanical tenor drum with the added power and flexibility of the Modicon Micro PLC to provide

- Reduced downtime due to the elimination of several moving parts
- Sequencing operations that can be easily programmed and maintained
- More accuracy in terms of timing between process steps
- More flexibility in setting dwell, clamp, and hold times

Modern drum sequencer applications include tire and rubber molding, injection molding, die casting, plating, bottling, and other batch-oriented uses.

SCIF combines two subfunctions—drum and ICMP. Drum mode is used to map a predefined bit pattern to the outputs on the Modicon Micro PLC in a sequential, step-by-step fashion. ICMP (input compare) mode is used to match inputs coming from the field devices with a predefined table of bit patterns for each step of the drum.

Using drum and ICMP together allows the programmer to fire outputs and compare the status of the inputs against a predefined status. If a mismatch occurs, the process is halted.

Application Example: Time-stepping with SCIF Blocks

This three-network ladder logic application example shows how SCIF blocks can be used in both drum and ICMP modes. The logic in network 1 starts and stops the sequencer cycle. Once the *Start Cycle* pushbutton is pressed, the logic cycles the drum sequencer until either the *Cycle Stop* pushbutton or *E-stop* pushbutton is pressed.

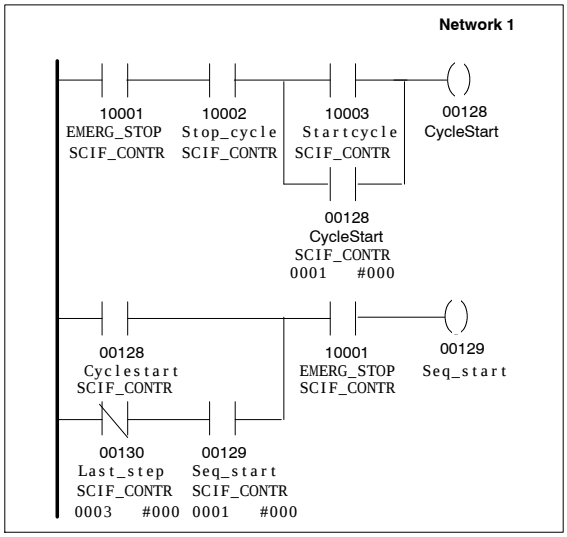
If *Cycle Stop* is requested, the drum sequence continues until the last step in the step data table has been completed. If *E-stop* is pressed, the drum sequencing stops immediately on the current step.

☞ **Note** In some applications, this *E-stop* implementation may not be desirable. If an immediate stop on the current step is not desirable in your application during an emergency shutdown, you should modify the logic to suit your specific requirements.

Network 1 controls the starting and stopping of the drum example.

Coil 00128—Cyclestart SCIF_CONTR—indicates that the SCIF cycle has started.

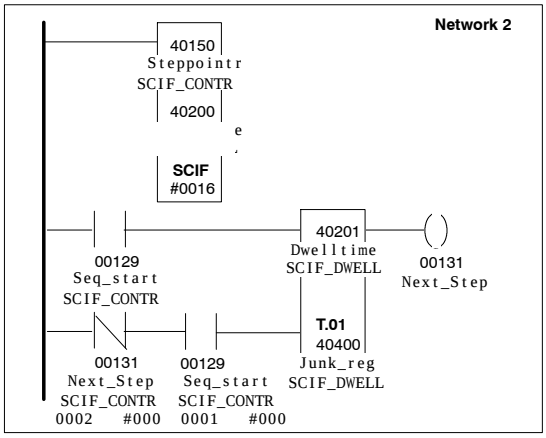
Coil 00129—Seq_start SCIF_CONTR—indicates that the SCIF sequence has started or restarted.



Caution Running this example will fire live outputs. Use this example only on a simulator, not on live machinery.

Network 2 controls the dwell time used at each step of the drum.

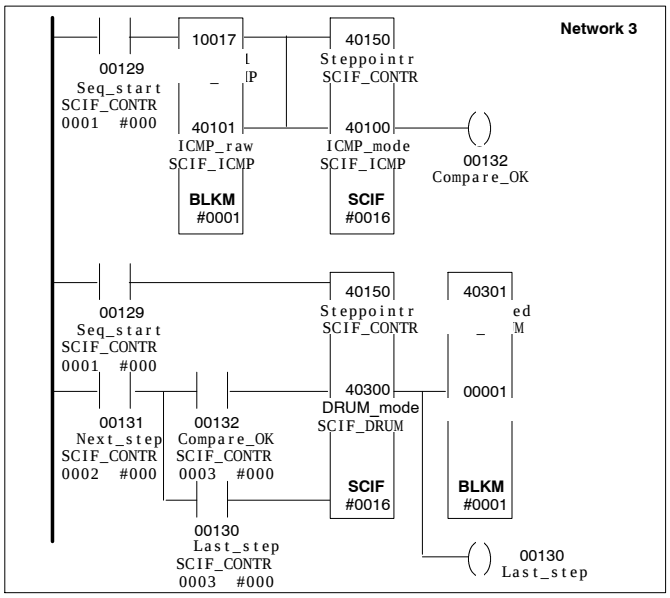
Coil 00131—`Next_step SCIF_CONTR`—increments the SCIF pointer to the next step.



Network 3 holds the ICMP and drum functions that are to be used to compare system inputs to a predetermined value and to fire the outputs of the drum.

The BLKM block in network 3 moves the feedback inputs that the ICMP-mode SCIF block next to it will monitor in its middle-node register. This SCIF block then compares the status of the feedback inputs to the expected result.

Coil 00132—`Compare_OK SCIF_CONTR`—indicates that the SCIF ICMP inputs equal the desired preset.



Network 3 performs the actual sequencer operation. As each step is executed, the value in register 40301 is changed by the drum-mode SCIF block to reflect the bit pattern of the current step.

The BLKM block takes the masked data in register 40301 and moves it into coils 00001 ... 00017. These coils could be I/O mapped directly to real outputs; however, it is also likely that contacts from these coils would be used to interlock the logic responsible for turning ON real inputs.

Reference Tags for the Application Example

The references in the table below are used to control the starting, stopping and interlocking of the SCIF function:

Control References			
Ref #	**Tag**	**Function**	**Description**
00128	Cyclestart	SCIF_CONTR	Indicates that the SCIF cycle has started
00129	Seq_start	SCIF_CONTR	Indicates SCIF sequence has started/restarted
00130	Last_step	SCIF_CONTR	Indicates SCIF at last step
00131	Next_step	SCIF_CONTR	Increments the SCIF pointer to the next step
00132	Compare_OK	SCIF_CONTR	Indicates that SCIF ICMP inputs = desired preset
10001	EMERG_STOP	SCIF_CONTR	Emergency stop halts SCIF at current step
10002	Stop_cycle	SCIF_CONTR	Cycle stop for SCIF halts SCIF at end of cycle
10003	Startcycle	SCIF_CONTR	Starts starts the SCIF cycle
40150	Steppointr	SCIF_CONTR	Step pointer register holds SCIF current step #

The references in the table below are used in the SCIF's Dwell function. When the SCIF function is used to hold step dwell times, it should be used in the drum mode = 0.

Dwell References			
Ref #	**Tag**	**Function**	**Description**
40400	Junk_reg	SCIF_DWELL	Junk register for dwell timer
40200	Dwelltable	SCIF_DWELL	SCIF used to hold dwell times for each drum step
40201	Dwelltime	SCIF_DWELL	Current dwell time for current step
40206	Dwelstep1	SCIF_DWELL	Dwell time step 1
40207	Dwelstep2	SCIF_DWELL	Dwell time step 2
40208	Dwelstep3	SCIF_DWELL	Dwell time step 3
40209	Dwelstep4	SCIF_DWELL	Dwell time step 4
40210	Dwelstep5	SCIF_DWELL	Dwell time step 5
40211	Dwelstep6	SCIF_DWELL	Dwell time step 6
40212	Dwelstep7	SCIF_DWELL	Dwell time step 7
40213	Dwelstep8	SCIF_DWELL	Dwell time step 8
40214	Dwelstep9	SCIF_DWELL	Dwell time step 9
40215	Dwelstep10	SCIF_DWELL	Dwell time step 10
40216	Dwelstep11	SCIF_DWELL	Dwell time step 11
40217	Dwelstep12	SCIF_DWELL	Dwell time step 12
40218	Dwelstep13	SCIF_DWELL	Dwell time step 13
40219	Dwelstep14	SCIF_DWELL	Dwell time step 14
40220	Dwelstep15	SCIF_DWELL	Dwell time step 15
40221	Dwelstep16	SCIF_DWELL	Dwell time step 16

The references in the table below are used by the SCIF's input compare (ICMP) function and are associated with system inputs.

ICMP Function References			
Ref #	**Tag**	**Function**	**Description**
10017	Input_1	SCIF_ICMP	1st physical input block-moved to SCIF_ICMP
10018	Input_2	SCIF_ICMP	2nd physical input block-moved to SCIF_ICMP
10019	Input_3	SCIF_ICMP	3rd physical input block-moved to SCIF_ICMP
10020	Input_4	SCIF_ICMP	4th physical input block-moved to SCIF_ICMP
10021	Input_5	SCIF_ICMP	5th physical input block-moved to SCIF_ICMP
10022	Input_6	SCIF_ICMP	6th physical input block-moved to SCIF_ICMP
10023	Input_7	SCIF_ICMP	7th physical input block-moved to SCIF_ICMP
10024	Input_8	SCIF_ICMP	8th physical input block-moved to SCIF_ICMP
10025	Input_9	SCIF_ICMP	9th physical input block-moved to SCIF_ICMP
10026	Input_10	SCIF_ICMP	10th physical input block-moved to SCIF_ICMP
10027	Input_11	SCIF_ICMP	11th physical input block-moved to SCIF_ICMP
10028	Input_12	SCIF_ICMP	12th physical input block-moved to SCIF_ICMP
10029	Input_13	SCIF_ICMP	13th physical input block-moved to SCIF_ICMP
10030	Input_14	SCIF_ICMP	14th physical input block-moved to SCIF_ICMP
10031	Input_15	SCIF_ICMP	15th physical input block-moved to SCIF_ICMP
10032	Input_16	SCIF_ICMP	16th physical input block-moved to SCIF_ICMP
40100	ICMP_mode	SCIF_ICMP	Selects SCIF mode set to 1 for ICMP
40101	ICMP_raw	SCIF_ICMP	Raw data input register for SCIF ICMP
40102	ICMP_CSD	SCIF_ICMP	Contains current step data for ICMP function
40103	ICMP_imask	SCIF_ICMP	Contains ICMP input mask
40104	ICMPmasked	SCIF_ICMP	ANDed result of raw data and ICMP masked data
40105	ICMPstatus	SCIF_ICMP	Contains XOR of masked data and ICMP step data
40106	ICMPstep1	SCIF_ICMP	1st entry in ICMP data table
40107	ICMPstep2	SCIF_ICMP	2nd entry in ICMP data table
40108	ICMPstep3	SCIF_ICMP	3rd entry in ICMP data table
40109	ICMPstep4	SCIF_ICMP	4th entry in ICMP data table
40110	ICMPstep5	SCIF_ICMP	5th entry in ICMP data table
40111	ICMPstep6	SCIF_ICMP	6th entry in ICMP data table
40112	ICMPstep7	SCIF_ICMP	7th entry in ICMP data table
40113	ICMPstep8	SCIF_ICMP	8th entry in ICMP data table
40114	ICMPstep9	SCIF_ICMP	9th entry in ICMP data table
40115	ICMPstep10	SCIF_ICMP	10th entry in ICMP data table
40116	ICMPstep11	SCIF_ICMP	11th entry in ICMP data table
40117	ICMPstep12	SCIF_ICMP	12th entry in ICMP data table
40118	ICMPstep13	SCIF_ICMP	13th entry in ICMP data table
40119	ICMPstep14	SCIF_ICMP	14th entry in ICMP data table
40120	ICMPstep15	SCIF_ICMP	15th entry in ICMP data table
40121	ICMPstep16	SCIF_ICMP	16th entry in ICMP data table

The references in the table below are used by the SCIF's drum function and are associated with system outputs.

Drum References			
Ref #	**Tag**	**Function**	**Description**
40300	DRUM_mode	SCIF_DRUM	Selects SCIF mode, set to 0 for drum
40301	DRUMmasked	SCIF_DRUM	Masked drum output = Mask AND current step data
40302	DRUM_CSD	SCIF_DRUM	Drum current step data (CSD)
40303	DRUM_omask	SCIF_DRUM	Drum output mask
40304	DRUM_R1	SCIF_DRUM	Reserved drum register 1
40305	DRUM_R2	SCIF_DRUM	Reserved drum register 2
40306	DRUMstep1	SCIF_DRUM	1st entry in drum data table
40307	DRUMstep2	SCIF_DRUM	2nd entry in drum data table
40308	DRUMstep3	SCIF_DRUM	3rd entry in drum data table
40309	DRUMstep4	SCIF_DRUM	4th entry in drum data table
40310	DRUMstep5	SCIF_DRUM	5th entry in drum data table
40311	DRUMstep6	SCIF_DRUM	6th entry in drum data table
40312	DRUMstep7	SCIF_DRUM	7th entry in drum data table
40313	DRUMstep8	SCIF_DRUM	8th entry in drum data table
40314	DRUMstep9	SCIF_DRUM	9th entry in drum data table
40315	DRUMstep10	SCIF_DRUM	10th entry in drum data table
40316	DRUMstep11	SCIF_DRUM	11th entry in drum data table
40317	DRUMstep12	SCIF_DRUM	12th entry in drum data table
40318	DRUMstep13	SCIF_DRUM	13th entry in drum data table
40319	DRUMstep14	SCIF_DRUM	14th entry in drum data table
40320	DRUMstep15	SCIF_DRUM	15th entry in drum data table
40321	DRUMstep16	SCIF_DRUM	16th entry in drum data table
00001	Output_1	SCIF_DRUM	1st physical output block-moved from SCIF_DRUM
00002	Output_2	SCIF_DRUM	2nd physical output block-moved from SCIF_DRUM
00003	Output_3	SCIF_DRUM	3rd physical output block-moved from SCIF_DRUM
00004	Output_4	SCIF_DRUM	4th physical output block-moved from SCIF_DRUM
00005	Output_5	SCIF_DRUM	5th physical output block-moved from SCIF_DRUM
00006	Output_6	SCIF_DRUM	6th physical output block-moved from SCIF_DRUM
00007	Output_7	SCIF_DRUM	7th physical output block-moved from SCIF_DRUM
00008	Output_8	SCIF_DRUM	8th physical output block-moved from SCIF_DRUM
00009	Output_9	SCIF_DRUM	9th physical output block-moved from SCIF_DRUM
00010	Output_10	SCIF_DRUM	10th physical output block-moved from SCIF_DRUM
00011	Output_11	SCIF_DRUM	11th physical output block-moved from SCIF_DRUM
00012	Output_12	SCIF_DRUM	12th physical output block-moved from SCIF_DRUM
00013	Output_13	SCIF_DRUM	13th physical output block-moved from SCIF_DRUM
00014	Output_14	SCIF_DRUM	14th physical output block-moved from SCIF_DRUM
00015	Output_15	SCIF_DRUM	15th physical output block-moved from SCIF_DRUM
00016	Output_16	SCIF_DRUM	16th physical output block-moved from SCIF_DRUM

Chapter 1 0
Subroutine Instructions

- Ladder Logic Subroutine Instructions
- The Interrupt and Counter/Timer Inputs
- The CTIF Instruction
- A CTIF Application Example

Ladder Logic Subroutine Instructions

Subroutine logic may be initiated either by the hardware interrupt or by a program-based instruction (**JSR**) in the control logic. If you are using a hardware-based interrupt to trigger the subroutine, you must configure the PLC's high-speed input circuitry to handle the interrupt(s) using an instruction called **CTIF**.

In this chapter, we will discuss the both methods of getting into and out of a subroutine.

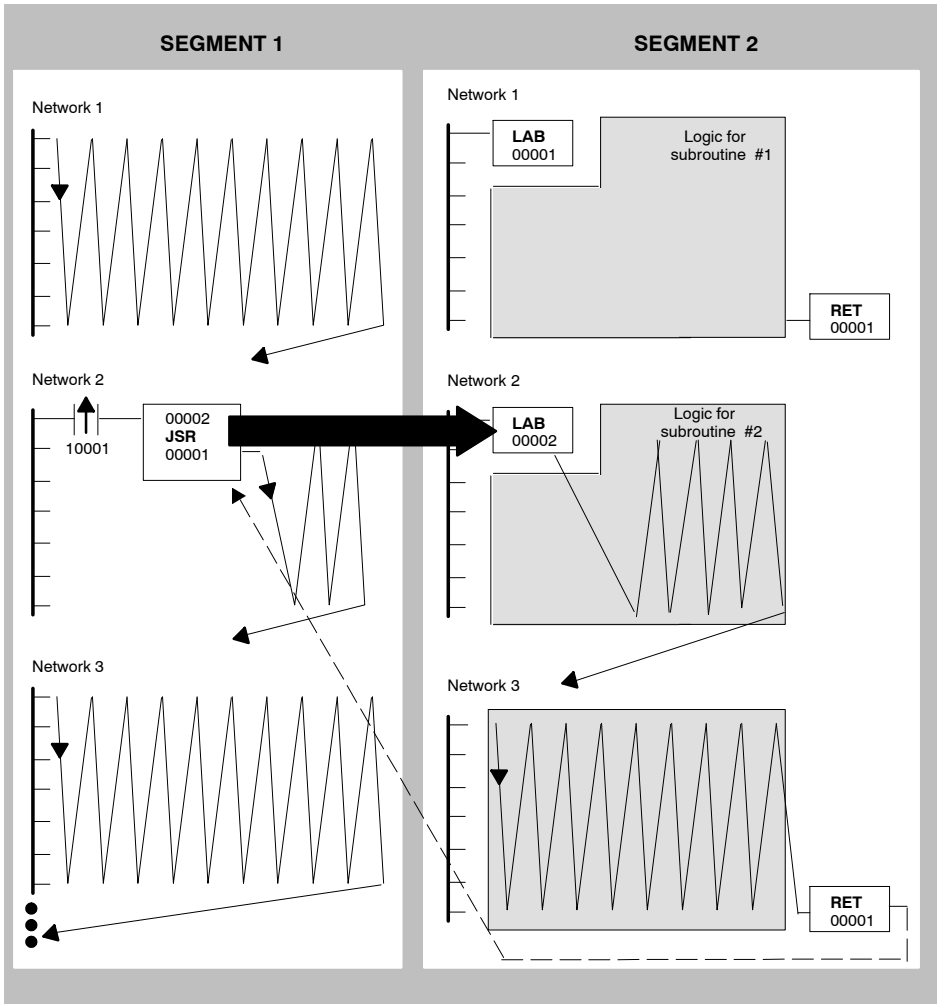
Subroutine logic is always kept in the last segment of the ladder logic program. No other logic except the subroutine logic is stored there. When a subroutine is initiated, the logic scan jumps to an instruction in the last segment called **LAB**. This instruction labels the beginning of that subroutine's logic. When the logic scan reaches an instruction in the subroutine called **RET**, it jumps out of that subroutine and returns to its previous position in the control logic.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Jump to a subroutine	I — 4*x* or K* — O JSR 00001 — O	*Top:* ON enables the source subroutine	*Top:* A constant or register value that indicates the desired subroutine *Bottom:* Always a constant value of 1	*Top:* echos the top input *Bottom:* ON if an error is detected	Causes the logic scan to jump to a specified subroutine in the last (unscheduled) segment of user logic
Label the subroutine	I — LAB K* — O	*Top:* ON activates the specified subroutine	*Top:* A unique constant value that identifies the selected subroutine	*Top:* ON if an error is detected	Marks the starting point of the subroutine in the user logic segment
Return to ladder logic	I — RET 00001 — O	*Top:* ON initiates the return out of the subfunction	*Top:* Always a constant value of 1	*Top:* ON if an error is detected	Returns the logic scan to the node immediately following the place where the subroutine was entered

*K is an integer constant in the range 1 ... 255

Below is a conceptual illustration of how a subroutine is called from ladder logic. When the logic scan in segment 1 encounters an enabled **JSR** instruction, it jumps to the indicated subroutine in segment 2. Only the logic associated with the called subroutine is scanned in segment 2—all other subroutine logic is ignored.

When the logic scan encounters a **RET** instruction in the subroutine logic, it jumps back to the node immediately following the **JSR** instruction in segment 1.



The Interrupt and Counter/Timer Inputs

The 110CPU411, 110CPU512, and 110CPU612 Models of the Micro PLC have a set of input points built into the hardware that can be configured as high-speed counters and/or hardware interrupts. These inputs are located on the left side of the input terminal block across the top of the PLC. (For specific terminal screws, refer to your PLC hardware manual.)

These inputs can be read on every scan by the PLC just like standard input points. In addition, they can used to trigger high-speed counting or hardware-initiated subroutine operations in ladder logic.

When they are used as standard inputs, they are addressed to references 10081 ... 10088 in the I/O map of the associated PLC. When they are used to trigger interrupts or high-speed counting operations, these inputs need to be configured in ladder logic via an instruction called CTIF. CTIF configures the internal high-speed interrupt and counter hardware for use with these high-speed inputs. CTIF-configured inputs do not need to be addressed in the I/O map unless their associated references are used in the ladder logic program.

Hardware Interrupt Operation

When a hardware interrupt is configured, a low-to-high transition on the input initiates an interrupt service subroutine. Interrupt-initiated subroutines are very similar to JSR-initiated subroutines. They interrupt the normal logic scan and send it to a LAB instruction in segment 2 that identifies the beginning of the appropriate subroutine. The subroutine executes until the scan encounters a RET instruction, at which point the logic scan returns to its previous location in segment 1. The primary difference is that the interrupt-initiated subroutine is triggered by an external event caused by a device hardwired to the input, while the JSR-initiated subroutine is triggered by internal conditions in the logic program.

To initiate more than one interrupt on the same input, the interrupt signal must go low then transition back to high again. The ladder logic operating system does not allow a new interrupt on the same input until the previous interrupt subroutine has been completed for about 2 ms. This delay prevents a PLC lock-up that could otherwise be caused by a continuous stream of high speed (> 2 ms) interrupts at the input.

The dedicated interrupt is connected to the CPU in the PLC through a hardware filter, which introduces approximately 50 μs of delay into the interrupt subroutine. The operating system also runs with the interrupts disabled for a certain time in each scan—about 300 μs. Thus, the initiation of the interrupt subroutine could be delayed by about 350 μs.

High-speed Input	110CPU411				110CPU512				110CPU612	
	00	01	02	03	00	01	02	03	00	03
Dedicated Interrupt	1	1	1	1	2	1	1	2	2	2
Configurable Counter/Interrupt	1	1	1	1	1	1	1	1	1	1

ll the 110CPU411, 110CPU512, and 110CPU612 PLCs have at least one input that is dedicated to interrupt signals and another input that is configurable via the CTIF instruction as either a hardware interrupt or as a high speed counter.

Interrupt User Logic Considerations

User logic to handle an immediate interrupt must be present in the last segment in the controller logic. This user logic sequence is known as the *interrupt level processing* for that particular interrupt; the user logic that was executing before the interrupt occurred is known as the *background level processing*.

When an immediate interrupt occurs, the background processing is immediately suspended and the interrupt level processing executed. Only when the interrupt level processing has finished (i.e., reached a RET label or encounters no more user logic in the controller), will the Micro return to where it was prior to the interrupt and continue executing the background level user logic.

A few of the DX functions – the COMM and the EMTH functions – have restrictions on their use in interrupt level user logic.

The COMM Dx Block

The COMM Dx block may **not** be used in an interrupt level routine.

The EMTH Dx Block

The EMTH function can **never** be interrupted while it is executing at background level if the interrupt level user logic also contains an EMTH function.

This can be done by inserting a CTIF function block just before and just after each occurrence of an EMTH function in the background level logic. The first CTIF block should be programmed to turn off interrupts, and the second CTIF block to turn them on again.

It should be emphasized that if the interrupt level user logic does not contain an EMTH function, there is no need to employ the above technique.

Block Manipulation of Registers and I/O Points

When a common block of registers must manipulated by user logic at both background and at interrupt level, be aware that the block of registers may give misleading results unless protected by temporarily turning the immediate interrupts off. This can be done by inserting a CTIF function block before and after the critical area.

The problems often encountered with block transfers can best be illustrated by the BLKM function, which fills up to a block of 100 registers. These registers may be unique sets of data to be manipulated at periodic intervals by interrupt level user logic. Some of these values may represent double precision operands, where the values may range from 0.1 through 9999.0 to be used as the divisor in a subsequent interrupt level computation.

For example, if the background user logic had previously written 0.9999 and was now about to write 1.0, then if an immediate interrupt occurred when only the least significant value (0) had been block moved, then the value which the interrupt logic would use would be 0. This value used as a divisor would cause an exception state in the processing, and would result in an error condition. This in turn could result in incorrect values being passed and processed.

The solution to this problem is to insert a CTIF function block before the BLKM Dx function, and program it to disable the external interrupt(s). Another CTIF function block should also be inserted immediately after the BLKM function and programmed to re–enable the external interrupt(s). This will then insure that the block move is uninterruptible and data integrity is maintained.

This principle applies to any group of two or more registers which are manipulated by both background and interrupt level user logic. It is up to the user to be aware of their own application and use of data in block form.

Another important issue concerns the reading or writing of banks of registers by Modbus commands when these registers are used by interrupt level logic. Remember that *user–initiated interrupt level processing can occur at **any** time* – including the time that Modbus message processing may be underway.

Solving this problem entails the use of a bank switching technique to effectively buffer the registers being manipulated by the Modbus commands. Construct a block move Dx function to move the data from a shadow register bank to or from the Modbus register block. This BLKM Dx function should be placed somewhere in the background logic with CTIF functions on either side to disable and then enable interrupts. You may then use the shadow register bank in user logic processing instead of the register bank used by Modbus communications.

Simple Interrupt Level Handlers

Interrupt handlers in user logic should be kept as small and as simple as possible for two reasons:

- To avoid “Locking–out” other user interrupts, since only one interrupt at a time can be processed and user interrupts cannot be interrupted by each other.
- To minimize the conflict due to use of common banks of registers used at both background and interrupt level.

The High Speed Counter Input

When the configurable input is set for high-speed counting, it must be configured with a *terminal count value* and it must be enabled. These conditions are set via the CTIF instruction.

The counter will count pulses on its input until the terminal count value is reached, then stop counting. You can configure the input so that the terminal count event triggers an interrupt or by addressing the terminal count and the current count in the I/O map.

The operating system runs with the interrupts disabled for a certain time in each scan—about 300 µs. Thus, the initiation of the interrupt subroutine could be delayed by about 350 µs.

To initiate another interrupt on the same terminal count, the counter must be restarted. The ladder logic operating system does not allow a new terminal count interrupt on the same input until the previous interrupt subroutine has been completed for about 2 ms. This delay prevents a PLC lock-up that could otherwise be caused by the specification of a small terminal count value with a fast input clock.

The CTIF Instruction

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Sets up the inputs for interrupt and counter/timer operations	I — 4*x* — O CTIF K* — O	*Top:* ON performs the operation specified in the top node	*Top:* First word in the CTIF parameter block *Bottom:* drop number where the operation is performed	*Top:* echos the top input *Bottom:* ON if an error is detected	Configures the hardware interrupts and counter/timer—always finishes in the same scan that it starts in

*K is an integer constant in the range 1 ... 5.

CTIF Parameter Block

Register 4*x* — Error/Operation Type

1		3		5	6	7	8	9		11	12			15	16

Error Code

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 (> 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

Operation Type

0 0	Set Mode
0 1	Get Mode

Register 4*x* + 1 — Control setup for Set Mode operation

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

Terminal-count loading:
0 Disable
1 Enable

Disable interrupt service for Int3 0 1
Enable interrupt service for Int3 1 0

Disable interrupt service for Int2 0 1
Enable interrupt service for Int2 1 0

Disable interrupt service for Int1 0 1
Enable interrupt service for Int1 1 0

0 1 Disable interrupt service for timer/counter input
1 0 Enable interrupt service for timer/counter input

0 1 Disable auto-restart operation
1 0 Enable auto-restart operation

0 1 Stop counter/timer operation
1 0 Start counter/timer operation

0 1 Counter Mode
1 0 Timer Mode

Register 4*x* + 2 — Status for Get Mode operation

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

Int3 disabled 0
Int3 enabled 1

Int2 disabled 0
Int2 enabled 1

Int1 disabled 0
Int1 enabled 1

No subroutine for timer/counter interrupt

No subroutine for Int1 interrupt

No subroutine for Int2 interrupt

No subroutine for Int3 interrupt

0 Disabled interrupt service for timer/counter input
1 Enabled interrupt service for timer/counter input

0 Disabled auto-restart operation
1 Enabled auto-restart operation

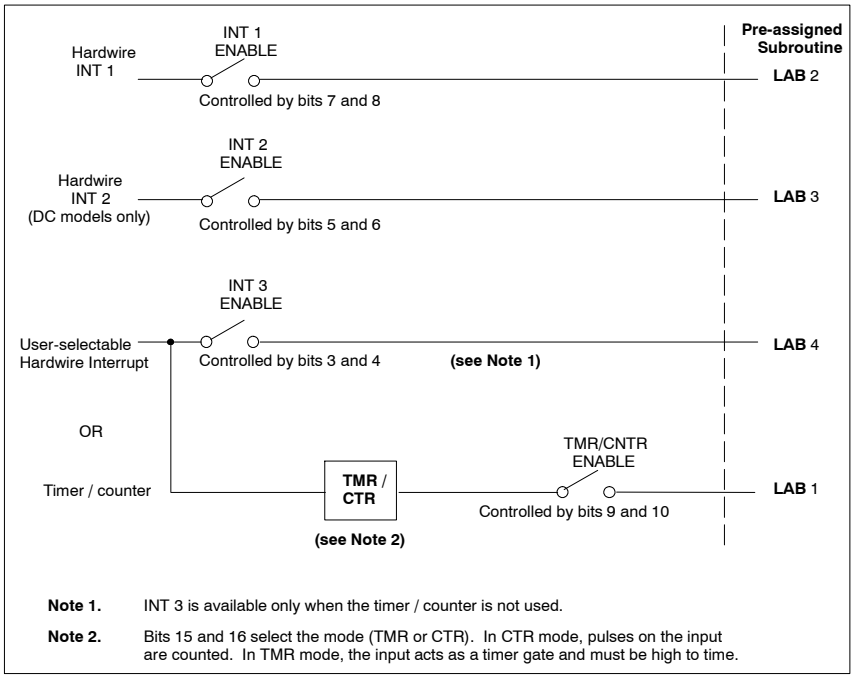
0 Stopped counter/timer operation
1 Started counter/timer operation

0 Counter Mode
1 Timer Mode

Register 4*x* + 3 — Current count value of the timer/counter input
(set by the instruction block as the current count in Get Mode;
set by the user to the counter/timer preset in Set Mode)

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 located in the PLC hardware, and the CTIF instruction is what is used to set up this hardware.

The illustrations below show how the *configuration switches* interact with the interrupt functions.



Input Type	Availability 110CPU Models	State RAM References for Interrupt Data	Subroutine Triggered by this Input
User-selectable timer/ counter interrupt	**All 411, 512, and 612 units**	10081, updated once/scan 10084, updated at the start of each subroutine	Subroutine #1
Hardwire interrupt 1	**All 411, 512, and 612 units**	10082, updated once/scan 10085, updated at the start of each subroutine	Subroutine #2
Hardwire interrupt 2	Only units that use DC power	10083, updated once/scan 10086, updated at the start of each subroutine	Subroutine #3
User-selectable interrupt 3	**All 411, 512, and 612 units**	10081, updated once/scan 10084, updated at the start of each subroutine	Subroutine #4

A CTIF Application Example

Here is a six-network demonstration program that explains how the CTIF function is configured and operated in the different available modes.

The first two networks, written in segment 1 of ladder logic, are control logic. The last four networks, written in segment 2 (the last segment) are subroutine logic that is called by the hardwired interrupts.

The example illustrates:

- High speed counter mode
- Combined high speed (1 ms) timer and interrupt mode
- A discrete, hardwired interrupt mode

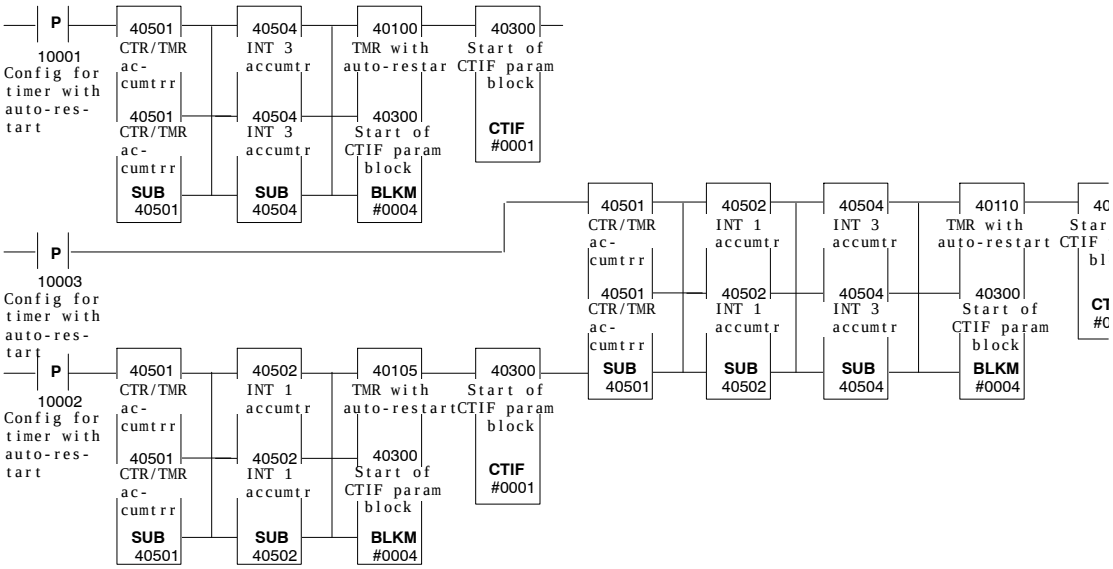
Not shown in the example is the ability to run two hardwired interrupts, a capability available only in the DC PLCs.

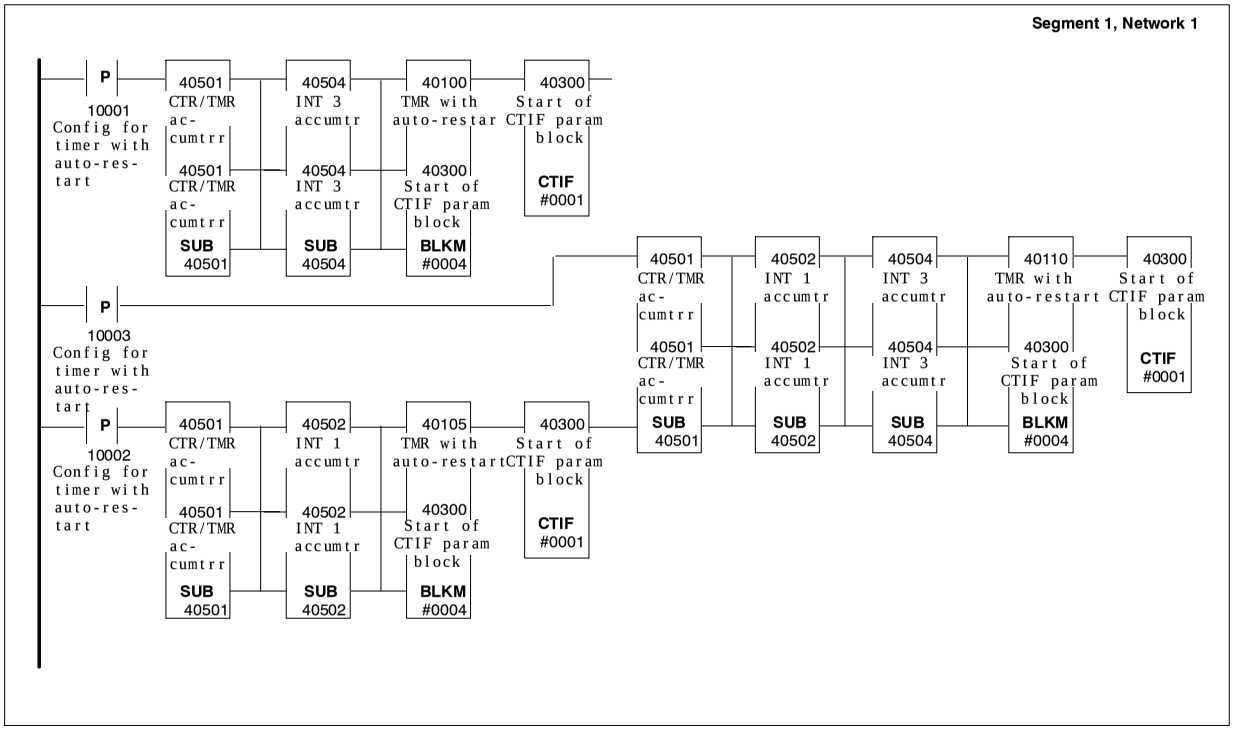
Segment 1, Network 1

Network 1 of segment 1, shown on the next page, is the first of two control networks. All counter/timer information programmed in this network has been I/O mapped to be available in input register 30001; hardwired interrupt data is available in inputs 10081 ... 10088.

When contact 10001 transitions from OFF to ON, the information in registers 40501 and 40504 is cleared. Register 40501, the accumulation register in subroutine 1 (in segment 2), increments by 1 each time it is called by the counter/timer function. Register 40504, the accumulation register in subroutine 4, increments by 1 each time the hardwired counter/timer terminal is pulsed in timer mode.

The configuration data in registers 40100 ... 40103 is moved into the CTIF parameter block (registers 40300 ... 40303). This information is immediately sent to the CTIF and is ready to run. The information sets up the parameter block as follows:


P
10001
Config for timer with auto-restart
40501
CTR/TMR ac-cumtrr
40501
CTR/TMR ac-cumtrr
SUB
40501
40504
INT 3 accumtr
40504
INT 3 accumtr
SUB
40504
40100
TMR with auto-restar
40300
Start of CTIF param block
BLKM
#0004
40300
Start of CTIF param block
CTIF
#0001
P
10003
Config for timer with auto-restart
P
10002
Config for timer with auto-restart
40501
CTR/TMR ac-cumtrr
40501
CTR/TMR ac-cumtrr
SUB
40501
40502
INT 1 accumtr
40502
INT 1 accumtr
SUB
40502
40105
TMR with auto-restart
40300
Start of CTIF param block
BLKM
#0004
40300
Start of CTIF param block
CTIF
#0001
40501
CTR/TMR ac-cumtrr
40501
CTR/TMR ac-cumtrr
SUB
40501
40502
INT 1 accumtr
40502
INT 1 accumtr
SUB
40502
40504
INT 3 accumtr
40504
INT 3 accumtr
SUB
40504
40110
TMR with auto-restart
40300
Start of CTIF param block
BLKM
#0004


Segment 1, Network 1
P
10001
Config for timer with auto-restart
40501 CTR/TMR ac-cumtrr
40501 CTR/TMR ac-cumtrr
SUB 40501
40504 INT 3 accumtr
40504 INT 3 accumtr
SUB 40504
40100 TMR with auto-restar
40300 Start of CTIF param block
BLKM #0004
40300 Start of CTIF param block
CTIF #0001
P
10003
Config for timer with auto-restart
P
10002
Config for timer with auto-restart
40501 CTR/TMR ac-cumtrr
40501 CTR/TMR ac-cumtrr
SUB 40501
40502 INT 1 accumtr
40502 INT 1 accumtr
SUB 40502
40105 TMR with auto-restart
40300 Start of CTIF param block
BLKM #0004
40300 Start of CTIF param block
CTIF #0001
40501 CTR/TMR ac-cumtrr
40501 CTR/TMR ac-cumtrr
SUB 40501
40502 INT 1 accumtr
40502 INT 1 accumtr
SUB 40502
40504 INT 3 accumtr
40504 INT 3 accumtr
SUB 40504
40110 TMR with auto-restart
40300 Start of CTIF param block
BLKM #0004
40300 Start of CTIF param block
CTIF #0001

Register	Content
40300	Error code information and mode type (always Set mode)
40301	Actual configuration information as follows: Terminal count loading enabled Interrupt service for Int 3 enabled Interrupt service for Int 2 disabled Interrupt service for Int 1 disabled Interrupt service for Int timer/counter input enabled Auto-restart operation enabled Start timer/counter operation Timer mode selected The register bit pattern is: 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 (A5AA in hex)
40302	Status information
40303	The preset value for the timer—400

The timer continues to accumulate as long as the hardwired contact remains ON. Once the timer preset is reached, subroutine 1 is called and its function is performed—i.e., 1 is added to the contents of register 40501.

Because the auto-restart option has been selected, the timer resets to 0 and begins timing once again for as long as the hardwired input is ON. The only condition under which the timer will self-reset is when it reaches its timer reset value. Interrupt 3 counts the number of OFF-to-ON transitions the input makes.

With each transition of the timer-hardwired input, subroutine 4 is called and its function is performed—i.e., 1 is added to the contents of register 40504.

When contact 10002 transitions from OFF to ON, the information in registers 40501 and 40502 is cleared. Register 40501, the accumulation register in subroutine 1, increments by 1 each time it is called by the counter/timer function.

Register 40502 is the accumulation register in subroutine 2, which increments by 1 each time the hardwired interrupt 1 input terminal is pulsed.

The configuration data in registers 40105 ... 40108 is moved into the CTIF parameter block (registers 40300 ... 40303). This information is immediately sent to the CTIF and is ready to run. The information sets up the parameter block as follows:

Register	Content
40300	Error code information and mode type (always Set mode)
40301	Actual configuration information as follows: Terminal count loading enabled Interrupt service for Int 3 disabled Interrupt service for Int 2 disabled Interrupt service for Int 1 enabled Interrupt service for Int timer/counter input enabled Auto-restart operation enabled Start timer/counter operation Counter mode selected The register bit pattern is: 1 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 (96A9 in hex)
40302	Status information
40303	The preset value for the counter—9999

The hardwired contact must transition for the counter to accumulate counts. When the counter preset is reached, subroutine 1 is called again, and it increments the contents of register 40501 by 1 each time it is called.

Because the auto-restart option has been selected, the counter resets to 0 and begins counting once again when the hardwired input transitions from OFF to ON. The only condition under which the counter will self-reset is when it reaches its counter reset value.

Each time hardwire interrupt 1 transitions from OFF to ON, subroutine 2 is called, and its function is performed—i.e., 1 is added to the contents of register 40502.

When contact 10003 transitions from OFF to ON, the information in registers 40501, 40502, and 40504 is cleared. Register 40501, the accumulation register in subroutine, increments by 1 each time it is called by the counter/timer function. Register 40501, the accumulation register in subroutine 1, increments by 1 each time it is called by the counter/timer function. Register 40504, the accumulation register in subroutine 4, increments by 1 each time the hardwired counter/timer terminal is pulsed in timer mode.

The configuration data in registers 40110 ... 40113 is moved into the CTIF parameter block (registers 40300 ... 40303). This information is immediately sent to the CTIF and is ready to run. The information sets up the parameter block as follows:

Register	Content
40300	Error code information and mode type (always Set mode)
40301	Actual configuration information as follows: Terminal count loading enabled Interrupt service for Int 3 enabled Interrupt service for Int 2 disabled Interrupt service for Int 1 enabled Interrupt service for Int timer/counter input enabled Auto-restart operation disabled Start timer/counter operation Timer mode selected The register bit pattern is: 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 (A69A in hex)
40302	Status information
40303	The preset value for the timer—400

The timer continues to accumulate as long as the hardwired contact remains ON. Once the timer preset is reached, subroutine 1 is called and its function is performed—i.e., 1 is added to the contents of register 40501.

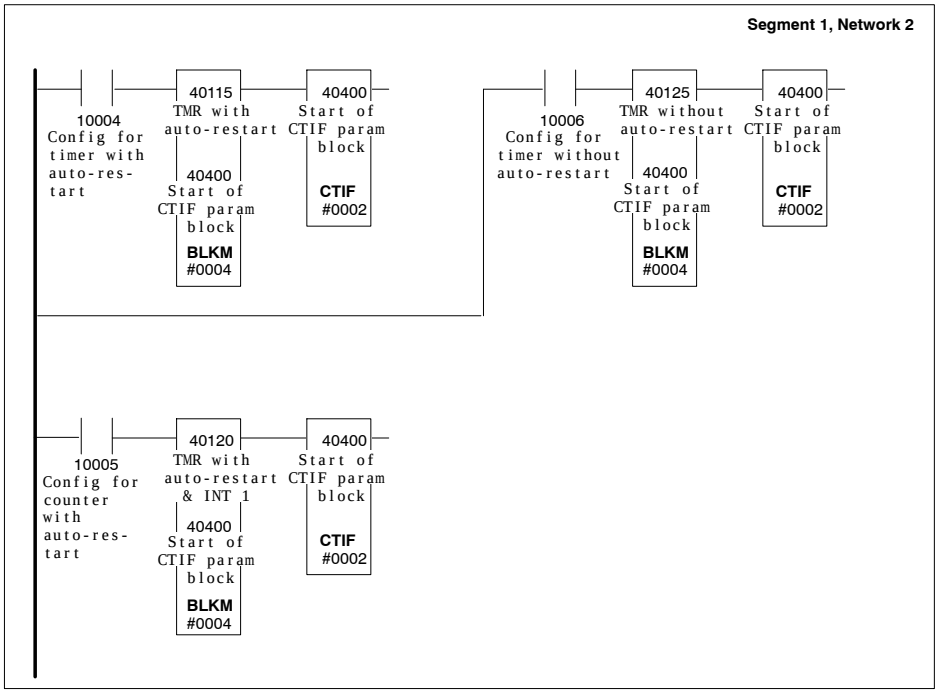
In this instance, the auto-restart option is disabled. The timer will reset to 0, but it will not begin to time until contact

10003 transitions from OFF to ON again, starting the process entire over. Interrupt 3 counts the number of OFF-to-ON transitions the input makes.

Each time hardwire interrupt 1 transitions from OFF to ON, subroutine 2 is called and its function is performed—i.e., 1 is added to the contents of register 40502.

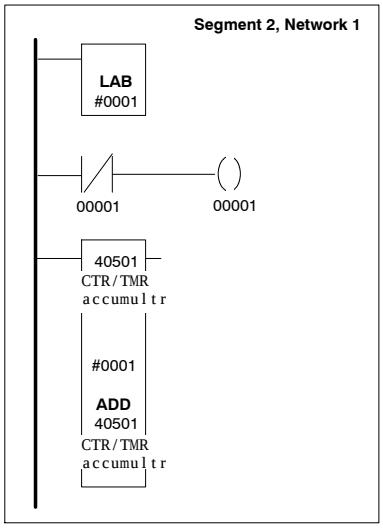
Segment 1, Network 2

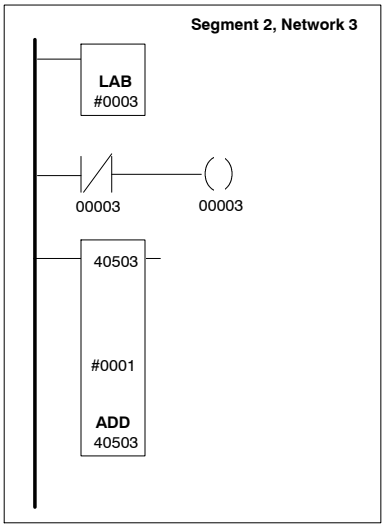
The second network in segment 1 follows the same configuration as the first. The major difference here is that network 2 is used to configure the CTIF in a child PLC. Information from that child is not readily available to the parent PLC.

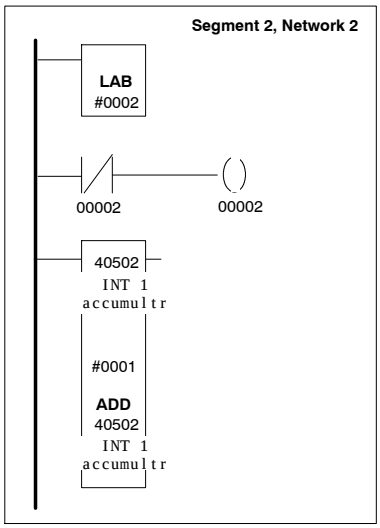


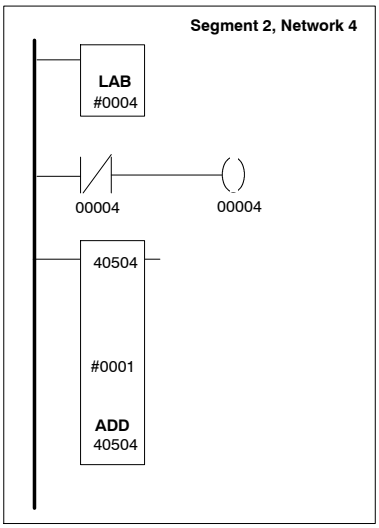
Segment 2, the Subroutines

On the following page is a series of four networks of subroutines that are called by the hardwire inputs from the previous two networks.









Chapter 11
Other Standard Instructions

- Skipping Networks
- Checking the Health Status of the PLC
- Sweep Instructions

Skipping Networks

The SKP instruction allows you to skip a specified number of networks in a ladder logic program.

When it is powered, the SKP operation is performed on every scan. The remainder of the network in which the instruction appears counts as the first of the specified number of networks to be skipped; the CPU continues to skip networks until the total number of networks skipped equals the number specified in the instruction block or until a segment boundary is reached. A SKP operation cannot cross a segment boundary.

A SKP instruction can be activated only if you specify in the PLC set-up editor that skips are allowed.

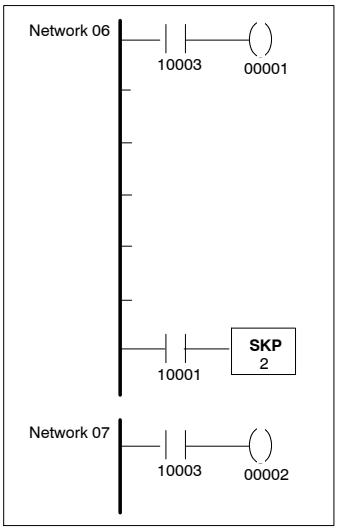
Warning If inputs and outputs that normally effect control are unintentionally skipped (or not skipped), the result can create hazardous conditions for personnel and application equipment.

SKP is a one-high nodal instruction.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Skip logic networks	I — **SKP** 3*x*, 4*x*, *or* K*	*Top:* ON activates the skip function	*Top:* Specifies the number of logic networks to be skipped		Bypasses networks of ladder logic in the program and does not solve skipped logic
*K is an integer constant in the range 1 ... 255					

A Simple SKP Example

When contact 10001 is closed, the remainder of network 06 and all of network 07 are skipped. Power flow in the skipped networks is invalid. Coil 00001 is still controlled by contact 10003 because it is solved before the SKP.



Checking the Health Status of the PLC

The Modicon Micro PLCs maintain a table in memory that contains vital system diagnostic information regarding the PLC, its I/O, and its communications. This table is 56 words long, and its contents are structured as follows:

Status Word	Content of Status Register
1 ... 11	PLC status information
12 ... 31	Health of I/O locations
32	Error codes generated at system start-up
33 ... 36	Global communications status
37 ... 40	Health of I/O communications at the local drop
41 ... 56	Health of I/O communications to and from the remote drops

Each status word is 16 bits long, and the status information is conveyed by the sense of the bits in each word. The illustrations on the following pages show how the status information is presented in the status table.

Some or all of the words in the status table can be accessed in ladder logic using the STAT instruction. The STAT block displays the bit patterns of the status words in a table of contiguous 4*x* registers, the values of which can then be seen in the panel software.

☞ **Note** Although you are allowed to specify either a 0*x* or 4*x* register in the top node, we recommend that you specify a 4*x* because of the excessive number of 0*x* registers that would be required to manage the status information.

The register you specify in the top node of the block is loaded with the current *word 1* bit values, and as many registers as you specify in the bottom node will be loaded with bit values from the corresponding words in the status table.

For example, if you are interested only in accessing PLC status information, you could specify a register address of, say, 40701 in the top node of the block and a value of 11 in the bottom node—the bit values of the first 11 words in the status table will be loaded into registers 40701 ... 40711, respectively.

If you want to load the whole status table, specify 56 in the bottom node of the instruction. If you are not using expanded I/O, you need only specify 40 in the bottom node to get all the relevant status information.

STAT is a two-high nodal instruction.

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Check CPU/ I/O Status	I — 0*x* or 4*x* — O **STAT** K*	*Top:* ON accesses the status table	*Top:* First word in the system status table *Bottom:* size of the status table	*Top:* operation completed	Gets status data from the status table in system memory and displays it in user registers
*K is an integer constant in the range 1 ... 56					

The Modicon Micro PLC Status Table

Word 1 CPU Status

If the bit is set to 1, the condition is TRUE

1		3	4			7		9		11	12			15	

- 12: Battery failed
- 11: RUN light OFF
- 9: 1 = 16-bit user logic; 0 = 24-bit user logic
- 7: Single Sweep delay enabled
- 3: Constant Sweep enabled

Word 2 PLC Drop Address

1		3	4			7		9		11	12		14	15	16

PLC is configured in single or parent mode	=	0 0 1		01
PLC is configured as child #1 on an expanded I/O network	=	0 1 0		02
PLC is configured as child #2 on an expanded I/O network	=	0 1 1		03
PLC is configured as child #3 on an expanded I/O network	=	1 0 0		04
PLC is configured as child #4 on an expanded I/O network	=	1 0 1		05

Word 3 More PLC Status

If the bit is set to 1, the condition is TRUE

1	2	3	4			7		9		11		13	14	15	16

- 3: Scan time has exceeded constant scan target
- 2: START command pending
- 1: First scan
- 13–16: Single sweeps

Word 4 Maximum number of drops allowed in an I/O network

1		3	4			7		9		11	12		14	15	16

(always set to 4) 1 0 0

The Modicon Micro PLC Status Table (continued)

Word 5 CPU Stop State Conditions

If the bit is set to 1, the condition is TRUE

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

- 16: Bad PLC setup
- 15: Coil disabled in RUN mode
- 14: Logic checksum error
- 13: Invalid node in ladder logic
- 12: Fatal error on the A120 I/O link
- 11: Mismatch between coil use table and coils in ladder logic
- 10: Real time clock error
- 9: Watchdog timer has expired
- 8: Invalid number of DOIOs/EOLs
- 7: State RAM test has failed
- 6: No SON at the start of a segment
- 5: Invalid segment scheduler
- 4: Illegal peripheral intervention
- 2: Error in the I/O map
- 1: Peripheral port stop

Word 6 Segments in Program

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

Number of segments in the current ladder logic program

Word 7 End-0f-Logic Pointer

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

Address of the EOL pointer

Word 8 is reserved

Word 9 is reserved

Word 10 RUN/LOAD/DEBUG Status

	2			5	6		8		10			13	14	15	16

DEBUG	=	0	0
RUN	=	0	1
LOAD	=	1	0

Word 11 is reserved

Words 12 ... 31 Health of I/O Units

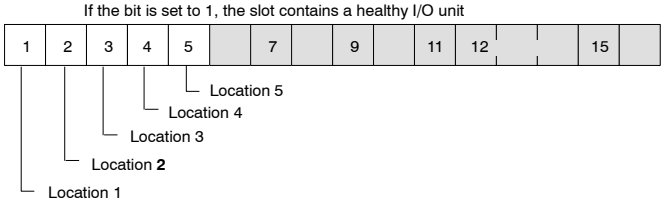
Four contiguous words are used for each of up to five Modicon Micro PLCs on an I/O expansion network; one word in each group of four is used for each possible I/O rack, assuming A120 I/O expansion:

Word		Rack
12	1	1
13		2
14		3
15		4
16	2	1
17		2
18		3
19		4
20	3	1
21		2
22		3
23		4
24	4	1
25		2
26		3
27		4
28	5	1
29		2
30		3
31		4

Rack 1 is always a Modicon Micro PLC, and racks 2 ... 4 are A120 I/O racks connected to rack 1 via an A120 I/O expansion port.

Each word contains five representative bits that show the health of the associated I/O unit in each rack—i.e., each rack can support a maximum of five I/O locations:

If the bit is set to 1, the slot contains a healthy I/O unit



With respect to A120 I/O modules, a location is the physical slot position of the module in its DTA housing. With respect to a Modicon Micro *PLC*, the location relates to the following fixed components on the unit:

- Location 1 represents the fixed discrete inputs and outputs on the unit
- Location 2 represents the dedicated interrupt component status on the unit
- Location 3 represents the user-selectable counter/timer count on the unit
- Location 4 represents any fixed analog inputs and outputs on the unit
- Location 5 represents the data transfer component on the unit for serial I/O expansion

An I/O location is healthy when it is configured and I/O mapped correctly, its personality is correct, and valid communications exist between it and the CPU that controls it.

$$\frac{\textit{word \#} - 12}{4} = \textit{quotient} + \textit{remainder}$$

where

quotient + 1 = *drop #*
remainder + 1 = *rack #*

Converting from Drop and Rack to Word #

$$\textit{word \#} = (\textit{drop \#} \times 4) + \textit{rack \#} + 7$$

Word 32 Start-up Error Codes (Always 0 when the system is running properly)

1		3	4			7		9		11	12	13	14	15	16

Bad I/O map length	0	0	0	0	0	1	01
Bad link number for child PLCs on the network	0	0	0	0	1	0	02
Wrong number of child PLCs in I/O map	0	0	0	0	1	1	03
Bad I/O map checksum	0	0	0	1	0	0	04
Bad child PLC descriptor length	0	0	1	0	1	0	10
Bad child PLC number	0	0	1	0	1	1	11
Bad holdup time for child PLC on the network	0	0	1	1	0	0	12
Bad ASCII port number	0	0	1	1	0	1	13
Bad number of slots in a child PLC	0	0	1	1	1	0	14
Child PLC has already been set up	0	0	1	1	1	1	15
comm port has already been set up	0	1	0	0	0	0	16
More than 1024 output points	0	1	0	0	0	1	17
More than 1024 input points	0	1	0	0	1	0	18
Bad slot address	0	1	0	1	0	0	20
Bad rack address	0	1	0	1	0	1	21
Bad number of output bytes	0	1	0	1	1	0	22
Bad number of input bytes	0	1	0	1	1	1	23
Bad first reference number	0	1	1	0	0	1	25
Bad second reference number	0	1	1	0	1	0	26
No input or output bytes	0	1	1	0	1	1	27
Discrete not on a 16-bit boundary	0	1	1	1	0	0	28
Unpaired odd output unit	0	1	1	1	1	0	30
Unpaired odd input unit	0	1	1	1	1	1	31
Unmatched odd input/output unit reference	1	0	0	0	0	0	32
1*x* reference after 3*x* register	1	0	0	0	0	1	33
Dummy unit reference already used	1	0	0	0	1	0	34
3*x* reference not a dummy	1	0	0	0	1	1	35
4*x* reference not a dummy	1	0	0	1	0	0	36
Dummy, then real 1*x* reference	1	0	1	0	0	0	40
Real, then dummy 1*x* reference	1	0	1	0	0	1	41
Dummy, then real 3*x* reference	1	0	1	0	1	0	42
Real, then dummy 3*x* reference	1	0	1	0	1	1	43
Too many I/O points in a drop	1	0	1	1	0	0	44
Bad unit descriptor rack	1	1	0	0	1	0	50
Bad unit descriptor slot	1	1	0	0	1	1	51
Bad unit descriptor input byte count	1	1	0	1	0	0	52
Bad unit descriptor output byte count	1	1	0	1	0	1	53
I/O driver has not been loaded	1	1	0	1	1	0	54
Unit can be used only in rack 1	1	1	0	1	1	1	55

Word 33 Global Communications

for a parent- or single-mode PLC:

1		3	4			7		9	10	11	12	13	14	15	16

0 = unsuccessful communication to any child on the I/O expansion net

Number of nonrecoverable communication losses at any PLC set-up on the I/O expansion net

for a child-mode PLC:

1		3	4			7		9	10	11	12	13	14	15	16

0 = child has not received a valid output command from the parent before holdup time has expired

Number of times the child's holdup time has expired

The Modicon Micro PLC Status Table (continued)

Word 34 Additional Global Communications

for a parent-mode PLC:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of no responses on the system								Number of retries due to a previous comm error							

for a child-mode PLC:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of ms remaining before holdup time expires															

Word 35 Additional Global Communications (for a parent-mode PLC only)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of parity errors detected on received characters								Number of framing errors detected on received characters							

Word 36 Additional Global Communications (for a parent-mode PLC only)

1	2	3	4		6		8	9	10	11	12	13	14	15	16
Last detected no response error	Last detected overrun error	Last detected framing error	Last detected parity error					Number of overrun errors detected on received characters							

Words 37 Healthy Communications in Rack 1 (for A120 expansion only)

If the bit is set to 1, the condition is TRUE

1		3	4			7		9	10	11	12	13	14	15	16
All units healthy								Number of times any local unit goes from healthy to unhealthy							

Words 38 I/O Error Detection in Rack 1 (for A120 I/O expansion only)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of times an error has been detected while communicating with I/O															

Words 39 I/O Retry Counter in Rack 1 (for A120 I/O expansion only)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of times a retry has been logged to a local I/O location															

Words 40 is reserved, all bits are 0

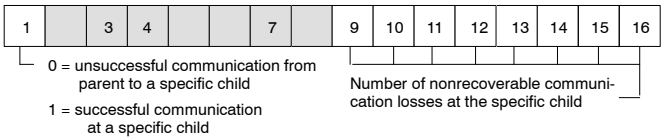
The Modicon Micro PLC Status Table (concluded)

Words 41 ... 56 are for Communications on the I/O expansion network—they have meaning only in parent units

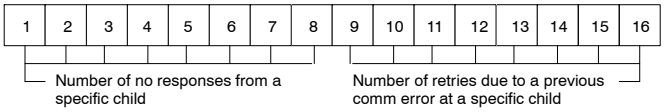
Each potential child PLC on the network is described by a group of four contiguous words:

Words 41 ... 44 apply to child #1
Words 45 ... 48 apply to child #2
Words 49 ... 52 apply to child #3
Words 53 ... 56 apply to child #4

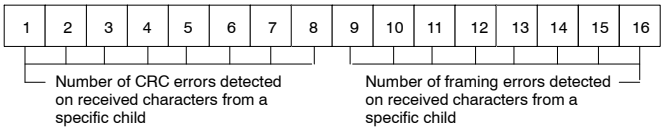
Word 41, 45, 49, 53 Format



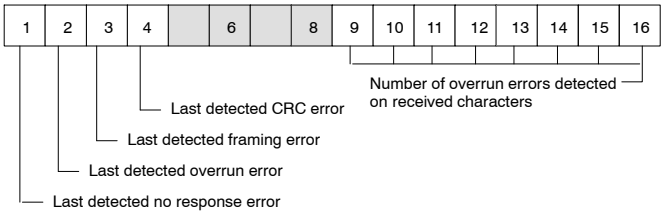
Word 42, 46, 50, 54 Format



Word 43, 47, 51, 55 Format



Word 44, 48, 52, 56 Format



Sweep Instructions

Sweep functions allow you to scan logic at fixed intervals—they do not make the controller solve logic faster or terminate scans prematurely. Sweeps may be *constant* or predetermined over some fixed number of scans—i.e., *single sweeps*.

Constant sweep allows you to target your scan times from 10 ... 200 ms (in multiples of 10 ms). A target scan time is the time that elapses between the start of one scan and the start of the next. If a constant sweep is invoked with a time lapse smaller than the actual scan time, the sweep time is ignored and the system uses its normal scan rate.

The target scan time in a constant sweep encompasses logic solve time, I/O and Modbus port servicing, and system diagnostics. If you set a constant sweep target scan at 40 ms and the actual logic solve, port servicing, and diagnostics require only 30 ms, the controller will wait for 10 ms at the end of each scan before continuing to the next.

Single sweep functions allow your controller to execute a fixed number of scans—from 1 ... 15—and then to stop solving logic but continue servicing I/O. This function is useful for diagnostic work. It allows solved logic, moved data, and completed calculations to be examined for errors.

STOP

Warning Single sweeps should not be used to debug controls on machine tools, processes, or material handling systems once they have become active. Once the specified number of scans has been solved, all the outputs are frozen in their last state; since no logic solving takes place, the controller ignores all input information. This can result in unsafe, hazardous, and destructive operation of the tools or processes connected to the controller.

Consult your programming documentation for procedures to invoke sweep instructions.

Chapter 12
Enhanced Instruction Set Available on Select Micro PLC Models

- Block↔Table Move Instructions
- The Checksum Instruction
- The Proportional-Integral-Derivative Instruction
- Extended Math Instructions

Block↔Table Move Instructions

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Block-to-table move	I — 4*x* — O I — 4*x* — O I — **BLKT** K*	*Top:* ON initiates the move *Middle:* ON freezes the pointer *Bottom:* ON resets the pointer to 0	*Top:* First register in the source block *Middle:* pointer to the first register (4*x* + 1) in the destination table *Bottom:* size of the destination table	*Top:* ON when operation is completed *Middle:* Error detected—Move not possible	Moves large quantities of 4*x* registers from a fixed source block to a destination in a table
Table-to-block move	I — 4*x* — O I — 4*x* — O I — **TBLK** K*	*Top:* ON initiates the move *Middle:* ON freezes the pointer *Bottom:* ON resets the pointer to 0	*Top:* First register in the source table *Middle:* pointer to the first register (4*x* + 1) in the destination block *Bottom:* size of the destination block	*Top:* ON when operation is completed *Middle:* Error detected—Move not possible	Moves a large number of contiguous registers in a table to a fixed-destination block

*K is an integer constant in the range 1 ... 100

The Checksum Instruction

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Checksum	I — 4*x* — O I — 4*x* — O I — **CKSM** K*	*Top:* ON calculates the source table cksm *Middle:* Used with bottom input to determine cksm type *Bottom:* Used with middle input to determine cksm type	*Top:* First register in the source table *Middle:* First of two registers containing the result and the implied register count *Bottom:* size of the source table	*Top:* ON when calculation is completed *Middle:* Error detected: register count = 0 or register count > size of the source table	Performs straight check, binary addition check, CRC-16 check, or LRC check, depending on state of the middle and bottom inputs (see table below)

*K is an integer constant in the range 1 ... 255

CKSM Input Usage

		Bottom Input
Straight check	OFF	ON
Binary addition	ON	ON
CRC-16	ON	OFF
LRC	OFF	OFF

The Proportional-Integral-Derivative Instruction

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Proportional-Integral-Deriviative	I — 4*x* — O I — 4*x* — O I — **PID2** K* — O	*Top:* 0 = Manual Mode 1 = Auto Mode *Middle:* 0 = Tracking ON 1 = Tracking OFF *Bottom:* 0 = output increases as E** increases 1 = output decreases as E** increases	*Top:* First of 21 registers in the source table *Middle:* First of 9 registers used by the block for calculations *Bottom:* constant representing the interval at which the calculation is performed in tenths of a second	*Top:* invalid parameter or loop active but not being solved *Middle:* PV ≥ low alarm limit*** *Bottom:* PV ≥ low alarm limit***	Implements an algorithm that performs the specified P, PI, or PID operation, as defined in registers *4x + 5, 4x + 6, 4x + 7, and 4x* + 8 of the source table

* K is an integer constant in the range 1 ... 255
** E is error expressed in raw analog units
*** PV is the process variable

Block Function	Source Table Register Value			
	4*x* + 5	**4*x* + 6**	**4*x* + 7**	**4*x* + 8**
P	non-zero	zero	zero	non-zero
PI	non-zero	non-zero	zero	zero
PI	non-zero	non-zero	non-zero	zero

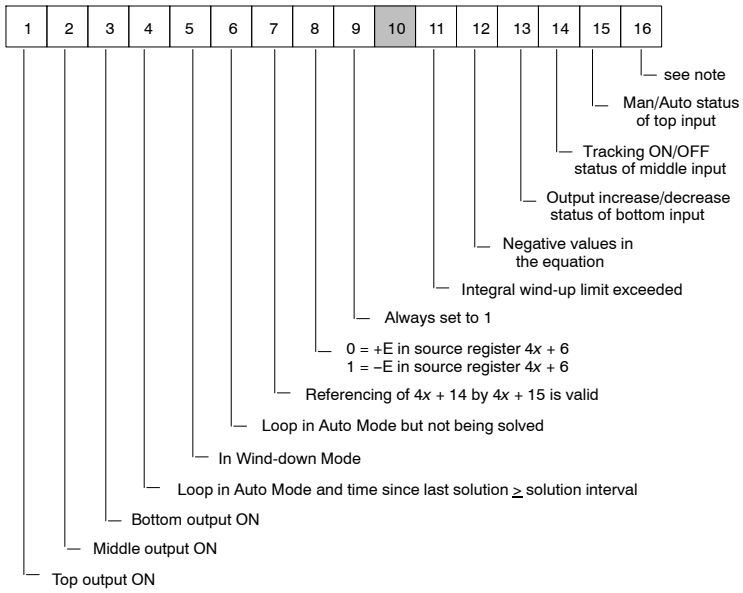
PID2 Source Table (Top Node)

Register Number	Register Content
4*x*	**Scaled PV**: loaded by the block each time it is scanned; a linear scaling is done on register 4*x* + 13 using the high and low ranges in 4*x* + 11 and 4*x* +12: $\text{scaled PV} = \frac{\text{reg } 4x + 13}{4095} \times (\text{reg } 4x + 11 - \text{reg } 4x + 12) + \text{reg } 4x + 12$ Truncate the result at the decimal point and discard all digits to the right of the decimal point—do not round off the result.
4*x* + 1	**SP**: the set point specified in engineering units; its value must be > 4*x* + 11 > 4*x* + 12
4*x* + 2	**M_v**: loaded by the block every time the loop is solved; it is clamped to the range 0 ... 4095, making the output compatible with an analog output; the manipulated variable register may be used for furhter CPU calculations such as cascaded loops
4*x* + 3	**High alarm limit**: load a value into this register to specify a high alarm for PV (at or above SP); enter the value in engineering units within the range specified in registers 4*x* + 11 and 4*x* + 12
4*x* + 4	**Low alarm limit**: load a value into this register to specify a low alarm for PV (at or below SP); enter the value in engineering units within the range specified in registers 4*x* + 11 and 4*x* + 12
4*x* + 5	**Proportional band**: load this register with the desired proportional constant in the range 5 ... 500; the smaller the number, the larger the proportional contribution; a valid number is required in this register for PID2 to operate

Proportional-Integral-Derivative Instruction (continued)

PID2 Source Table (Top Node)	
Register Number	**Register Content**
4x + 6	**Reset time constant**: load this register to add integral action to the calculation; the value is an integer constant in the range 0000 ... 9999, representing a range of 00.00 ... 99.99 repetitions per minute—values <9999 or >0000 stop the PID2 calculation; the larger the number, the larger the integral contribution
4x + 7	**Rate time constant**: load this register to add derivative action to the calculation; the value is an integer constant in the range 0000 ... 9999, representing a range of 00.00 ... 99.99 repetitions per minute—values <9999 or >0000 stop the PID2 calculation; the larger the number, the larger the derivative contribution
4x + 8	**Bias**: load this register to add a bias to the output—the value, which is added directly to M_v, must be between 0000 ... 4095
4x + 9	**High integral wind-up limit**: load this register with the upper limit of the output value (between 0 ... 4095) where the anti-reset wind-up takes place; if the specified value (normally 4095) is exceeded, the integral sum is no longer updated
4x + 10	**Low integral wind-up limit**: load this register with the lower limit of the output value (between 0 ... 4095) where the anti-reset wind-up takes place—the specified value is normally 0
4x + 11	**High engineering range**: load this register with the highest value for which the measurement device is spanned—e.g., if a resistance temperature device ranges from 0 ... 500 degrees C, the high engineering range value is 500; the high range value must be specified as a positive integer between 0001 ... 9999, corresponding to a raw analog input value of 4095
4x + 12	**Low engineering range**: load this register with the lowest value for which the measurement device is spanned; the low range value must be specified as a positive integer between 0001 ... 9998, corresponding to a raw analog input value of 0—it must be less than the value specified in register 4x + 11
4x + 13	**Raw analog measurement**: the logic program loads this register with PV; the measurement must be scaled and linear in the range 0 ... 4095
4x + 14	**Pointer to loop counter register**: the value you load in this register points to the register that counts the number of loops solved in each scan; the value entered in the register is the reference number of the register where the loop count is kept—e.g., if register 41236 keeps the count, enter the value 1236 in register 4x + 14 of the PID2 source table; the same value must be loaded to the 4x + 14 register in the source table of every PID2 block in a logic program
4x + 15	**Maximum number of loops/scan**: if register 4x = 14 contains a non-zero value, you may load a value into this register to specify the limit on the number of loops to be solved in a single scan
4x + 16	**Pointer to reset feedback input**: the value you load in this register points to the holding register that contains the feedback value (F); integration calculations rely on the F value being connected to M_v—as the PID2 output varies from 0 ... 4095, so should F vary from 0 ... 4095; the value entered in the register is the feedback register reference number—e.g., if the feedback register is 42250, enter the value 2250 in register 4x + 16 of the PID2 source table
4x + 17	**Output clamp high**: the value entered in this register determines the upper limit of M_v (normally 4095)
4x + 18	**Output clamp low**: the value entered in this register determines the lower limit of M_v (normally 0)
4x + 19	**RGL constant**: the *rate gain limit* value entered in this register determines the effective degree of derivative filtering; the range for this value is from 2 ... 30; the smaller the value, the more filtering takes place
4x + 20	**Pointer to track input**: the value entered in this register points to the holding register containing the track input (T) value; the T value is connected to the input of the integral lag whenever the auto bit and track bit are both TRUE; the value entered in this register is the track input register reference number—e.g., if the track input register is 40956, enter 0956 in register 4x + 20 in the PID2 source table

PID2 Calculation Block (Middle Node)	
Register Number	**Register Content**
4x	Loop status register



Note: Bit 16 is set after initial start-up or installation of the loop. If the bit is cleared, the following actions all take place in one scan:

The loop status register is rest

The current value in the real-time clock is stored in register 4*x* + 1 in this block

Registers 4*x* + 3, 4*x* + 4, and 4*x* + 5 in this block are set to zero

The value in source table register 4*x* + 13 is multiplied by 8 and stored in register 4*x* + 6 of this block

Register 4*x* + 7 and 4*x* + 8 in this block are cleared

PID2 Calculation Block (Middle Node)			
Register Number	**Register Content**		
4x + 1	Error (E) status		
	Bit Code	**Meaning**	**Check This Registe in the Source Table (Top Node)**
	0000	No errors, all validations OK	
	0001	Scaled SP above 9999	4*x* + 1
	0002	High alarm above 9999	4*x* + 3
	0003	Low alarm above 9999	4*x* + 4
	0004	Proportional band below 5	4*x* + 5
	0005	Proportional band above 500	4*x* + 5
	0006	Reset above 99.99 repeats/min	4*x* + 6
	0007	Rate above 99.99 min	4*x* + 7
	0008	Bias above 4095	4*x* + 8
	0009	High integral limit above 4095	4*x* + 9
	0010	Low integral limit above 4095	4*x* + 10
	0011	High engineering unit scale above 9999	4*x* + 11
	0012	Low engineering unit scale above 9999	4*x* + 12
	0013	High engineering unit scale below low engineering unit	4*x* + 11 and 4*x* + 12
	0014	Scaled SP above high engineering unit	4*x* + 1 and 4*x* + 11
	0015	Scaled SP below low engineering unit	4*x* + 1 and 4*x* + 11
	0016	Loops/scan > 9999	(4*x* + 15 = 0)
	0017	Reset feedback pointer out of range	4*x* + 16
	0018	High output clamp above 4095	4*x* + 17
	0019	Low output clamp above 4095	4*x* + 18
	0020	Low output clamp above high output clamp	4*x* + 17 and 4*x* + 18
	0021	RGL below 2	4*x* + 19
	0022	RGL above 30	4*x* + 19
	0023	Track F pointer out of range	4*x* + 20 and middle input ON
	0024	Track F pointer is zero	4*x* + 20 and middle input ON
	0025	Node locked out (short of scan time)	see note below
	0026	Loop counter pointer is zero	4*x* + 14 and 4*x* + 15
	0024	Loop counter pointer out of range	4*x* + 14 and 4*x* + 15
	Note: If lockout occurs often and all the parameters are valid, increase the maximum allowable number of loops/scan. Lockout may also occur if the counting registers in use are not cleared as required.		
4x + 2	**Loop timer register**: stores the real-time clock reading on the system clock each time the loop is solved; the difference between the current clock value and the value stored in this register is the elapsed time; if elapsed time ≥ the solution interval (10 times the value given in the bottom node of the PID2 block), the loop should be solved in the current scan		
4x + 3 *4x* + 4 *4x* + 5	Reserved for internal use		

Proportional-Integral-Derivative Instruction (concluded)

PID2 Calculation Block (Middle Node)	
Register Number	**Register Content**
*4*x + 6	P_V x 8 (filtered): stores the result of the filtered analog input (from source register 4*x* + 14) multiplied by eight; this value is useful in derivative control operations
*4*x + 7	**Absolute value of E**: contains the absolute value of SP – PV; bit 8 in register 4*x* + 1 of this block indicates the sign of E; the value in this register is updated after each loop solution
*4*x + 8	Reserved for internal use

Extended Math Instructions

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Double precision (32-bit) addition	I — 4*x* — O 4*x* — O **EMTH** 1	*Top:* ON initiates the double precision addition	*Top:* First of two contiguous registers containing operand 1—its value is in the range 0 ... 99,999,999 *Middle:* First of six registers in the block described below *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed *Middle:* an operand is invalid or out of range	Adds operand 1 (the value in the top node register block) and operand 2 (the value in the first two registers of the middle node block), then places the result in the registers 4*x* + 3 *and 4*x + *4 in* the middle node block

Middle Node Block	
Register Number	**Register Content**
4*x* and 4*x* + 1	the value of operand 2, in the range 0 ... 99,999,999
4*x* + 2	a non-zero value indicates that an overflow condition exists
4*x* + 3 and 4*x* + 4	the result of the double precision addition
4*x* + 5	not used but must be configured

Extended Math Instructions (continued)

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Double precision (32-bit) subtraction	I — 4*x* — O 4*x* — O **EMTH** 2 — O	*Top:* ON initiates the double precision subtraction	*Top:* First of two contiguous registers containing operand 1—its value is in the range 0 ... 99,999,999 *Middle:* First of six registers in the block described below *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed *Middle:* operand 1 = operand 2 *Bottom*: operand 1 < operand 2	Subtracts operand 2 (the value in the first and second registers in the middle node block) from operand 1 (the value in the top node block), then places the result in the third and fourth registers of the middle node block

Middle Node Block

Register Number	Register Content
4*x* and 4*x* + 1	the value of operand 2, in the range 0 ... 99,999,999
4*x* + 2 and 4*x* + 3	the result of the double precision subtraction
4*x* + 4	non-zero value indicates that an out-of-range condition exists
4*x* + 5	not used but must be configured

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Double precision multiplication	I — 4*x* — O 4*x* — O **EMTH** 3	*Top:* ON initiates the double precision multiplication	*Top:* First of two contiguous registers containing operand 1, whose value is in the range 0 ... 99,999,999 *Middle:* First of six registers in the block described below *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed *Middle*: an operand is out of range	Multiplies operand 1 (the value in the top node register block) by operand 2 (the value in the first two registers of the middle node block), then places the result in the third, fourth, fifth, and sixth registers of the middle node block

Middle Node Block

Register Number	Register Content
4*x* and 4*x* + 1	the value of operand 2, in the range 0 ... 99,999,999
4*x* + 2, 4*x* + 3, 4*x* + 4, and 4*x* + 5	the result of the double precision multiplication

Extended Math Instructions (continued)

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Double precision division	I — 4*x* — O I — 4*x* — O **EMTH** 4 — O	*Top:* ON initiates the double precision division *Middle*: ON = remainder is stored as a fraction OFF = remainder is stored as a whole number	*Top:* First of two contiguous registers containing operand 1—its value is in the range 0 ... 99,999,999 *Middle:* First of six registers in the block described below *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed *Middle*: an operand is out of range *Bottom*: operand 2 = 0	Divides operand 1 (the value in the top node register block) by operand 2 (the first two registers in the middle node block), then places the result in the third and fourth registers of the middle node block and the remainder in the fifth and sixth registers of the middle node block

Middle Node Block

Register Number	Register Content
4*x* and 4*x* + 1	the value of operand 2, in the range 0 ... 99,999,999
4*x* + 2 and 4*x* + 3	the result (quotient) of the double precision division
4*x* + 4 and 4*x* + 5	the remainder of the double precision division

Square root	I — 3*x* or 4*x* — O 4*x* — O **EMTH** 5	*Top:* ON initiates the √ operation	*Top:* First of two registers containing a source value in the range 0 ... 99,999,999 *Middle:* First of two registers where the result is stored in the fixed-decimal format: 1234.5600 *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed *Middle*: source value is out of range	Calculates the square root of the source value in the top node registers and stores the result in the middle node registers
Process square root	I — 3*x* or 4*x* — O 4*x* — O **EMTH** 6	*Top:* ON initiates the √ operation	*Top:* First of two registers containing a source value in the range 0 ... 99,999,999 *Middle:* First of two registers where the *linearized* result is stored *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed *Middle*: source value is out of range	Calculates the square root of the source value in the top node registers, linearizes it by multiplying it by 63.9922 (the square root of 4095), then stores the linearized result in the middle node registers Process square roots are often used in PID2 operations

Extended Math Instructions (continued)

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Logarithm	I — 3*x* or 4*x* — O 4*x* — O **EMTH** 7	*Top:* ON initiates a logarithmic operation	*Top:* First of two contiguous registers containing a source value in the range 0 ... 99,999,999 *Middle:* A holding register where the result is stored *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed *Middle*: an error has been detected or a value is out of range	Performs a base 10 logarithmic operation on the value in the source registers in the top node, then stores the result in the middle-node register
Antilogarithm	I — 3*x* or 4*x* — O 4*x* — O **EMTH** 8	*Top:* ON initiates a logarithmic operation	*Top:* A single register that contains a source value stored in the fixed decimal format 1.234 and in the range 0 ... 7.999 *Middle:* First of two contiguous registers where the result is stored *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed *Middle*: an error has been detected or a value is out of range	Performs a base 10 antilogarithmic operation on the value in the source register and stores the result in the middle-node registers in the fixed-decimal format: 12345678
Integer-to-floating point conversion	I — 4*x* — O 4*x* **EMTH** 9	*Top:* ON initiates the conversion	*Top:* First of two contiguous registers containing a double-precision integer source value *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Converts a double-precision integer value into a 32-bit floating point value and stores the result in the third and fourth registers of the middle-node block The first two registers in the block are not used*
		* **Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 9 instruction.			
Integer + floating point addition	I — 4*x* — O 4*x* **EMTH** 10	*Top:* ON initiates the addition	*Top:* First of two contiguous registers containing a double-precision integer value *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Adds the double-precision integer value in the top- node register block and the FP value in the first two registers in the middle-node block then stores the result in the third and fourth registers of the middle-node block

Extended Math Instructions (continued)

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
Integer – floating point subtraction	I — 3*x* or 4*x* — O 4*x* **EMTH** 11	*Top:* ON initiates the subtraction	*Top:* First of two contiguous registers containing a double-precision integer value *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Subtracts the FP value in the first two registers of the middle-node block from the integer value in the top-node register block then stores the result in the third and fourth registers of the middle-node block
Integer x floating point multiplication	I — 3*x* or 4*x* — O 4*x* **EMTH** 12	*Top:* ON initiates the multiplication	*Top:* First of two contiguous registers containing a double-precision integer value *Middle:* First in a block of four contiguous registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Multiplies the double-precision integer value in the top-node register block by the FP value in the first two registers of the middle-node block, then stores the product in the third and fourth registers of the middle-node block
Integer/floating point division	I — 4*x* — O 4*x* **EMTH** 13	*Top:* ON initiates the division	*Top:* First of two contiguous registers containing a double-precision integer value *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Divides the double-precision integer value in the top-node register block by the FP value in the first two registers of the middle-node block, then stores the quotient in the third and fourth registers of the middle-node block
floating point – integer subtraction	I — 4*x* — O 4*x* **EMTH** 14	*Top:* ON initiates the subtraction	*Top:* First of two contiguous registers containing a floating point value *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Subtracts the double-precision integer value in the first two registers of the middle-node block from the FP value in the top-node register block, then stores the result in the third and fourth registers of the middle-node block

Extended Math Instructions (continued)

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
floating point/ integer division	I — 3*x* or 4*x* — O 4*x* **EMTH** 15	*Top:* ON initiates the division	*Top:* First of two contiguous registers containing a floating point value *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Divides the double-precision integer value in the first two registers of the middle-node block by the FP value in the top-node register block, then stores the quotient in the third and fourth registers of the middle-node block
Integer-floating point comparison	I — 3*x* or 4*x* — O 4*x* — O **EMTH** 16 — O	*Top:* ON initiates the comparison	*Top:* First of two contiguous registers containing a double-precision integer value *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed *Middle:* used with the bottom output to indicate the value relationship *Bottom:* used with the middle output to indicate the value relationship	Compares the double-precision integer value with the floating point value (in the first two registers of the middle-node block), then indicates the relationship via the middle and bottom outputs (see table below) The third and fourth registers in the middle-node block are not used but must be configured
floating point-to-integer conversion	I — 4*x* — O 4*x* **EMTH** 17 — O	*Top:* ON initiates the conversion	*Top:* First of two contiguous registers containing a double-precision integer *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed *Bottom:* 0 = + integer value 1 = – integer value	Converts the FP value stored in the top two registers of the middle-node block into a double- precision integer value and stores the converted value in the third and fourth registers The first and second registers in the middle node are not used but must be configured*

EMTH 16 Outputs

Middle Output State	Bottom Output State	Value Relationship
ON	OFF	I > FP
OFF	ON	I < FP
ON	ON	I = FP

*** Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 17 instruction.

Extended Math Instructions (continued)

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
floating point addition	I — 4*x* — O 4*x* **EMTH** 18	*Top:* ON initiates the subtraction	*Top:* First of two contiguous registers containing FP value 1 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Adds FP value 1 (in the top-node register block) and FP value 2 (from the first two registers of the middle-node block), then stores the sum in the third and fourth registers of the middle-node block
floating point subtraction	I — 4*x* — O 4*x* **EMTH** 19	*Top:* ON initiates the multiplication	*Top:* First of two contiguous registers containing FP value 1 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Subtracts FP value 2 (stored in the first and second registers of the middle-node block) from FP value 1 (in the top-node register block), then stores the difference in the third and fourth registers of the middle-node block
floating point multiplication	I — 4*x* — O 4*x* **EMTH** 20	*Top:* ON initiates the division	*Top:* First of two contiguous registers containing FP value 1 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Multiplies FP value 1 (in the top-node register block) by FP value 2 (stored in the first and second registers of the middle-node block), then stores the product in the third and fourth registers of the middle-node block
floating point division	I — 4*x* — O 4*x* **EMTH** 21	*Top:* ON initiates the subtraction	*Top:* First of two contiguous registers containing FP value 1 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Divides FP value 1 (in the top-node register block) by FP value 2 (stored in the first and second registers of the middle-node block), then stores the quotient in the third and fourth registers of the middle-node block

Extended Math Instructions (continued)

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
floating point comparison	I — 4*x* — O 4*x* — O **EMTH** 22 — O	*Top:* ON initiates the comparison	*Top:* First of two contiguous registers containing FP value 1 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when comparison is complete *Middle:* used with the bottom output to indicate the value relationship *Bottom:* used with the middle output to indicate the value relationship	Compares FP value 1 (in the top-node register block) and FP value 2 (in the first two registers of the middle-node block), then indicates the relationship via the middle and bottom outputs (see table below) The third and fourth registers in the middle node block are not used but must be configured

EMTH 22 Outputs

Middle Output State	Bottom Output State	Value Relationship
ON	OFF	FP value 1 > FP value 2
OFF	ON	FP value 1 < FP value 2
ON	ON	FP value 1 = FP value 2

floating point square root	I — 4*x* — O 4*x* **EMTH** 23	*Top:* ON initiates the √ operation	*Top:* First of two contiguous registers containing an FP value *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Performs a square root operation on the FP value in the top-node block and stores the result in the third and fourth registers of the middle-node block. The first and second registers in the middle-node block are not used but must be configured*

* **Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 23 instruction.

floating point sign change	I — 4*x* — O 4*x* **EMTH** 24	*Top:* ON initiates the sign change operation	*Top:* First of two registers containing an FP value *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when operation is completed	Changes the sign of the FP value in the top-node register block and stores the result in the third and fourth registers of the middle-node block. The first and second registers of the middle-node block are not used
floating point π loading	I — O 4*x* **EMTH** 25	*Top:* ON loads π into the middle-register block	*Top:* Not used *Middle:* First of four registers where the FP value of pi is loaded *Bottom:* appropriate EMTH function code	*Top:* ON when loading is completed	Loads the FP value of pi into the third and fourth registers of the middle-node block; the first and second registers of the middle-node block are not used

Extended Math Instructions (continued)

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
floating point sine of an angle	I — 4*x* — O 4*x* **EMTH** 26	*Top:* ON initiates the calculation	*Top:* First of two contiguous registers containing the FP value of an angle in radians; the magnitude is < 65536.0 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Calculates in radials the sine of the floating point value in the top-node registers and stores the result in the third and fourth registers of the middle-node block. The first and second registers of the middle-node block are not used but must be configured.*
		* **Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 26 instruction.			
floating point cosine of an angle	I — 4*x* — O 4*x* **EMTH** 27	*Top:* ON initiates the calculation	*Top:* First of two contiguous registers containing the FP value of an angle in radians; the magnitude is < 65536.0 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Calculates in radians the cosine of the floating point value in the top-node registers and stores the result in the third and fourth registers of the middle-node block. The first and second registers of the middle-node block are not used but must be configured.*
		* **Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 27 instruction.			
floating point tangent of an angle	I — 4*x* — O 4*x* **EMTH** 28	*Top:* ON initiates the calculation	*Top:* First of two contiguous registers containing the FP value of an angle in radians; the magnitude is < 65536.0 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Calculates in radians the tangent of the floating point value in the top-node registers and stores the result in the third and fourth registers of the middle-node block. The first and second registers of the middle-node block are not used but must be configured.*
		* **Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 28 instruction.			

Extended Math Instructions (continued)

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
floating point arcsine of an angle	I — 4*x* — O 4*x* **EMTH** 29	*Top:* ON initiates the calculation	*Top:* First of two registers containing the FP value of the sine of an angle between $-\pi / 2 \ldots \pi / 2$ radians; the value must be in the range −1.0 ... +1.0 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Calculates in radians the arcsine of the floating point value in the top-node registers and stores the result in the third and fourth registers of the middle-node block;. The first and second registers of the middle-node block are not used but must be configured.*
		* **Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 29 instruction.			
floating point arc cosine of an angle	I — 4*x* — O 4*x* **EMTH** 30	*Top:* ON initiates the calculation	*Top:* First of two registers containing the FP value of the cosine of an angle between $0 \ldots \pi$ radians; in the range of −1.0 ... +1.0 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Calculates in radians the arc cosine of the floating point value in the top-node registers and stores the result in the third and fourth registers of the middle-node block. The first and second registers in the middle-node block are not used but must be configured*
		* **Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 30 instruction.			
floating point arctangent of an angle	I — 4*x* — O 4*x* **EMTH** 31	*Top:* ON initiates the calculation	*Top:* First of two contiguous registers containing the FP value of the tangent of an angle between $-\pi/2 \ldots \pi/2$ radians *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Calculates in radians the arctangent of the floating point value in the top-node registers and stores the result in the third and fourth registers of the middle-node block. The first and second registers of the middle-node block are not used but must be configured.*
		* **Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 31 instruction.			

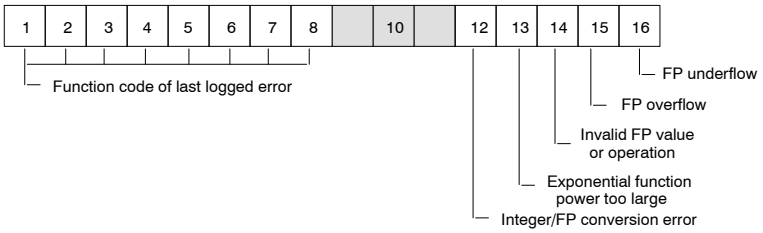
Extended Math Instructions (continued)

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
floating point radian-to-degree conversion	I — 4*x* — O 4*x* **EMTH** 32	*Top:* ON initiates the conversion	*Top:* First of two contiguous registers containing the FP value of an angle in radians *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when conversion is completed	Converts the FP value in the top-node registers to an FP representation of that value in radians, and stores the conversion in the third and fourth registers of the middle-node block. The first and second registers of the middle-node block are not used but must be configured.*
		*** Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 32 instruction.			
floating point degree-to-radian conversion	I — 4*x* — O 4*x* **EMTH** 33	*Top:* ON initiates the conversion	*Top:* First of two contiguous registers containing the FP value of an angle in degrees *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when conversion is completed	Converts the FP value in the top-node registers to an FP representation of that value in degrees, and stores the converted value in the third and fourth registers of the middle-node block. The first and second registers of the middle-node block are not used but must be configured.*
		*** Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 33 instruction.			
floating point number raised to an integer power	I — 4*x* — O 4*x* **EMTH** 34	*Top:* ON initiates the calculation	*Top:* First of two registers containing an FP value *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Raises the FP value in the top-node registers to the integer power specified in the second register of the middle-node block, and stores the result in the third and fourth registers of the middle-node block; the first register in the middle node must be set to zero
floating point exponential	I — 4*x* — O 4*x* **EMTH** 35	*Top:* ON initiates the calculation	*Top:* First of two contiguous registers containing an FP value in the range –87.34 ... +88.72 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Calculates the exponential value of the FP number in the top-node registers and stores the result in the third and fourth registers of the middle-node block. The first and second registers of the middle-node block are not used but must be configured.*
		*** Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 35 instruction.			

Extended Math Instructions (concluded)

Instruction	Structure	Inputs (I)	Nodes	Outputs (O)	Function
floating point natural logarithm	I — 4*x* — O 4*x* **EMTH** 36	*Top:* ON initiates the calculation	*Top:* First of two contiguous registers containing an FP value > 0 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Calculates the natural logarithm of the FP value in the top-node registers and stores the result in the third and fourth registers of the middle-node block The first and second registers of the middle-node block are not used but must be configured.*
		* **Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 36 instruction.			
floating point common logarithm	I — 4*x* — O 4*x* **EMTH** 37	*Top:* ON initiates the calculation	*Top:* First of two contiguous registers containing an FP value > 0 *Middle:* First in a block of four contiguous holding registers *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed	Calculates the common logarithm of the FP number in the top-node registers and stores the result in the third and fourth registers of the middle-node block. The first and second registers of the middle-node block are not used but must be configured.*
		* **Note** If you want to preserve registers, you may store the double-precision integer value in the first and second registers of the middle-node block and not configure a top-node register block in the EMTH 37 instruction.			
Error report log	I — O 4*x* — O **EMTH** 38	*Top:* ON initiates the calculation	*Top:* Not used *Middle:* First of four registers that contain the error log data (see below) *Bottom:* appropriate EMTH function code	*Top:* ON when calculation is completed *Middle:* 1 = nonzeros in the register 0 = all bits set to zero	Error data are logged in the third register of the middle-node block, and the fourth register is always set to zero The first and second registers in the middle-node block are not used, but must be configured.

Register 4*x* + 2 in the Middle Node of EMTH 38



Appendix A
Updating the Operating System in Flash

- Executive Update Utilities
- Accessing Modfax

Executive Update Utilities

The ladder logic operating system for your Modicon Micro PLC has been loaded into the PLC's Flash memory at the factory. The operating system defines the basic functionality of the PLC and its instruction set.

Updates may need to be made to increase system functionality or to fix bugs. The following information is provided in the event that you have to update the Flash.

Updating the System with a Loader Utility Program

Although Flash is nonvolatile, it can be easily changed. You can update an operating system revision through the Modbus port without any changes to hardware.

All that is required is a binary executive software file and a loader utility program. You may download the binary executive file from a personal computer to the Micro PLC utilizing the loader utility or Modsoft.

The loader utility contains five files:

- ❒ LOADER.EXE, an executable file that performs the loading function
- ❒ LOADER.HLP, a help text file
- ❒ LOADER.NDX, an index file for help screens
- ❒ MCMIII.MSG, the Modcom III error message file
- ❒ README.1ST, a file that explains exactly how to perform the update

The loader utility and the latest executive software can be obtained:

- ❒ Via the Customer Service Bulletin Board (24 hours a day, 365 days a year, at no charge)
- ❒ From your local Modicon Representative

The latest revisions of the various operating systems (or executive firmwares) are listed on the Customer Service BBS and on Modfax. The Modfax document number for latest Micro PLC upgrades is 3727. Details on accessing Modfax and the Customer Service BBS are provided in this appendix.

Determining the Latest Available Revision

There are two ways to determine what revision level of the operating system is currently installed in a PLC.

- ❒ If you have MODSOFT Lite, check the Exec ID line on the Controller Status Screen—e.g., if the Controller Status Information Screen displays:

```
EXEC ID      0861   REV   0101
```

then, your operating system is at revision 1.01.

- ❒ If you do not have MODSOFT or MODSOFT Lite, call up the following absolute memory locations to display the controller executive revision:

Controller	Page	Location
All Micro PLCs	F	4020 (hex)

Access to the above location depends on the software you are using. Please contact your software vendor for details.

Accessing Modfax

Modfax is an automatic document retrieval system available to Modicon customers. The system is self-prompting. To access Modfax, call (800) 468–5342 and select option 3. Have your FAX number available when you call.

For additional hardware or software technical assistance, call the Modicon Field Support Center at (800) 468–5342 or (508) 794–0800 (outside U.S. and Canada) and select option 1.

Accessing the Customer Service Bulletin Board

The Modicon Customer Service BBS provides several features and benefits. For more information, request Modfax Document #1113 or contact the Modicon Field Support Center.

BBS members may use the procedure given below or may proceed directly to the `Flash Lib`. Downloading Flash executives does not cost any credits. Non–BBS members should use the following procedure to retrieve a binary executive file and the loader utility from the BBS:

Executive Update Procedure

Step 1. Using your modem and communication package, dial 508–975–9779. Dial at your modem's maximum baud—we support up to 14,400 baud, no parity, 8 data and 1 stop.

Step 2. If it is your first time calling, you will need to create an account—to do this, answer the five questions you will be asked at this time.

Step 3. When you reach the main menu, select `m` and push < **enter** > . You will be welcomed to the Flash download service.

Step 4. The menu shows a number of PLC models. Select the number corresponding to the model you have.

Step 5. You will get a list of files num bered 1 ... 8, with a description of each file on the right of the screen. Select the num ber with the latest revision of your PLC—usually 1 or 2.

Step 6. Select the download protocols that matches your communication package protocol. If you have ZMODEM, use it—otherwise, try KERMIT or XMODEM.

Step 7. If your package has ZMODEM, the download commences automatically. With the other protocols, you may need to tell your communications software that you wish to download a file, then select the protocol to match the one previously selected on the BBS.

Step 8. You should now have the appropriate file in your download path (determined by the communications package).

Step 9. Let's assume that the first file you take is the binary executive file. You now want to get the loader utility. Push < **enter** > once—this should take you back to the main menu. If not, type `/GO EXEC` and push < **enter** >.

Step 10. To get the loader utility, repeat the above procedure starting at **step 5**, this time using the letter `L`.

Step 11. The downloaded files are compressed and will self-extract when executed.

The result of executing a particular downloaded .exe executive file is an executive binary file.

Step 12. Follow the instructions given in the README.1ST file to update the ladder logic operating system.

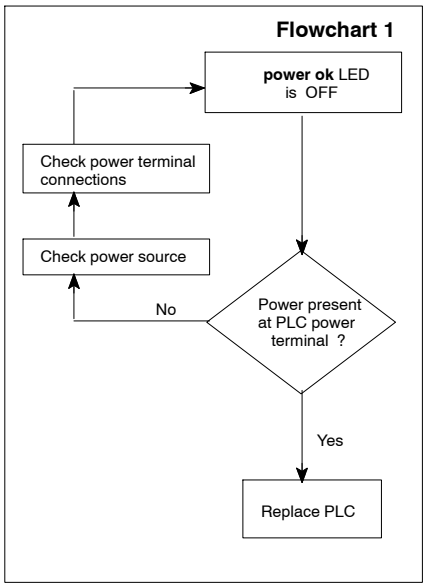
Appendix B
Troubleshooting

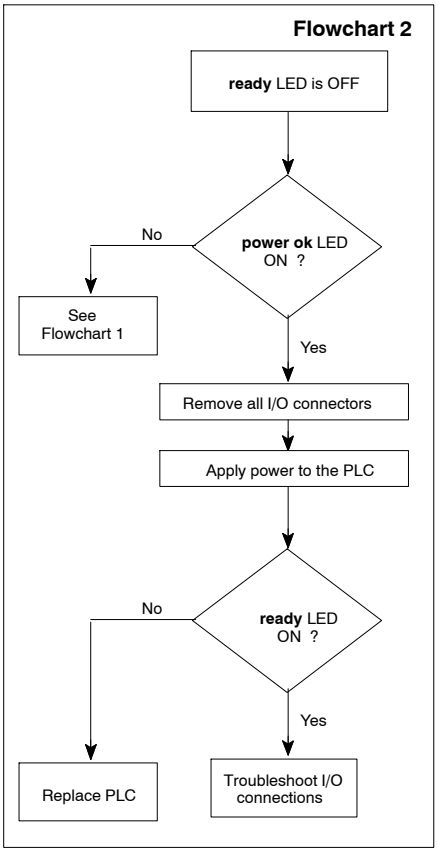
- Diagnosing Start-up Conditions
- PLC Stopped Error Codes
- PLC Crash Codes Displayed on the LEDs

Diagnosing Start-up Conditions

Symptom: No *power ok* LED

The green **power ok** LED on the Micro PLC goes ON when the internal power conditions of the PLC are healthy and receiving power from an external supply. If this LED fails to go ON after power has been applied to the PLC refer to flowchart 1.





Symptom: No *ready* LED

The amber **ready** LED goes ON once the PLC has successfully passed its power-up diagnostics, and it remains ON as long as the PLC has power and is healthy. If this LED fails to go ON after power-up, refer to flowchart 2.

Symptom: *run* LED Not ON or Flashing

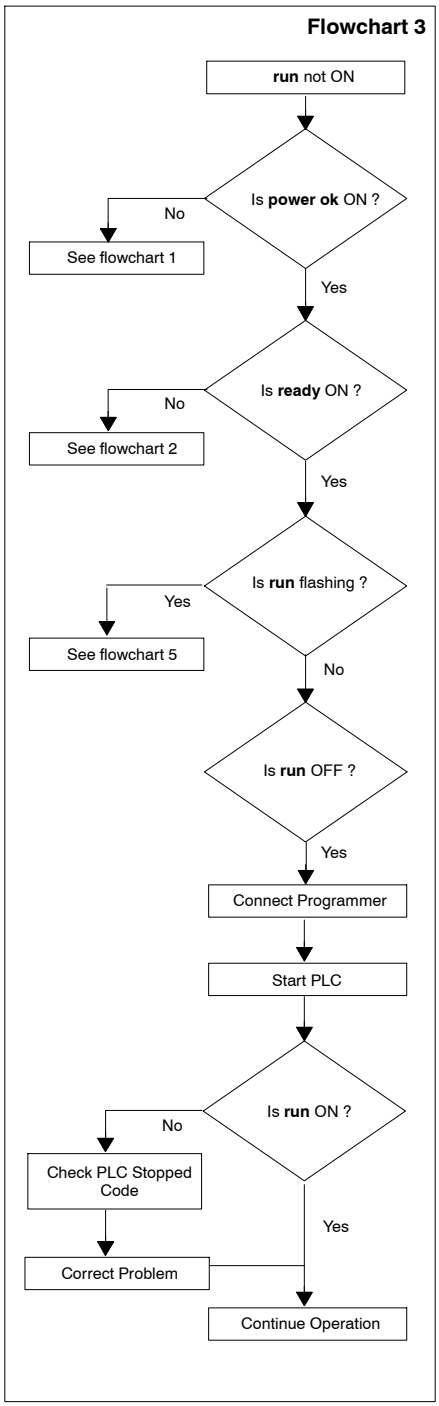
The **run** LED on the PLC goes ON steadily when the PLC has been started and is scanning ladder logic. It flashes when the PLC has power but cannot find a valid configuration. If this LED is OFF or is behaving unexpectedly, refer to flowchart 3 on the next page.

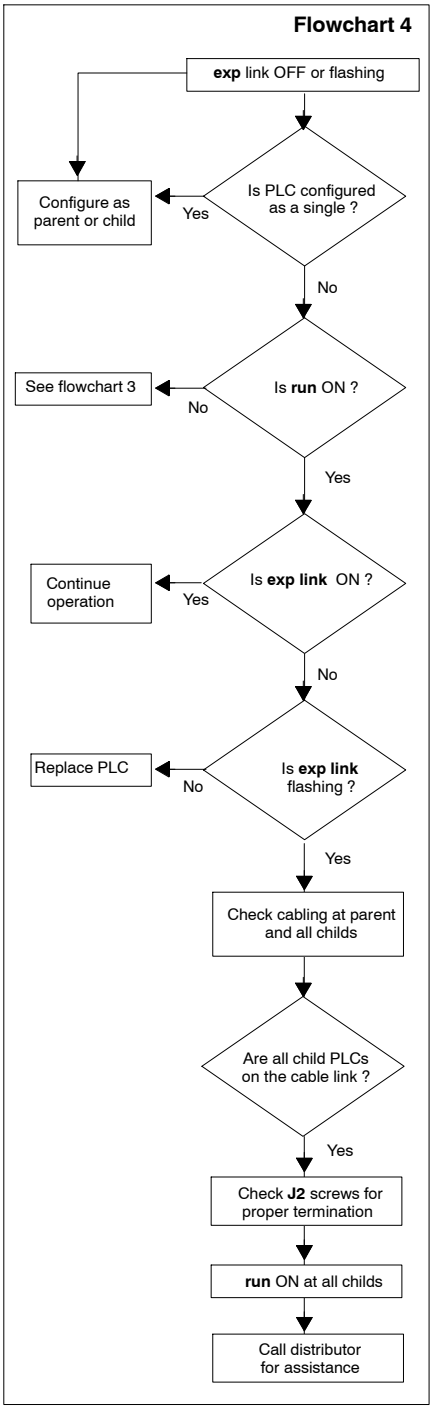
Symptom: *exp link* LED OFF or Flashing

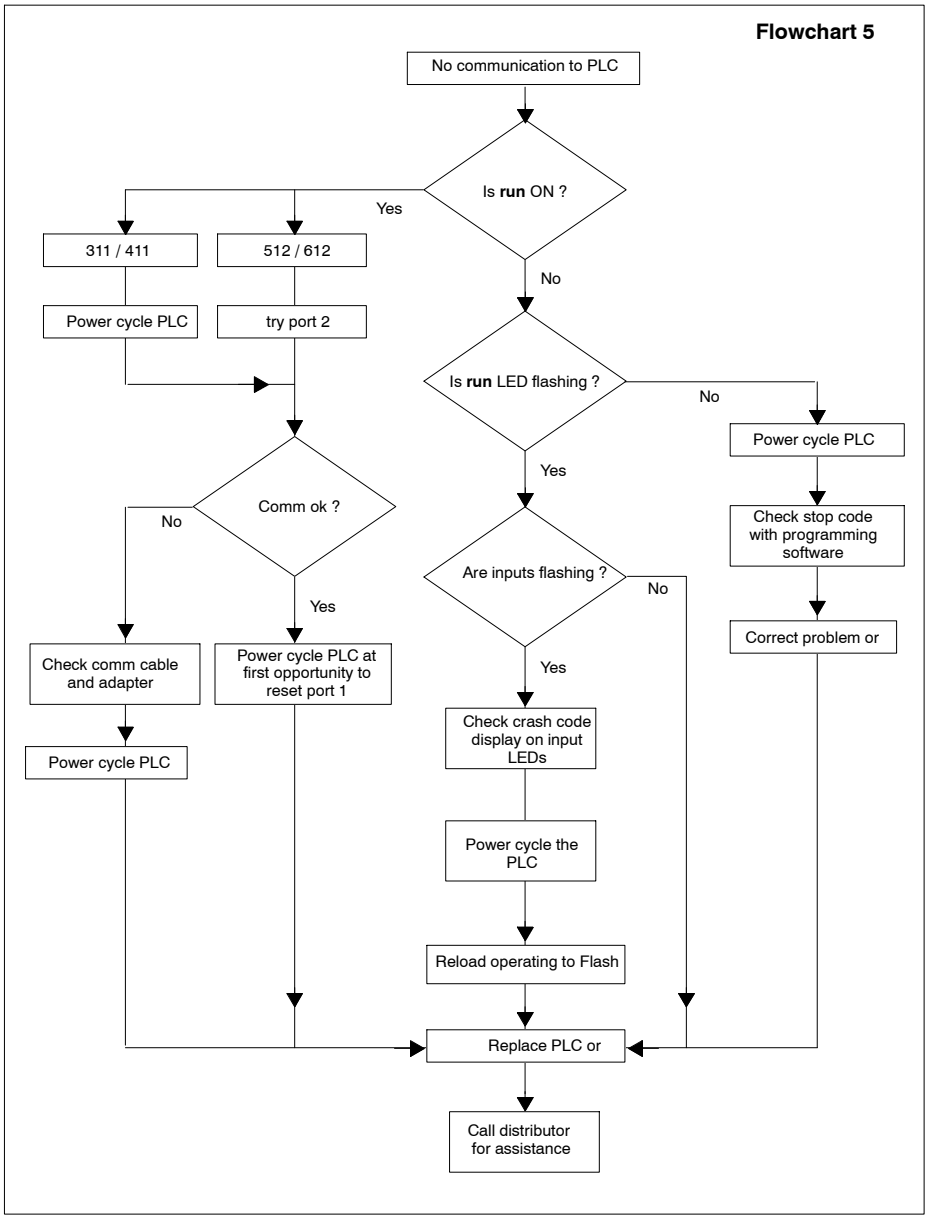
The green **exp link** LED goes ON steadily when valid communications are occurring on the I/O expansion link between a parent and child PLC; the same LED pattern should appear on both PLCs.. It flashes when errors are occurring on the link. If you see this LED flashing or if it is OFF when I/O communications should be occurring, refer to flowchart 4 on the next page.

Symptom: No Comms to the PLC

If communications fail unexpectedly on the PLC, refer to flowchart 5. You may also want to check for a PLC stopped error code displayed on your programming panel or a crash code display flashing on the input LEDs of the Micro PLC. The stopped error codes and system crash codes are described later in this appendix.


Flowchart 3
run not ON
Is power ok ON ?
No
See flowchart 1
Yes
Is ready ON ?
No
See flowchart 2
Yes
Is run flashing ?
Yes
See flowchart 5
No
Is run OFF ?
Yes
Connect Programmer
Start PLC
Is run ON ?
No
Check PLC Stopped Code
Correct Problem
Yes
Continue Operation


Flowchart 4
exp link OFF or flashing
Is PLC configured as a single ?
Yes
Configure as parent or child
No
Is run ON ?
No
See flowchart 3
Yes
Is exp link ON ?
Yes
Continue operation
No
Is exp link flashing ?
No
Replace PLC
Yes
Check cabling at parent and all childs
Are all child PLCs on the cable link ?
Yes
Check J2 screws for proper termination
run ON at all childs
Call distributor for assistance


Flowchart 5
No communication to PLC
Is **run** ON ?
Yes
No
311 / 411
512 / 612
Power cycle PLC
try port 2
Is **run** LED flashing ?
No
Power cycle PLC
Check stop code with programming software
Correct problem or
Yes
Comm ok ?
No
Yes
Are inputs flashing ?
No
Yes
Check comm cable and adapter
Power cycle PLC
Power cycle PLC at first opportunity to reset port 1
Check crash code display on input LEDs
Power cycle the PLC
Reload operating to Flash
Replace PLC or
Call distributor for assistance

PLC Stopped Error Codes

Should the PLC stop unexpectedly, you can find a stopped error code displayed in the panel software. The code will be displayed as a four-character hex number. In MODSOFT Lite, the stopped error code is shown in the PLC Status screen; on an HHP, the stopped error code is shown

The meanings of the various codes are listed in the table below.

Stopped Error Codes

Stop Bit Code	Stopped Condition	Description
8000	PCSTOPPED	The PLC is stopped
4000	BADTCOP	An error in the I/O map
2000	DIMAWAR	The PLC does not have a valid configuration
1000	PORTIVENT	Bad port intervention
0800	BADSEGSCH	Ladder logic segments are not scheduled properly for scanning
0400	SONNOTIST	The Start-of-network element did not start the network
0200	PDCHECKSUM	Bad power-down checksum diagnostic
0080	NOEOLDOIO	Watchdog timer expired befor the logic scan completed
0040	RTCFAILED	The real-time clock has failed
0020	BADOXUSED	An error in the coil-used table
0010	RIOFAILED	Failure on the I/O expansion link
0008	NODETYPE	An illiegal node type has been used
0004	ULCSUMERR	User logic checksum error
0002	DSCRDISAB	Discrete disable error
0001	BADCONFIG	Bad configuration

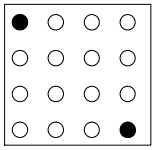
PLC Crash Codes Displayed on the LEDs

If the CPU detects a fatal error, one of the error codes listed below will be flashed on the input LED array on the front of the PLC. The **ready** LED will be ON steadily, and the **run** LED will flash at the same rate as the input LEDs—0.5 s ON and 2.5 s OFF.

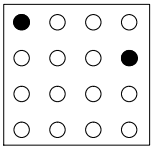
There are two categories of PLC crash codes, those generated by the PLC in *kernel* mode, and those generated during the application.

In the illustrations below, the flashing input LEDs are shown as ● and the OFF state input LEDs are shown as ○ .

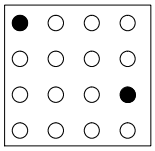
Kernel Mode Crash Codes



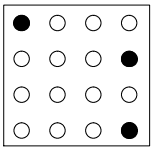
Kernel PROM checksum error



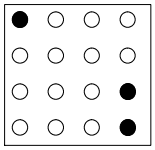
System RAM data error



Flash program/erase error



System RAM address error



Unexpected executive return

Application Crash Codes

○○○○ ○○○○ ○○○○ ○○○● PROM checksum error	○○○● ○○○● ○○○○ ○○○○ Run output active failed	○○○○ ○○○○ ○○●● Bad comm state—transmit RTU
○○○○ ○○○○ ○○○● ○○○○ RAM data test error	○○○● ○○○● ○○○○ ○○○● Unexpected interrupt	○○○○ ○○○● ○○●○ ○○○○ Bad comm state—received RTU
○○○○ ○○○○ ○○○● ○○○● RAM address test error	○○○● ○○○● ○○○● ○○●○ Bad UART hardware	○○○○ ○○○● ○○●○ ○○○● Bad comm state—received ASCII
○○○○ ○○○● ○○○● ○○○● Modbus command buffer overflow	○○○● ○○○● ○○○● ○○●● Bad UART interrupt (external to the microprocessor)	○○○○ ○○○● ○○●● ○○○○ Bad Modbus state—timer 0 event
○○○● ○○○○ ○○○○ ○○○○ Modbus command length = 0	○○○○ ○○○○ ○○●○ ○○○○ Bad receive comm state	○○○○ ○○○● ○○●● ○○○● Bad Modbus state—transmit interrupt
○○○● ○○○○ ○○○○ ○○○● Modbus abort command error	○○○○ ○○○○ ○○●○ ○○○● Bad transmit comm state	○○○● ○○○○ ○○●○ ○○○○ Bad Modbus state—received interrupt
○○○● ○○○○ ○○○● ○○○● RAM error during sizing	○○○○ ○○○○ ○○●● ○○○○ Bad comm state—transmit ASCII	○○○● ○○○○ ○○●○ ○○○● Timeout on kicking ready

Index

A

B

C

E

F

G

H

I

L

M

O

P

R

S

T

U

X