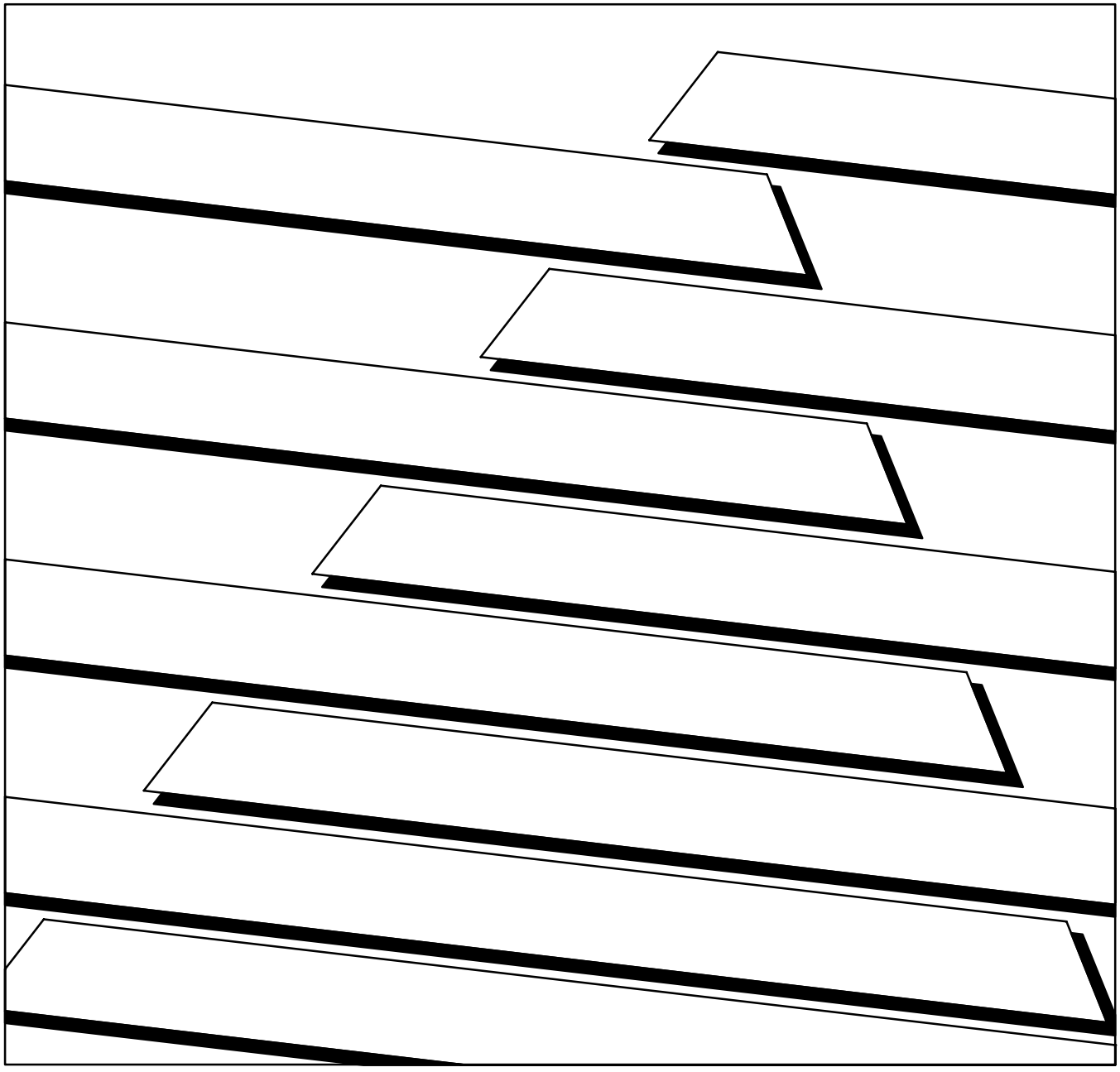
ALLEN-BRADLEY

PLC-3 Family Programmable Controller

Programming Reference Manual



Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes, and standards.

The illustrations, charts, sample programs, and layout examples shown in this guide are intended solely for example. Since there are many variables and requirements associated with any particular installation, Allen-Bradley does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley publication SGI–1.1, “Safety Guidelines For The Application, Installation and Maintenance of Solid State Control” (available from your local Allen-Bradley office) describes some important differences between solid-state equipment and electromechanical devices which should be taken into consideration when applying products such as those described in this publication.

Throughout this manual we make notes to alert you to possible injury to people or damage to equipment under specific circumstances.

WARNING: Tells readers where people may be hurt if procedures are not followed properly.

CAUTION: Tells readers where maninery may be damaged or economic loss can occur if procedures are not followed properly.

Warnings and Cautions:

- identify a possible trouble spot
- tell what causes the trouble
- give the results of improper action
- tell the reader how to avoid trouble

Important: We recommend that you frequently back up your application programs on an appropriate storage medium to avoid possible data loss.

Chapter 1

Using this Manual

1.0 Chapter Objectives

After reading this chapter you should know:

- what the manual contains
- who the manual is written for
- how the manual is organized

This chapter tells you how to use this manual properly and efficiently for the tasks you have to do. Read this chapter before you program a PLC-3 family programmable controller.

1.1 Manual's Purpose

This manual describes the concepts behind programming PLC-3 family processors. Included in this manual is detailed information on how the PLC-3 programming instructions work. By using the information presented in this manual, you should be able to develop ladder-diagram programs to control your application processes.

This manual does not provide information on loading ladder-diagram programs into the PLC-3 controller. For detailed information on loading PLC-3 ladder-diagram programs, refer to the PLC-3 Industrial Terminal (cat. no. 1770-T4) User's Manual (publication 1770-6.5.15).

1.2 Audience

Before attempting to execute programs on a PLC-3 family programmable controller, you should be familiar with the hardware components and installation procedures needed to operate the controller. If you are not familiar with this, you can refer to the following publications:

Publication	Title
1770-6.5.15	PLC-3 Industrial Terminal (Cat. No. 1770-T4) User's Manual
1775-6.3.1	PLC-3 Backup Concepts Manual
1775-6.5.1	Communication Adapter Module (Cat. No. 1775-KA)
1775-6.5.2	I/O Scanner-Programmer Module (Cat. No. 1775-S4A) User's Manual
1775-6.5.3	I/O Scanner-Message Handling Module (Cat. No. 1775-S4B) User's Manual
1775-6.5.4	Peripheral Communication Module (Cat. No. 1775-GA) User's Manual
1775-6.7.1	PLC-3 Family Installation and Operation Manual

You can also use our Publication Index (publication SD499) as a guide to further information about products related to our PLC-3 family of programmable controllers. Consult your local Allen-Bradley distributor or sales engineer for information regarding this publication or any needed information.

1.3 Vocabulary

We refer to certain types of equipment and terms throughout this manual. To make the manual easier for you to read and understand, we avoid repeating full product names where possible.

We refer to the :

- PLC-3 or PLC-3/10 programmable controller system as the **controller**
- Processor Module (cat. nos. 1775-L1, -L2, -L3, -L4) as the **processor**
- hardware device used to enter or load ladder-diagram programs into the PLC-3 processor as the **program loader**
- I/O scanner module (cat. nos. 1775-S5, -S4A, -S4B, -SR5, -SR) that scans the I/O chassis as the **scanner**
- ladder-diagram or user program that controls PLC-3 processor operation as the **ladder program**

1.4 Important Information

In this manual, there are three different types of important information:

- **WARNINGS** inform you where you could be injured if you do not follow the written procedure.
- **CAUTIONS** inform you where you could damage your equipment if you do not follow the written procedure.
- **IMPORTANTS** inform you of exceptions to general rules or remind you about important information.

1.5 Manual Organization

This manual is organized into the following chapters:

Chapter/ Appendix	Title	What is covered
1	Using this Manual	manual's purpose, audience, vocabulary, design, and lists related publications
2	Intorduction to Programming PLC-3 Family Controllers	memory organization and concepts used to program the processor
3	Using the Data Table	overview of the data table with a description for each section
4	Getting Started	introduction to the rung, relay-type instructions, I/O addressing formats, modes of operation, instruction set
5	Using Timers and Counters	how to use timers and counters in the ladder program
6	Using Data Manipulation Instructions	how to use data manipulation instructions in the ladder program
7	Using Files	concept of files for the processor
8	Using Data Manipulation Instructions with Files	how to use data manipulation instructions in the ladder program
9	Using Shift Registers	how to use shift register instructions to program synchronous and asynchronous shift registers in the ladder program
10	Indexing Bits within Files	concept of decimal bit addressing used with indexed logic instructions in the ladder program
11	Using Pointers for Indirect Addressing	concept of pointers and how to use pointer instructions in the ladder program
12	Using Diagnostic Instructions	how to use diagnostic instructions in the ladder program
13	Controlling Ladder Program Execution	how to use program control instructions in the ladder program, recovering from major faults, real-time interrupt, and switching contexts
14	Addressing Memory and Monitoring Controller Status	concept of extended addressing, status bit organization in memory
15	Executing Block Transfers	concept of block transfer and using block-transfer instructions in the ladder program
16	Using the Message Instruction	how to use the message instruction to execute tasks on other PLC-3 modules
17	Writing the Ladder Program	tips on writing the ladder program
A	Instruction Set Execution Times and Memory Usage	typical times for the processor to execute the instructions and the amount of memory used for each instruction
B	Numbering Systems	binary, decimal, integer, octal, hexadecimal, high-order integer, and floating-point numbering systems
C	Memory Management Forms	forms you can use to organize your I/O and data table assignments
D	Glossary	listing of words and definitions pertaining to PLC-3 programming
E	Ladder Instruction Listings	listings of the entire instruction set with abbreviations for each instruction

Chapter 2

Introduction to Programming PLC-3 Family Controllers

2.0 Chapter Objectives

This chapter introduces concepts behind programming the controller. In this chapter we describe the:

- definition of memory for the processor
- principle sections of memory
- organization of memory

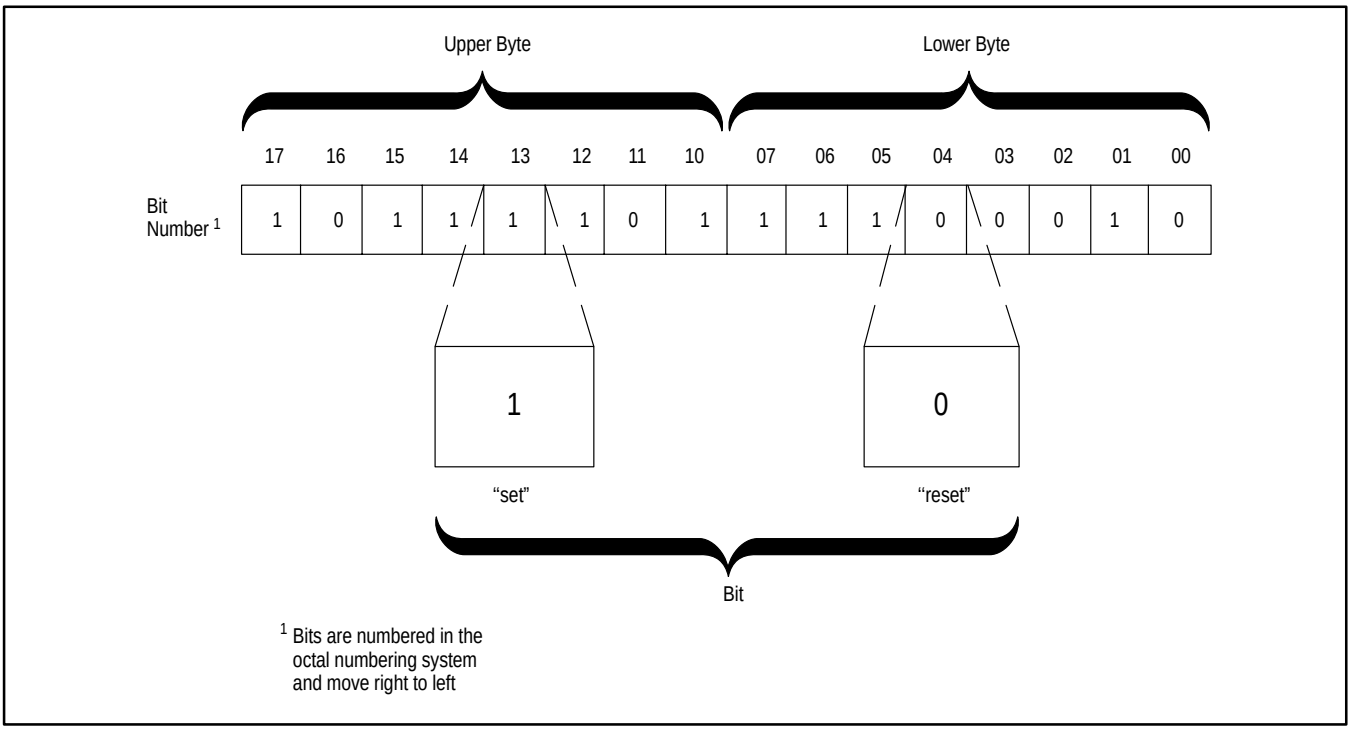
2.1 Storing Information in the Controller

Memory for the processor stores the information necessary for operation. This information includes the ladder-program instructions and other data. By monitoring or sending information to memory, you can see or tell the processor what to do.

You can think of memory as a large array of storage points. Each separate storage point is called a BInary digiT or BIT (Figure 2.1). A Bit is the smallest unit of information that memory can retain. Each bit in memory stores binary values, or values in base two. Thus, each bit can store:

- 1, meaning that the bit is on or **set**
- 0, meaning that the bit is off or **reset**

Figure 2.1
Structure of a Memory Word that Stores and Transmits Complete Units of Information



A group of eight bits forms a **byte.** A byte is defined as the smallest complete unit of information that can be transmitted to or from the processor at a given time.

A group of 6 bits makes up a **word.** A word can be thought of as being made up of two or four 8-bit bytes. The total number of words in the processor gives you the basis for the memory size, with 1K equal to 1,024 words.

The processor Chassis (cat. no 1775-A1, -A2, -A3) for PLC-3 family controllers must contain a memory module that contains the memory for the processor. Four memory modules are currently available:

Catalog Number	Contains
1775-ME4, -MS4	16K or 16,384 words
1775-ME8, -MS8	32K or 32,768 words
1775-MEA, -MSA	64K or 65,536 words
1775-MED	128K or 131,072 words

2.2 Interface Between Ladder Program and Hardware

Later in this chapter we explain the organization of memory. Before doing so, this section explains how the controller uses machine data, sensed by 1771 input modules to turn output devices on or off with 1771 output modules. This hardware-program interface occurs between two areas in memory:

- Data table which stores status and numeric data
- Ladder program which stores instruction that you use to control your application

2.2.1 I/O Image Tables in the Data Table

Within the data table, input and output image tables store the status of input and output devices connected to 1771 I/O modules. The primary purpose of the input image table is to duplicate the status of the input devices wired to 1771 input module terminals:

If the input device is	Then its corresponding input image table bit is
on	set
off	reset

You program instruction in the ladder program to monitor bits in the input image table.

The purpose of the output image table is to control the status of output devices wired to 1771 output module terminals:

If an output image bit is	Then its corresponding output device is
set	on
reset	off

You can program instructions in the ladder program to control bits in the output image table.

2.2.2 Addressing Instructions

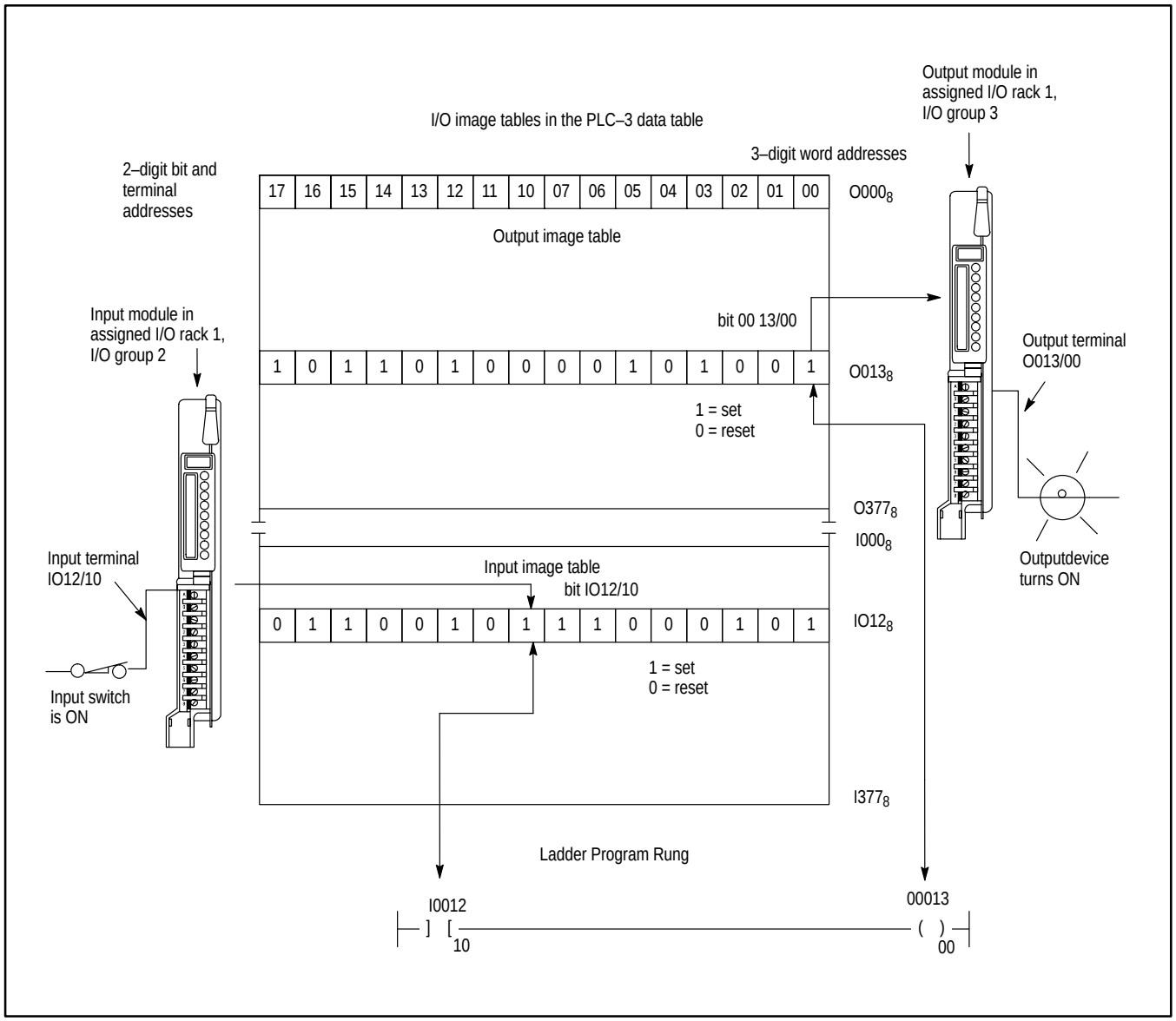
In programming instructions, you enter a code or an address that references an I/O image table location in the data table which corresponds to a hardware location in a 1771 I/O chassis

We describe instruction addressing in chapters 3 and 4.

2.2.3 Operation of the Ladder Program

Figure 2.2 illustrates the operational relationship between the input and output devices, the input and output image tables, and the ladder program.

Figure 2.2
Relating the Ladder Program to the Hardware



When an input switch connected to terminal I12/10 closes, the 1771 input module circuitry senses a voltage. The scanner reads the input status and sets input image table bit I12/10. During the program scan, the processor examines bit I12/10 for a set condition. If the bit is set, the **examine on** instruction is logically true, and a path of logic continuity is established which causes the rung to be true. The processor then sets output image

table bit O13/00 to 1. During the next I/O scan, the processor tells the output module to turn on output point O13/00, turning on the output device wired to this terminal.

When an input switch connected to terminal I12/10 opens, the 1771 input module circuitry senses no voltage. The scanner reads the input status and resets input image table bit I12/10. During the program scan, the processor examine bit I12/10 for a set condition. Since the bit is reset, a path of logic continuity is not established which causes the rung to be false. The processor then resets output image table bit O13/00 to 0. During the next I/O scan, the processor tells the output module to turn off output point O13/00, turning off the output device wired to this terminal.

2.3 Organization of Memory

A memory map is a chart that shows how processor memory is organized. The processor memory map contains the following areas:

Area	Contents
0	system status
1	system pointers
2	module status
3	data table
4	ladder program
5	message
6	system symbols
7	system scratchpad
8	converted procedures
10	force table
60	free memory
63	end of memory

We describe these memory areas in the following sections.

2.3.1 System Status (Area 0)

The system status area stores data used by the processor to monitor system operation. If the processor detects a fault condition, it alerts you through the front panel or CRT on the program loader.

Words in the system status area store the following information:

- System counters coordinate the timing of the processor's response to requests made by the modules within the processor.
- Module identification identifies modules in the processor chassis.
- Minor faults are problems that are not serious enough to cause the processor to stop control action.

- Major faults are serous problems that cause the processor to stop control action.
- Controller operation defines various parameters for processor operation such as the operating mode.
- Watchdog timer sets the maximum program scan time.
- Time-of-day clock and calendar set the time and date for the processor.

The system status area has a fixed size of 22 words. The remaining memory areas vary depending on the amount of data that you store in them.

2.3.2 System Pointers (Area 1)

This area contains the system pointers which are used to define actual physical address of the first word of each implemented area of memory.

The system pointers function like the individual entries within a table of contents. Just as each entry in a table of contents tells you the starting page number for a chapter, each pointer tells the system controller where each subsection starts in memory.

2.3.3 Module Status (Area 2)

This area describes the modules in the processor and includes the:

- total number of modules in the processor
- types of modules and revision levels
- fault status of each module
- proper module configurations, such as thumbwheel settings, communication rates, etc.

2.3.4 Data Table (Area 3)

This area contains information needed to execute the ladder program. This information includes:

- input/output status
- timer and counter data
- numeric data
- other data used by the ladder program

We describe the contents of the data table in chapter 3.

2.3.5 Ladder Program (Area 4)

This area contains the ladder-program instructions that are scanned and executed by the processor. The ladder program is divided into three sections:

- main program
- subroutine
- fault routine

Refer to chapters 4, 13, and 14 respectively.

2.3.6 Message (Area 5)

This area stores messages that you enter into the controller. The processor can prompt, display, or document these messages. The message area is divided into five sections:

- report generation
- rung comments
- terminal commands (MACROS)
- data highway
- assistance (HELP)

2.3.7 System Symbols (Area 6)

This area stores alphanumeric names that represent an address, message, or report-generation procedure. This area also stores program labels, which are numbers that you can use to identify parts of the ladder program. We describe the use of program labels in chapter 13. We describe system symbols in chapter 14 and 17.

2.3.8 System Scratchpad (Area 7)

This area stores variables used in report generation by the I/O Scanner-Message Handling Module (cat. no. 1775-S4B).

Important: This section is reserved for processor operation. You cannot access it when programming.

2.3.9 Converted Procedures (Area 8)

This area stores the contents of converted procedures used in GA Basic by the Peripheral Communication Module (cat. no. 1775-GA).

Important: This section is reserved for processor operation. You cannot access it when programming.

2.3.10 Force Table (Area 10)

This area stores data that reflects forced I/O conditions. You can specify forced I/O conditions to cause selected input bits to be forced (set or reset) regardless of the actual condition of corresponding input hardware and selected output bits to be forced (set or reset) regardless of the output's

state in the ladder program. You can implement forced I/O through the program loader or the front panel. Refer to the PLC-3 Programmable Controller Installation an Operation Manual (publication 1775-6.7.1)

2.3.11 Free Memory (Area 60)

This area identified the amount of unused memory. You can monitor free memory by using the memory map feature.

2.3.12 Reserved Areas and End of Memory

Areas 9 and 11 through 63 are undefined by the processor. The actual end of memory is area 63.

Important: These sections are reserved for processor operation. You cannot access them when programming.

Chapter 3

Using the Data Table

3.0 Chapter Objectives

In chapter 2, we introduced the areas that make up memory. In this chapter, we describe the contents of the data table.

3.1 What is the Data Table?

The data table is a portion of memory that contains the following information that is used by the processor to execute the ladder program:

- input/output status
- timer and counter data
- numeric data
- other data required by the ladder program

Figure 3.1 shows a map of the sections that make up the data table area. You can alter the size of each section to meet your application needs.

To address a section, you identify it by entering its section specifier (Table 3.A). Refer to chapter 14 for detailed information on addressing memory. Table 3.A also summarizes data types and the acceptable ranges for storing values in the data table sections.

Important: In addition most data table sections support structures called files. This chapter does not describe the file structure for the data table section. For detailed information on using files, refer to chapter 7.

3.2 Input/Output Status

You can monitor the status of I/O points in sections one (output image table) and two (input image table).

Figure 3.1
PLC-3 Data Table Map Showing the Contents of the Data Table Area

Section Number	Title	Maximum Size	Address Range
1	Output Image Table	4,096 values	$O0000_8$ to $O0377_8$
2	Input Image Table	4,096 values	$I0000_8$ to $I0377_8$
3	Timer Table (3 words/timer)	10,000 timers	T0 to T9999
4	Counter Table (3 words/counter)	10,000 counters	C0 to C9999
5	Integer Table (1 word/value)	10,000 values	N000:0000 to N000:9999
6	Floating Point Table (2 word/value)	10,000 values	F000:0000 to F000:9999
7	Decimal Table (1 word/ 4BCD values)	10,000 values	D000:0000 to D000:9999
8	Binary Table (1 word/value)	10,000 values	B000:0000 to B000:9999
9	ASCII Table (2 Charaacters/word)	20,000 characters	A000:0000 to A000:9999
10	High–order–integer Table (2 words/value)	10,000 values	H000:0000 to H000:9999
12	Pointer Table	10,000 addresses	P000:0000 to P000:9999
13	Status Table	10,000 values	S000:0000 to S000:9999

Table 3.A
Section Specifiers, Data Types, and Acceptable Ranges for Values Stored in the Data Table

Data Table Section	Section Specifier	Type of Data Stored in Section	Range	
			Low Limit	High Limit
Output image	O	Unsigned binary	0	65,535
Input image	I	Unsigned binary	0	65,535
Timer	T	Unsigned binary	0	65,535
Counter	C	Binary[1]	-32,768	32,767
Integer	N	Binary[1]	-32,768	32,767
Floating point	F	Floating point	±2.939 E-39	±1.701 E+38
Decimal	D	Binary coded decimal	0	9,999
Binary	B	Unsigned binary	0	65,535
ASCII[2]	A	Unsigned binary	– – – – –	– – – – –
High order integer	H	Binary[1]	-2,147,483,648	2,147,483,647
Pointer	P	Unsigned binary[3]	– – – – –	– – – – –
Status	S	Unsigned binary[3]	– – – – –	– – – – –

[1]The processor stores positive numbers in straight binary and negative numbers in two's complement form.

[2]The ASCII table can store ASCII characters as defined by ASCII (ANSI X3.4).

[3]The processor treats data in the pointer and status sections as unsigned binary, although these sections are intended to store non-numeric data.

3.2.1 Output Image Table

This section controls the status of output devices wired to 1771 I/O modules:

If an output image bit is	Then the corresponding output is
set	on
reset	off

Instructions in the ladder program control these output image bits in the data table.

Each bit in the output image table corresponds to a 1771 output module terminal. Figure 3.2 shows the correlation between the image tables and the 1771 I/O module. Refer to chapter 4 for detailed information on hardware addressing. To control the maximum amount of I/O points, you need:

- 512 words to control 64 racks of I/O for the PLC-3 controller
- 128 words to control 16 racks of I/O for the PLC-3/10 controller

One assigned I/O rack is equivalent to 128 I/O points. To address a word or bit in the output image table, use the following address format (Figure 3.3).

For example, the address O13/01 corresponds to an output terminal located in a chassis having assigned I/O rack number 1, I/O group 3, terminal 1. The status of this output is located in word 13, but 1 of the output image table.

3.2.2 Input Image Table

This section controls the status of input devices wired to 1771 I/O modules:

If the input switch is	Then the corresponding input bit is
on	set
off	reset

Instructions in the ladder program monitor these input image bits in the data table.

Each bit in the input image table can correspond to a 1771 input module terminal. Figure 3.2 shows the correlation between the input image table and the 1771 input module. Refer to chapter 4 for detailed information on hardware addressing. To control the maximum amount of I/O points, you need:

- 512 words to control 64 racks of I/O for the PLC-3 controller
- 128 words to control 16 racks of I/O for the PLC-3/10 controller

One assigned I/O rack number is equivalent to 128 I/O points. An I/O rack could have one or two I/O rack numbers assigned to it. To address a word or bit in the input image table, use the address format shown in Figure 3.3.

For example, the address I12/01 corresponds to an input terminal located in a chassis having assigned I/O rack number 1, I/O group 2, terminal 1. The status of this output is located in word 12, bit 1 of the input image table.

The address I12/10 corresponds to an input terminal located in a chassis having assigned I/O rack number 1, I/O group 2, terminal 10. The status of this input is located in word 12, bit 10 of the input image table.

Figure 3.2
Relating an I/O Hardware Location to the Input and Output Image Tables

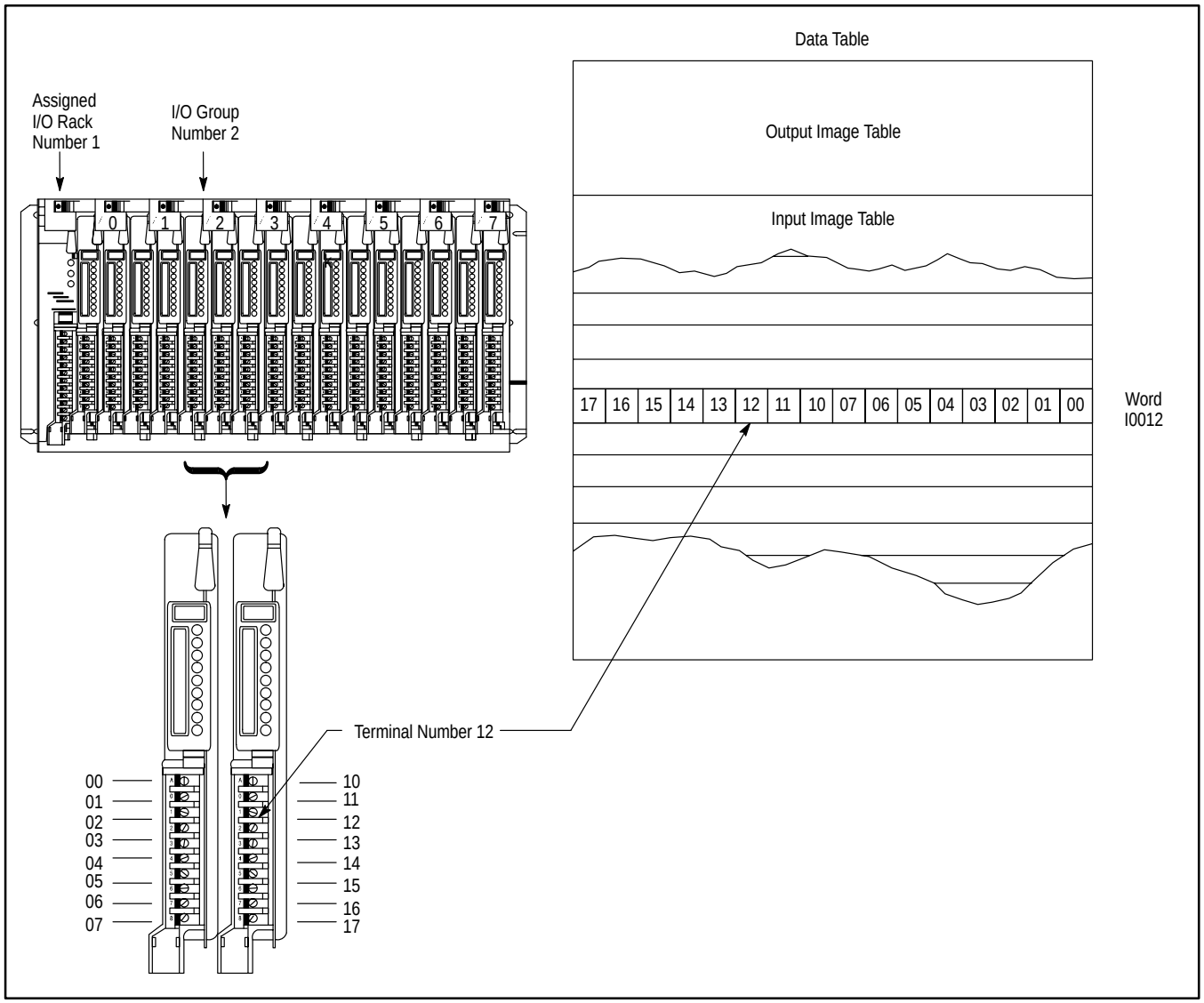
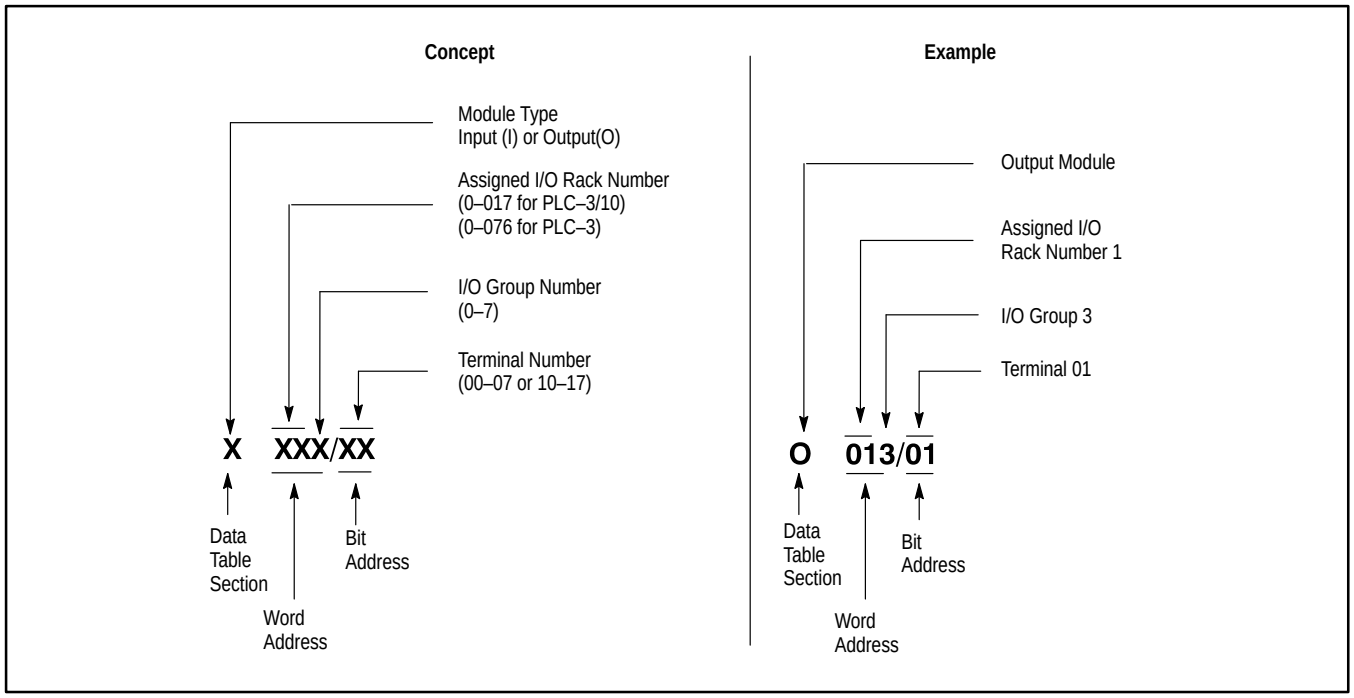


Figure 3.3
I/O Addressing Terminology Between the Hardware and the Data Table



3.3 Timer and Counter Data

You can use sections three (timer table) and four (counter table) to monitor timer and counter instruction in the ladder program.

3.3.1 Timer Table

This section monitors the status of timer instructions in the ladder program. Each timer takes up three words of memory. To address a timer, enter T<timer number>.

For example, the address T0 corresponds to timer 0. The status of timer 0 is stored in three words in the timer table. We describe the use of these words in chapter 5.

3.3.2 Counter Table

This section monitors the status of counter instructions in the ladder program. Each counter takes up three words of memory. To address a counter, enter C<counter number>.

For example, the address C10 corresponds to counter 10. The status of counter 10 is stored in the words in the counter table. We describe the use of these words in chapter 5.

3.4 Numeric and Alphanumeric Data

You can use sections:

- five (integer table)
- six (floating-point table)
- seven (decimal table)
- eight (binary table)
- nine (ASCII table)
- ten (high-order-integer table)

to monitor numeric data for the ladder program.

3.4.1 Integer Table

This section stores signed integers in a binary format. To address the integer table, enter N<word>/<bit>.

For example, the address N33 corresponds to the word 33 in the integer table of the data table.

3.4.2 Floating-point Table

This section stores signed floating-point values used in sophisticated arithmetic. Each value can be taken to eight-digit precision, although only six digits display on the industrial terminal. To address the floating-point table, enter F<word>/<bit>.

For example, the address F3 corresponds to the word 3 in the floating-point table of the data table.

3.4.3 Decimal Table

This section stores numeric values that support I/O devices such as thumbwheel switches and decimal readouts. Each value is stored as a positive integer in binary coded decimal (BCD) format. To address the decimal table, enter D<word>/<bit>.

For example, the address D1 corresponds to the word one in the decimal table of the data table.

3.4.4 Binary Table

This section stores numeric values used in bit and logical operations. Each value is stored in an unsigned binary format. To address the binary table, enter B<word>/<bit>.

For example, the address B1 corresponds to the word one in the binary table of the data table.

3.4.5 ASCII Table

This section stores alphanumeric data to be processed. The processor stores ASCII data used as numeric data in an unsigned binary format. The range for this format is 0 to 65,535. To address the ASCII table, enter A<word>/<bit>.

For example, the address A4 corresponds to the word four in the ASCII table of the data table.

3.4.6 High-order-integer Table

This section stores numeric data used in high-precision arithmetic operations. Each value is stored in a 32-bit signed integer format. To address the high-order-integer table, enter H<word/bit>.

For example, the address H5 corresponds to the word five in the high-order-integer table of the data table.

3.5 Other Data Table Sections

The data table also contains sections 12 (pointers) an 13 (status).

3.5.1 Pointers

This section stores address data that you can use for indirect addressing in the ladder program. That is by using pointers, program logic can change the address for a particular instruction automatically while the program is executing. Do not confuse ladder-program pointers with system pointers. System pointers are used by the controller to define physical addresses for the first word of each implemented area of memory. Refer to chapter 11 for detailed information on using pointers.

3.5.2 Status

This section stores status information monitored by the ladder program for proper operation. The processor transfers this status information from the system status area of memory to this data table section before it executes the ladder program. This section stores information on:

- major fault status
- minor fault status
- system operating status
- Time-of-day clock and calendar

We describe the organization of the status section in chapter 14.

Chapter 4

Getting Started

4.0 Chapter Objectives

In this chapter we describe information you need to develop programs for the controller. After reading this chapter, you should know how to:

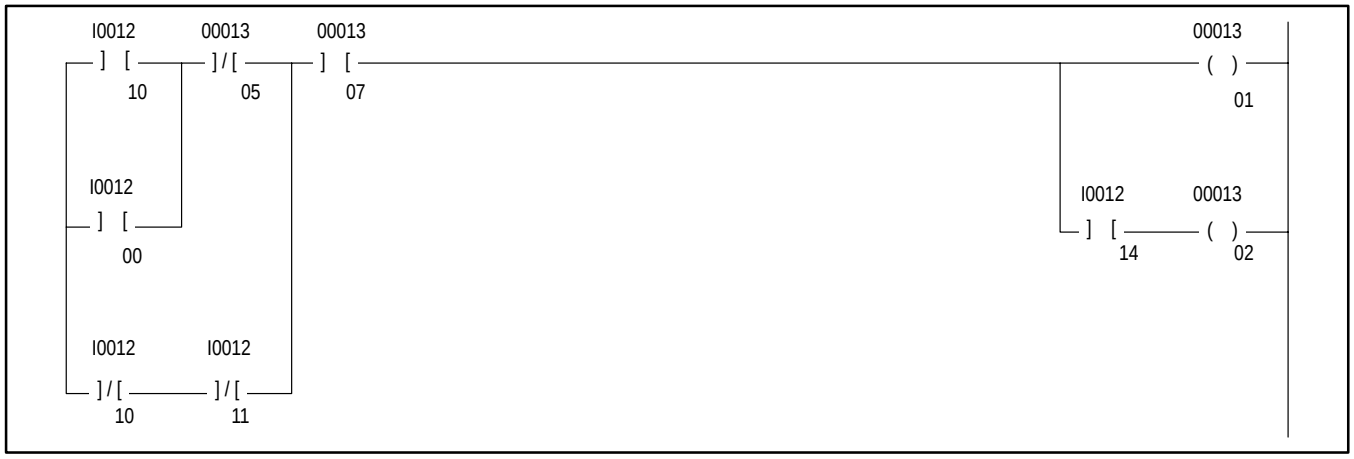
- use logic rungs to develop the ladder program
- identify I/O locations through the ladder program and the data table
- execute and monitor the ladder program
- use relay-type instructions in the ladder program

4.1 What is a Logic Rung?

Figure 4.1 shows you a sample logic rung that could be executed by the processor. A logic rung is a line or segment of the ladder program that specifies:

- the output(s) to be controlled by the processor. These outputs are identified by bits in the output image table. You program output instructions on the rung to tell the processor what output to control. The processor can have more than one output instruction on a rung.
- the input condition(s) for the output(s). These inputs are identified by bits in the input image table. Inputs can be monitored for a set or reset state. Input instructions on the rung tell the processor when to control the output.
- the path or paths for “logic continuity” to the output. Such paths can follow series or parallel logic. You can branch instructions to establish the logic path from the input(s) to the output(s).

Figure 4.1
Logic Rung



4.1.1 Identifying I/O Locations

Before we describe relay-type instructions, you need to know how to identify an I/O location in the ladder program. As we stated in chapter 3, bits in the input image table reflect the status of input devices connected to input terminals on 1771 input modules, while bits in the output image table control the status of output devices connected to output terminals on 1771 output modules.

Figure 4.2
Addressing I/O Locations

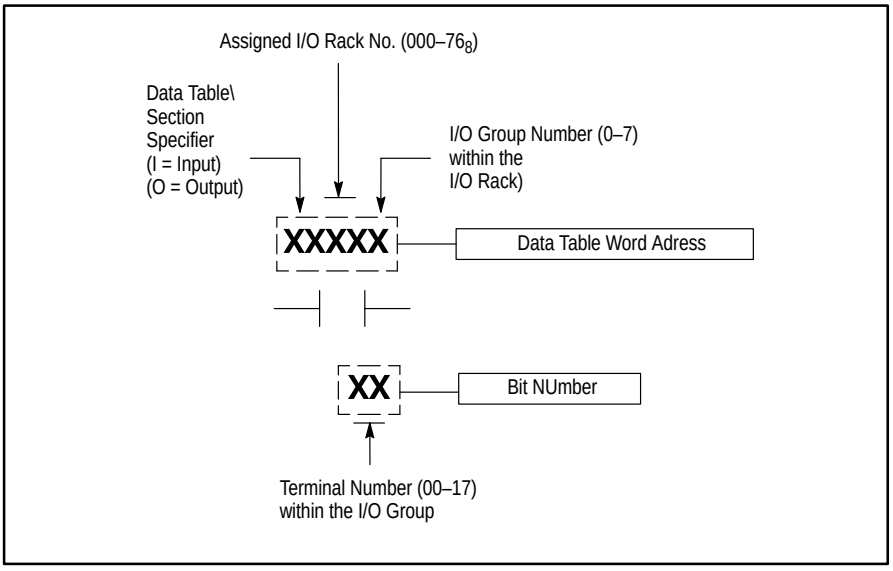
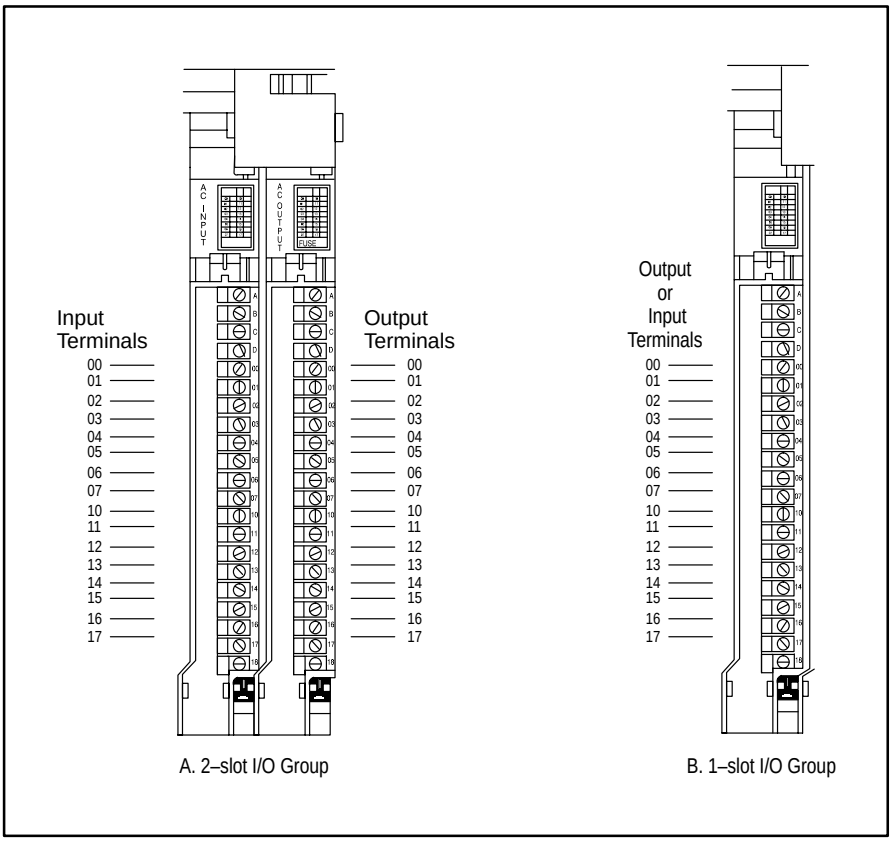


Figure 4.3
An I/O Group Consists of Up to 16 Input Terminals and 16 Output Termnials



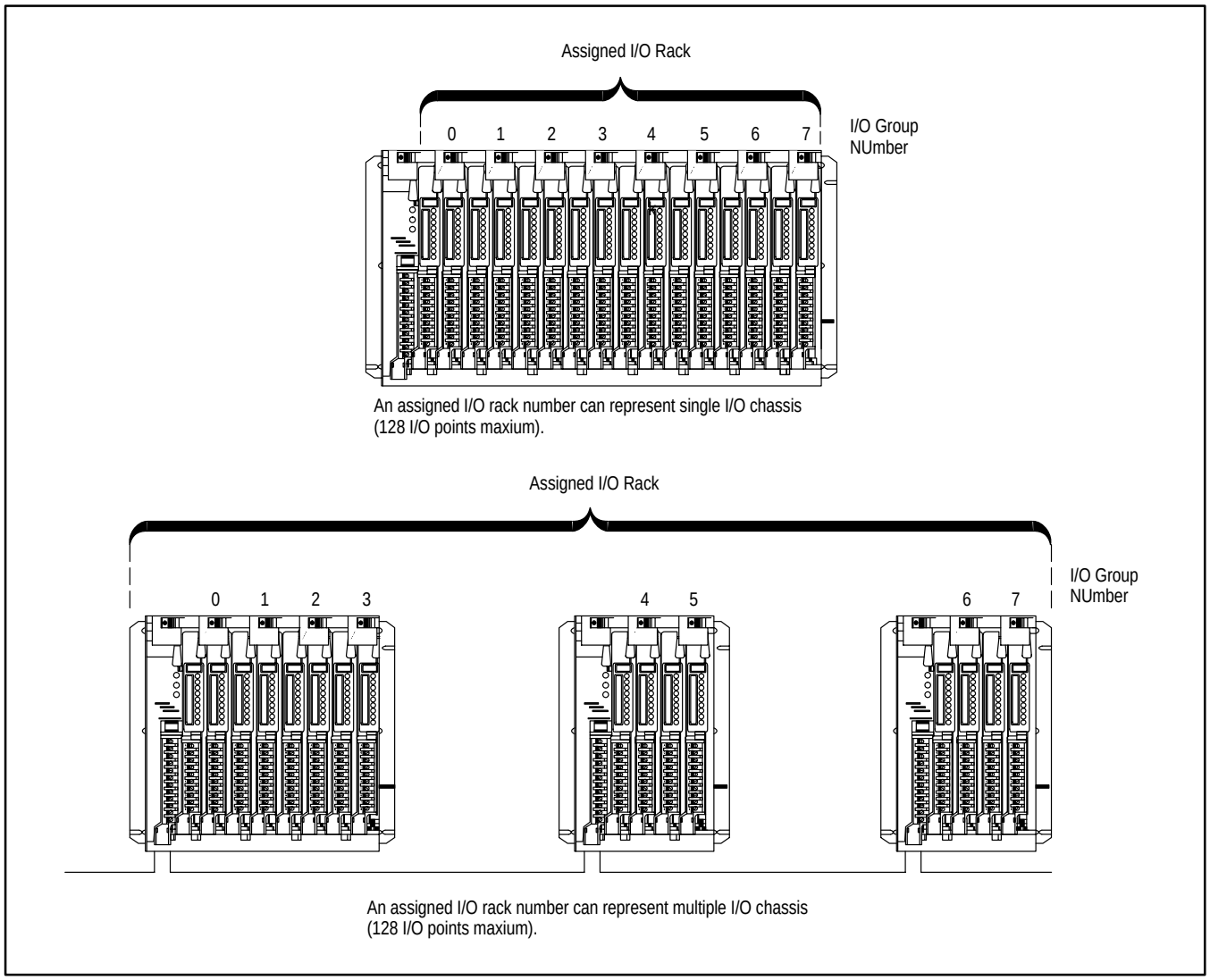
The controller communicates to these I/O modules through a Remote I/O Adapter modules (cat. no. 1771-AS, -ASB). To specify I/O locations, you enter an I/O address in the ladder program that the processor uses in communicating with the I/O adapter module. Figure 4.2 shows you the format for such an address.

To specify an I/O address, you need to provide the processor with the following information:

Terminal – tells the processor the terminal number on the I/O module that is connected to the input or output device. The terminal number corresponds to a bit address in the input or output image table.

I/O Group – tells the processor which I/O group within and I/O chassis contains the I/O module. An I/O group is made up of I/O terminals and can consist of up to 16 input terminals and/or 16 output terminals (Figure 4.3).

Figure 4.4
An I/O Rack Consists of Up to Eight I/O Groups



Depending on your I/O configuration, an I/O group can take up 1/2, 1, or 2 slots in an I/O chassis. We refer to the addressing methods for these configurations as 1/2-slot addressing, 1-slot addressing, and 2-slot addressing:

If you are using the	Then you can use
1771-AS adapter	2-slot addressing only
1771-ASB adapter	1/2, 1, or 2-slot addressing

If you are using a 1771-ASB adapter, you select an I/O chassis to have either 1/2, 1, or 2-slot I/O groups by setting a switch on the I/O chassis backplane. Refer to the Remote I/O Adapter Module (cat. no. 1771-ASB)

User's Manual (publication 1771-6.5.37) for detailed information on switch settings.

Rack - tells the processor the assigned rack number of the I/O chassis that contains the I/O module. A rack is an I/O addressing unit that corresponds to eight I/O groups (Figure 4.4). Note that a rack of I/O does not necessarily correspond to one I/O chassis.

2-slot Addressing

When you select 2-slot addressing, the processor addresses two I/O module slots as one I/O group.

Each physical 2-slot I/O group is represented by a word in the input image table and a word in the output image table. Each input terminal corresponds to a bit in the input image table word; each output terminal corresponds to a bit in the output image table word.

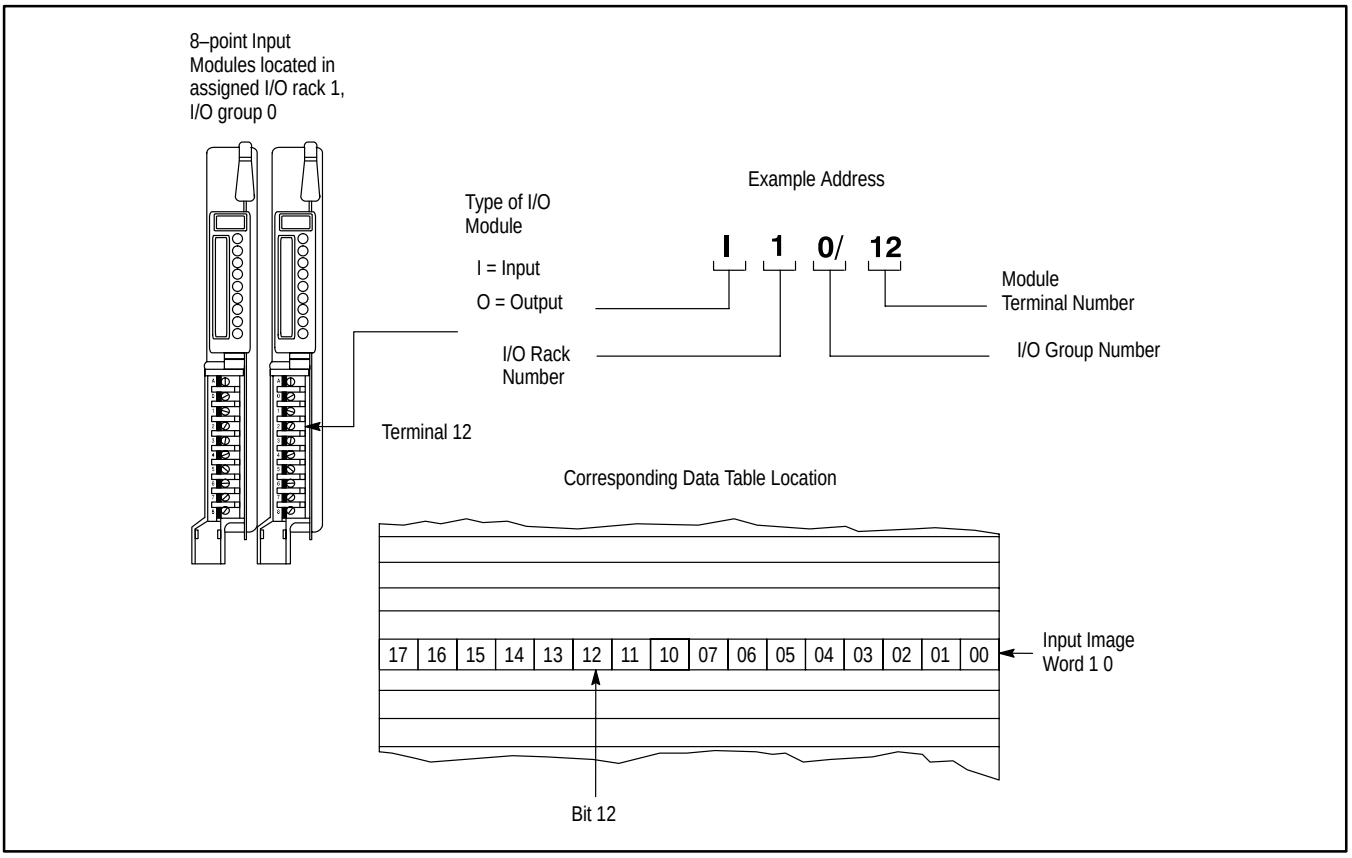
The maximum number of bits available for one 2-slot I/O group is 32; 16 in the input image table and 16 in the output image table.

The type of module that you install determines the number of bits in the words that are used. You can use either 8 to 16-point I/O modules with 2-slot addressing but not 32-point I/O modules.

Using 8-point I/O Modules

8-point I/O modules provide eight input or output terminals. Figure 4.5 shows an example address and how that address corresponds to the 2-slot I/O group concept with an 8-point input module and an 8-point output module.

Figure 4.5
Example Address that Identifies 2-slot I/O Groups with 8-point I/O Modules



Using 16-point I/O Modules

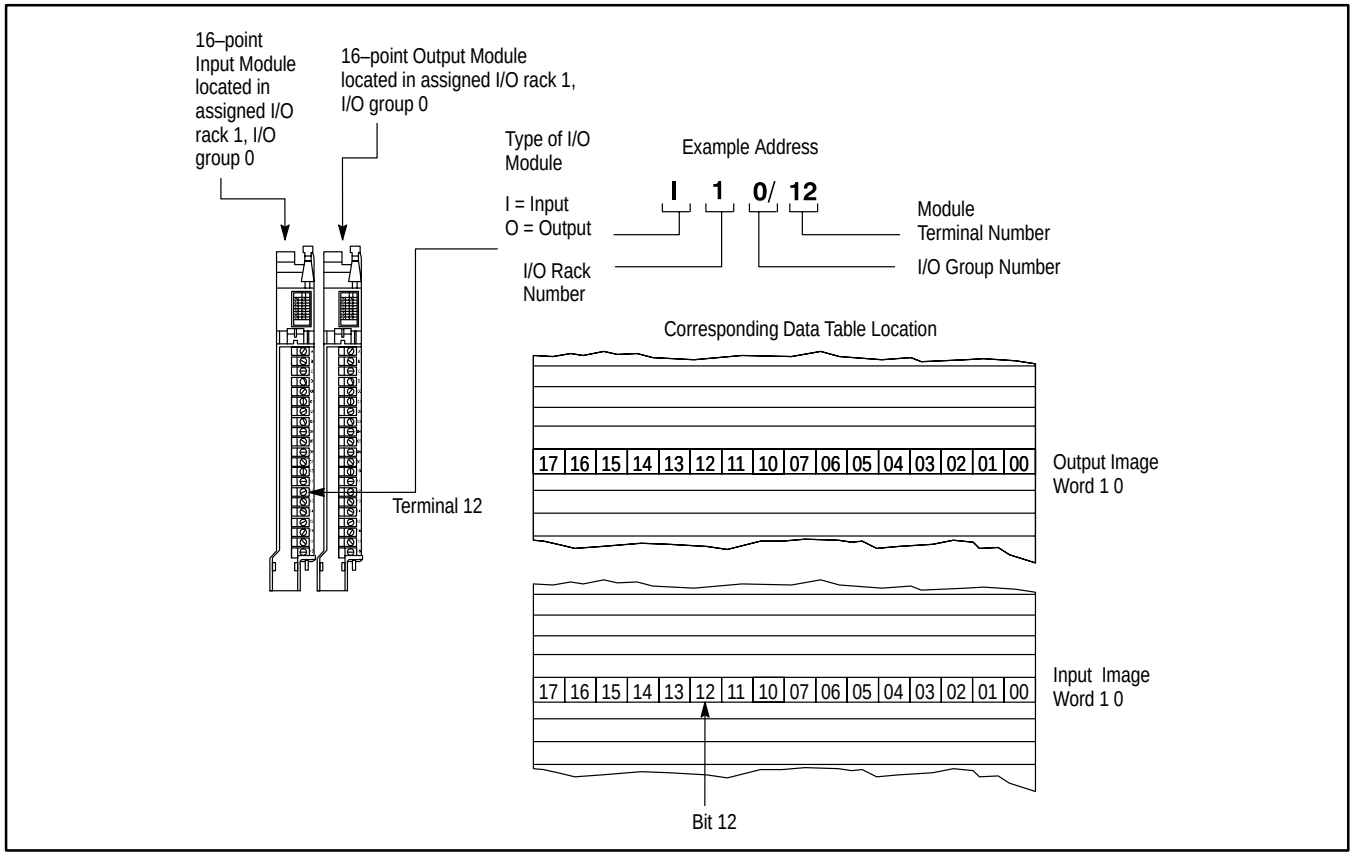
16-point I/O modules provide 16 input terminals or 16 output terminals. A 16-point I/O module uses a full word in the input or output image table when it is addressed in a 2-slot I/O group. Two 16-point I/O modules (one input and one output) can be used in a 2-slot I/O group.

Because these modules use a full word in the image table, the only type of modules that you can use in a 2-slot I/O group with a 16-point I/O module is one that performs the opposite (complementary) function. An input module complements an output module and vice-versa.

You can use an 8-point I/O module with a 16-point module in a 2-slot I/O group but it must perform the opposite function. That is, one input module and one output module. However, eight bits in the I/O image table are unused.

Figure 4.6 shows an example address and how that address corresponds to the 2-slot I/O group concept with a 16-point input module and a 16-point output module.

Figure 4.6
Example Address that Identifies 2-slot I/O Groups with 16-point I/O Modules



1-slot Addressing

When you select 1-slot addressing, the processor addresses one I/O module slot as one I/O group.

The physical address of each I/O group corresponds to an input and output image table word. The type of module that you install determines the number of bits in these words that are used.

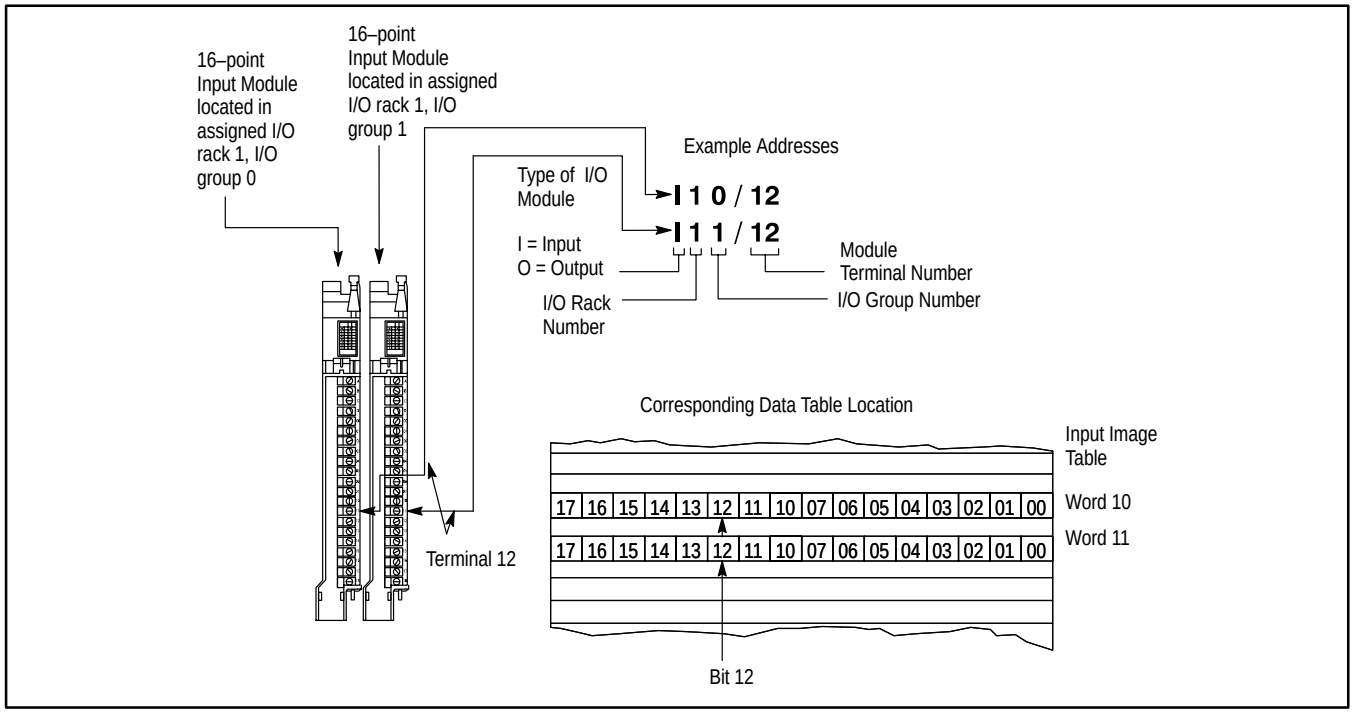
With 1-slot addressing, 16 input bits and 16 output bits are available in the input image table for each I/O group. Therefore, you can use any mix of 8-point, 16-point, or block-transfer modules, in any order, and you need only eight slots of a chassis to achieve 128 I/O.

When you use 8-point I/O modules with 1-slot addressing, only eight bits of the I/O image table word are used for that I/O group.

You can use 32-point I/O modules with 1-slot addressing with restrictions described in the next section.

Figure 4.7 shows example adresses and how these addresses correspond to the 1-slot I/O group concept with 16-point I/O modules.

Figure 4.7
Example Address that Identifies 1-slot I/O Groups with 16-point I/O Modules



Using 32-point I/O Modules

32-point I/O modules need 32 input or 32 output bits in the I/O image table. Since only 16 input and 16 output bits are available for each 1-slot I/O group, the processor uses the address of the unused input or output word associated with the adjacent I/O slot to address a 32-point I/O module.

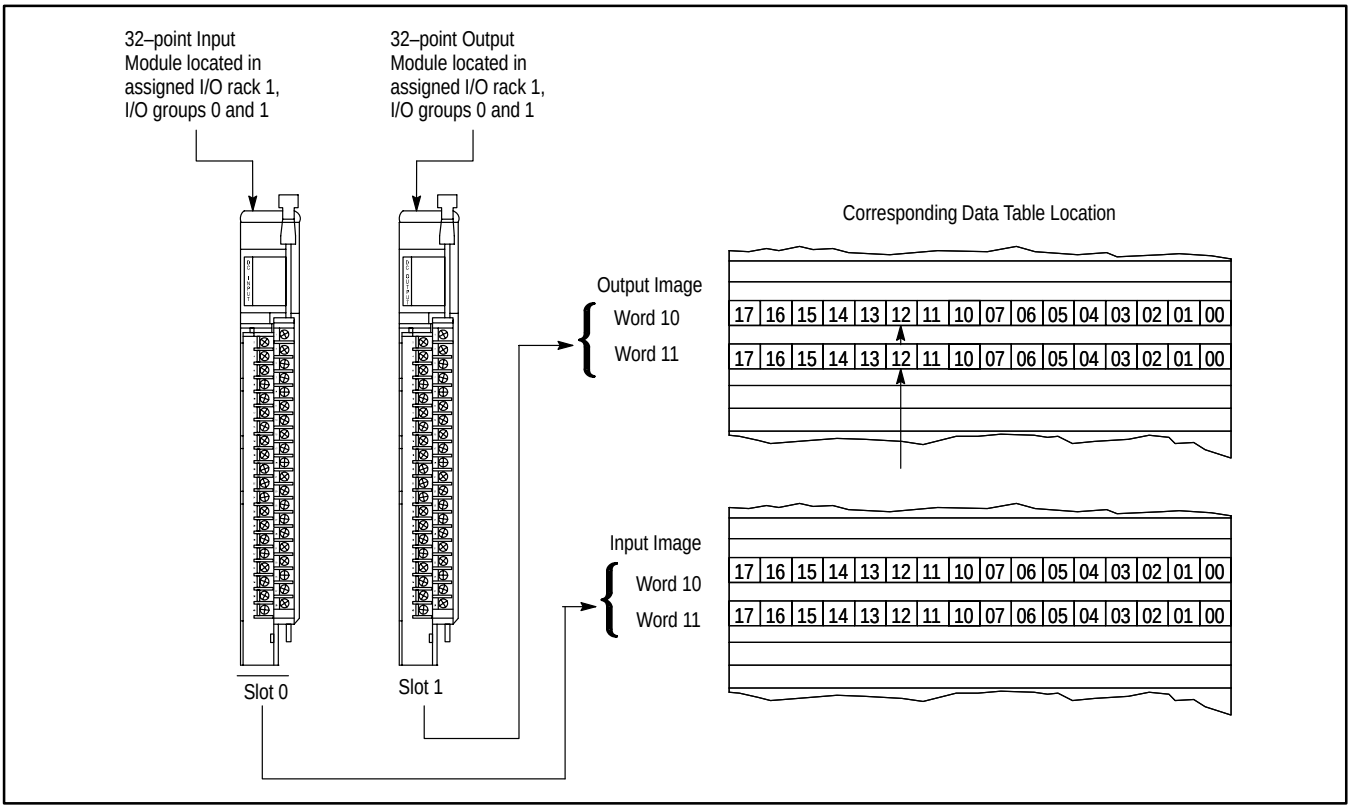
To use 32-point I/O modules with 1-slot addressing, you must install the modules as pairs in two adjacent slots of an I/O chassis beginning with I/O slot 0. A pair can consist of:

- a 32-point input module and an output module
- a 32-point output module and an input module

If you cannot pair the modules in this way, one of the two slots of the pair must be empty.

Figure 4.8 illustrates 1-slot addressing with two 32-point I/O modules.

Figure 4.8
1-slot I/O Group with 32-point I/O Modules



You cannot use Thermocouple Input Modules (cat. no. 1771-IX, -IY) in the same I/O chassis with 32-point I/O modules. If you need a thermocouple module, use a Thermocouple/Millivolt Module (cat. no. 1771-IXE).

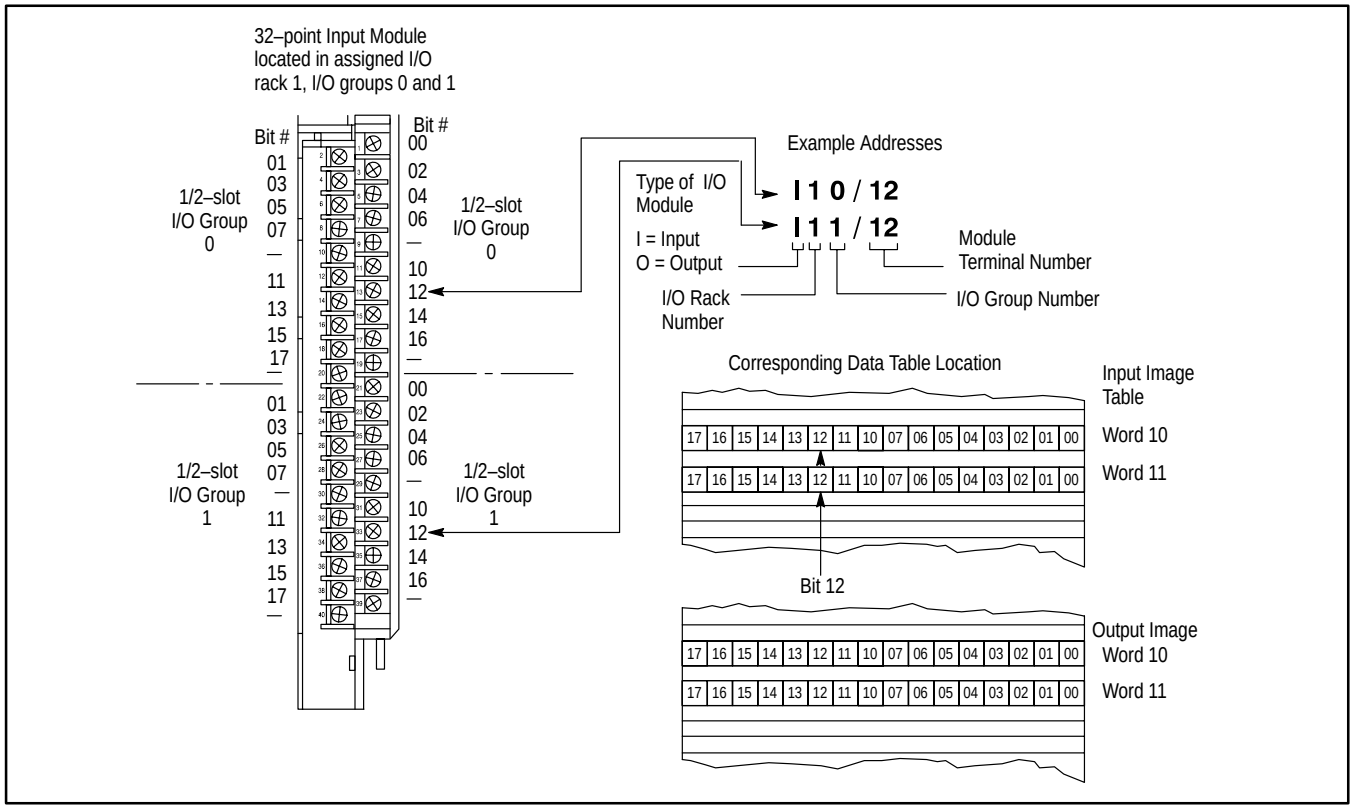
1/2-slot Addressing

When you select 1/2-slot addressing, the processor addresses one-half of an I/O module as one I/O group. The physical address of each I/O slot corresponds to two input and two output image table words. The type of module that you install determines the number of bits in these words that are used.

With 1/2-slot addressing, since 16 input bits AND 16 output bits are available in the processor's image table for each I/O group, you can mix 8, 16, and 32-point I/O modules in any order in the I/O chassis.

Figure 4.9 illustrates the 1/2-slot I/O group concept with a 32-point I/O module. A 32-point I/O module (two 1/2-slot I/O groups) uses two words of the image table. When you use 8 or 16-point I/O modules with 1/2-slot addressing, you get fewer total I/O.

Figure 4.9
1-slot I/O Group with 32-point I/O Modules



To address a block-transfer module in a 1/2-slot I/O group, use the lower assigned rack number and the lower assigned I/O group number of the slot(s) in which the module resides.

You cannot use Thermocouple Input Modules (cat. no. 1771-IX, -IY) in the same I/O chassis with 32-point I/O modules. If you need a thermocouple module, use a Thermocouple/Millivolt Module (cat. no. 1771-IXE).

4.2 Using Relay-type Instructions

By using the following input and output instructions, you can tell the processor to control outputs based on monitoring inputs:

- examine on
- examine off
- energize output

4.2.1 Examine On (XIC)

Required Parameters: Bit address in the data table.

Description: The examine-on instruction tells the processor to monitor a bit in the data table:

If the input switch is	Then the bit is
on	set
off	reset

You can address bits for all sections of the data table, but for now the examples in this chapter show you addresses for bits located in the input and output image tables.

Example: Consider the following examine on instruction:

```
I0012
–] [–
    07
```

This instruction tells the processor to examine input image word 12, bit 07 for a set condition. This bit corresponds to terminal 7 of an input module that is assigned I/O rack 1, I/O group 2.

4.2.2 Examine Off (XIO)

Required Parameters: Bit adress in the data table.

Description: The examine-off instruction tells the processor to monitor a bit in the data table:

If the input switch is	Then the bit is
off	set
on	reset

You can address bits for all sections of the data table, but for now the examples in this chapter show you addresses for bits located in the input and output image tables.

Example: Consider the following examine off instruction:

```
I0012
–]/[–
   03
```

This instruction tells the processor to examine input image word 12, bit 03 for a reset condition. This bit corresponds to terminal 3 of an input module that is in assisgned I/O rack 1, I/O group 2.

4.2.3 Output Energize (OTE)

Required Parameters: Bit address in the data table.

Description: The output-energize instruction tells the processor to control a bit in the data table based on the rung conditions:

If the rung is	Then the processor turns the output bit
true	on
false	off

You can address bits for all sections of the data table, but for now the examples in this chapter show you addresses for bits located in the input and output image tables.

Example: Consider the following output energize instruction:

```
O0013
–( )–
   01
```

This instruction tells the processor to set output image word 13,, bit 01 if the input condition(s) for the rung are true. This bit corresponds to terminal 1 of an output module that is in assigned I/O rack 1, I/O group 3.

4.3 Preparing to Program the Processor

Now that you know three relay-type instructions, you can enter and execute a simple ladder program. To program the processor, you need to connect a program loader such as the industrial terminal (cat. no. 1770-T4) to the processor. For detailed information on processor installation procedures and proper operation, refer to the PLC-3 Family Controller Installation and Operation Manual (publication 775-6.7.1, formerly 1775-800).

To begin programming:

1. Turn on the processor. If you have properly installed the PLC-3 components, the following message should briefly display on the front panel:

   A B PLC-3

2. Turn on the program loader. The ladder programming display should appear on the CRT screen. You are now ready to program the processor. For detailed information on loading programs, refer to the user's manual for your program loader.

4.3.1 Modes of Operation

The processor has three modes of operation:

- program load mode for entering and editing ladder program instructions. In the program load mode, the processor does not execute the ladder program and outputs are disabled.
- test mode for testing the ladder program. In the test mode, the processor executes the ladder program with outputs disabled.
- run mode for controlling outputs with the processor. In the run mode, the processor controls outputs by executing the ladder program.

When you first turn on the processor, it operates in the program load mode. Although you can program the processor in the run mode, we recommend not doing so until you are thoroughly familiar with its operation.

To summarize the modes of operation:

If the processor is in	Then ladder program execution is	And outputs are
program load mode	disabled	disabled
test mode	enabled	disabled
run mode	enabled	enabled

You can select the operating mode for the processor through the LIST function or program loader commands. For detailed information on selecting the operating mode through the:

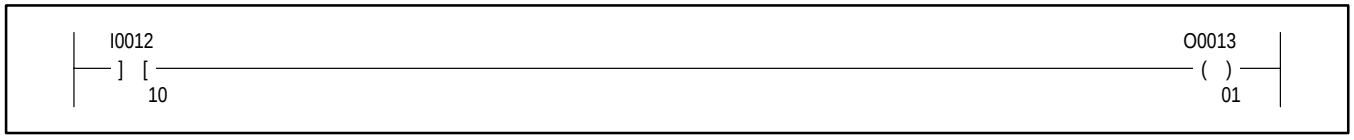
- LIST function, refer to the PLC-3 Family Controller Installation and Operation Manual (publication 1775-6.7.1)
- program loader, refer to the user's manual for your program loader

4.3.2 A Simple Rung

Figure 4.10 shows a simple rung consisting of an examine on instruction and an output energize instruction. In executing this rung, the processor sets output image word 13, bit 1 when input image word 12, bit 10 is set. If this input image bit is set, the processor resets the output bit.

If you enter this rung and put the processor in the run mode, it executes the rung. For this example, if you have a switch connected to terminal 10 of an input module in assigned I/O rack 1, I/O group 2, it could control an output device connected to terminal 1 of an output module in assigned I/O rack 1 I/O group 3.

Figure 4.10
Example Rung that Turns On an Output Bit if an Input Bit is Set



4.3.3 A Simple Rung with Multiple Inputs

Figure 4.11 shows more than one input instruction in series. In executing this rung, the processor sets output word 13, bit 2 if:

- input word 12, bit 10 is set, **and**
- input word 12, bit 11 is set, **and**
- input word 12, bit 01 is set

If any of these input bits are reset, the processor resets the output bit.

If you put the processor in the run mode, the output device connected to terminal 2 of an output module in assigned I/O rack 1, I/O group 3 does not turn on unless all three of the input switches are on.

Figure 4.11
Example Rung that Turns On an Output if Multiple Input Bits are Set

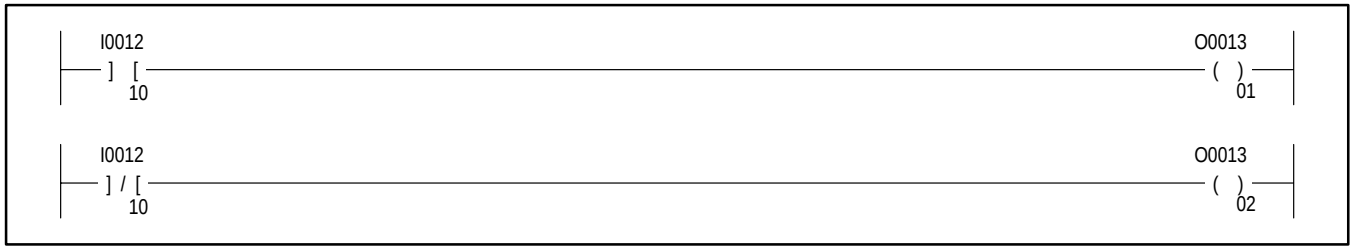
4.3.4 A Simple Rung with the Examine-off Instruction

Figure 4.12 shows two simple series rungs examining the same input bit. In executing these rungs, the processor examines input word 12, bit 10:

If the bit is	Then the processor sets
set	output word 13, bit 1
reset	output word 13, bit 2

If you put the processor into the run mode, the controller would turn on output devices connected to terminals 1 and 2 of an output module in assigned I/O rack 1, I/O group 3 depending on the condition of terminal 10 of an input module in assigned I/O rack 1, I/O group 2.

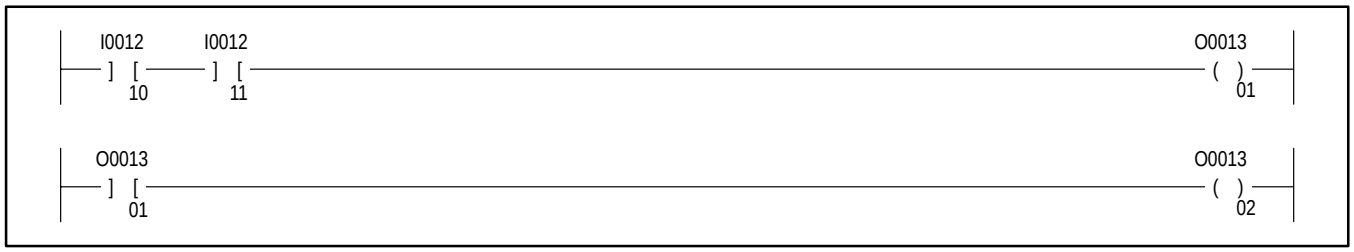
Figure 4.12
Example Rungs that Set an Output Bit if an Input Bit is Set or Set Another Output Bit if the Same Input Bit is Reset



4.3.5 Examining Output Bits

In the examples given so far, we have only examined input image table bits. In some applications, examining output bits can also be useful (Figure 4.13).

Figure 4.13
Example Rungs that Set an Output Bit if Two Input Bits are Set and Set Another Output Bit when the First Output Bit is Set

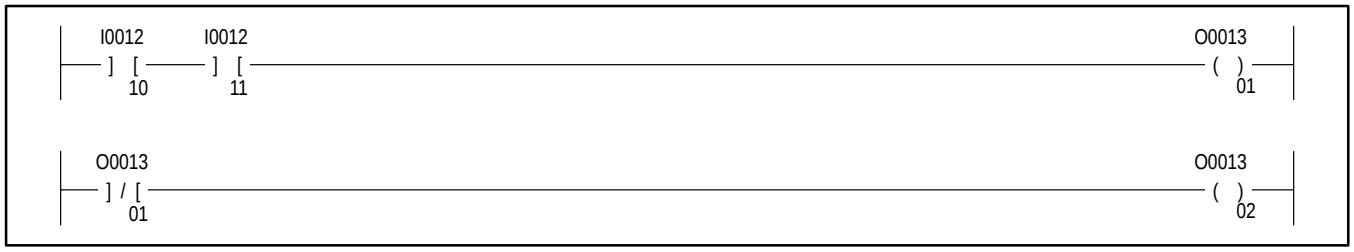


In executing the first rung, the two examine on instructions control an output. In executing the second rung, the output is controlled by the first rungs output. If the output bit in the first rung is set, the output bit in the second rung is set.

Figure 4.14 shows two more rungs.

In executing the first rung, the two examine on instructions control an output. In executing the second rung, the output is controlled by the first rungs output. If the processor resets the output bit in the first rung, it sets the output bit in the second rung.

Figure 4.14
Example Rungs that Set an Output Bit if Two Input Bits are Set and Set Another Output Bit if the First Output Bit Remains Set



4.4 Using Branch Instructions

Using branch instructions enables you to program parallel conditions on logic rungs. These are two types of branching (Figure 4.15):

- Input branching tells the processor that there is more than one logic path for logic continuity to an output.
- output branching tells the processor that there is more than one output decision to be made on a rung. Output branches can include more input conditions.

To program branches, refer to the user's manual for your program loader.

Figure 4.15
Input and Output Branching

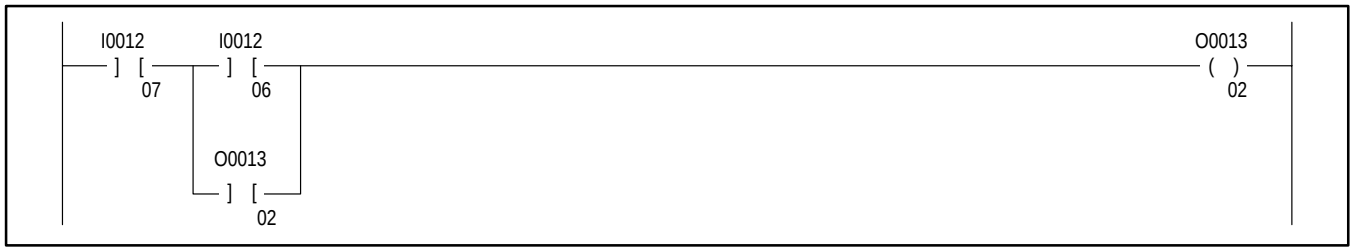
4.4.1 A Rung with a Hold-in Branch

Figure 4.16 shows a logic rung that operates similar to a hard-wired hold-in circuit. In executing this rung, the processor turns on the output if:

- I12/07 is set, **and**
- I12/06 **or** O13/02 is set

As long as input I12/07 is set and input I12/06 is set even momentarily, output O13/02 is set an is “held in” by the examine-on O13/02 instruction located in the branch.

Figure 4.16
Example Rung with a “Hold-in” Branch

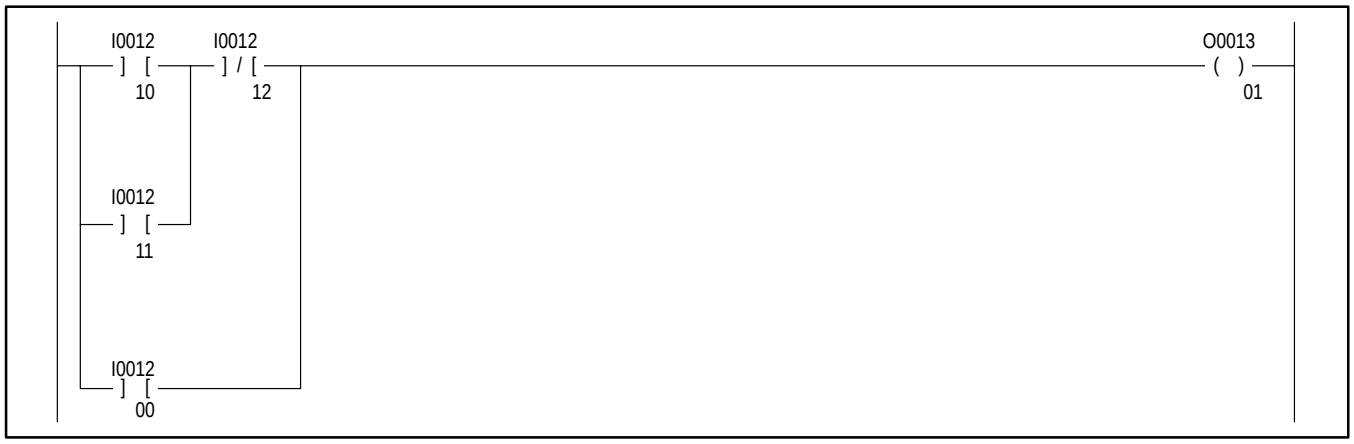


4.4.2 A Rung with an Input Branch within a Branch

Figure 4.17 shows a logic rung that has one branch circuit contained or **nested** within another branch. In executing this rung, the processor sets output bit O13/01 when:

- I12/10 or I12/11 and I12/12 is reset, **or**
- I12/00 is set by itself

Figure 4.17
Example Rung with a Branch within a Branch



4.5 Using Retentive Relay-type Instructions (OTL, OTU)

Required Parameters: Bit address in the data table.

Description: The output-latch and output-unlatch instructions are retentive meaning that they retain their last state in memory.

The output-latch instruction tells the processor to hold a bit in the data table set regardless of the rung conditions:

If the rung is	Then the processor
true	sets and latches the output bit
false	does not set the output bit

If the processor sets the output bit, the output bit remains set even after the rung input conditions go false. To reset the bit, you can use the output-unlatch instruction which performs the opposite operation:

If the rung is	Then the processor
true	resets the output bit
false	does not reset the output bit

During controller operation, if:

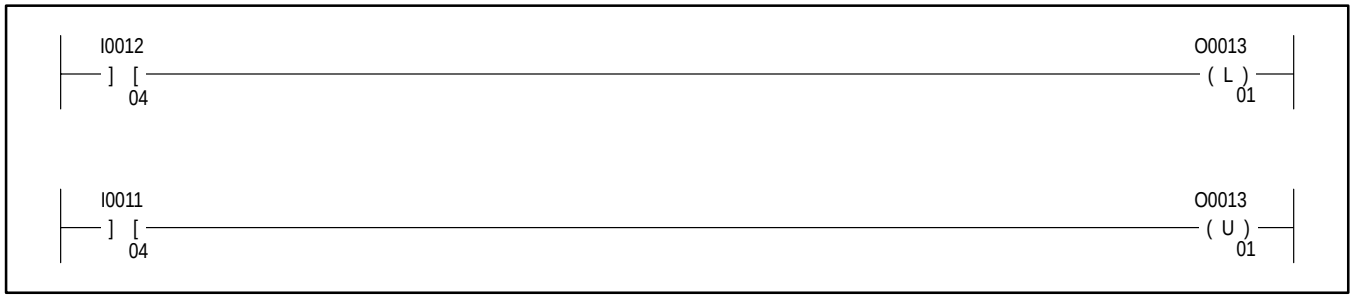
- you change the operating mode from run to program load, the last true output latch or unlatch instruction continues to control the output bit in memory.
- power is lost, and provided there is a battery backup for the memory module, the last true output latch or unlatch instruction continues to control the output bit in memory. The on or off status of the output device corresponding to the output bit depends on your selection for the last state switch on the I/O chassis:

If the last state switch is	The output device controlled by the output bit
on	remains in its last state
off	turns off but turns on again when you put the processor back into run mode even though the rung conditions controlling the output latch instruction may be false

Example: Figure 4.18 shows two rungs that contain the latch and unlatch instructions. In executing the first rung, the processor sets (latches) the output bit if the input is set. In executing the second rung, the processor resets (unlatches) the output bit if the input bit is set.

CAUTION: If the input conditions on both rungs are true at the same time, improper machine/process operation could result.

Figure 4.18
Example Rungs that Latch an Output Bit if an Input Bit is Set and Unlatch the Bit if Another Input Bit is Reset



Chapter 5

Using Timers and Counters

5.0 Chapter Objectives

Now that you have been introduced to programming the controller, this chapter begins explaining the instruction set. After reading this chapter, you should understand how the processor uses:

- timer instructions to time intervals determined by the ladder program
- counter instructions to count events determined by the ladder program

5.1 Using Timers

You can use a timer in applications where a time delay is required before the processor turns an output on or off. Timer instructions are output instructions with two associated 5-digit values:

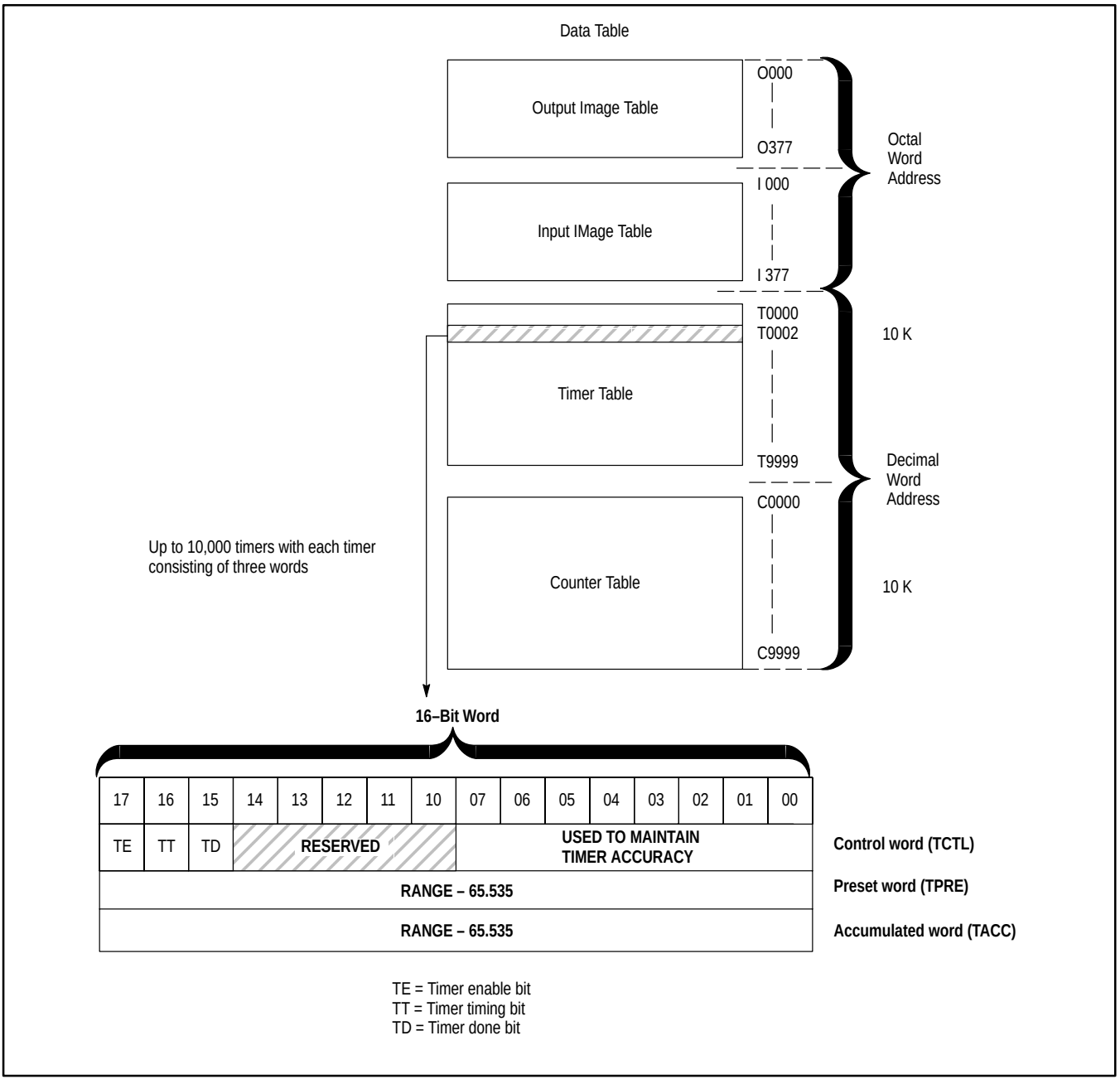
- **preset value** – You specify this value; which the timer must reach before it takes action. When the accumulated value equals the preset value, the timer instruction sets a status bit. You can use this bit of control an output device.

- **accumulated value** – current number of time increments that have elapsed. The timer instruction updates this value as long as it is enabled.

The range for these values is 0 to 65,535 and the processor stores them in a binary format. If you are using a move instruction to transfer data to or from these words, the data must follow this format (see chapter 6).

Figure 5.1 shows the timer section of the data table. You can program up to 10,000 timers (T0 to T9999) with each timer instruction requiring three words in the data table:

Figure 5.1
Memory Storage for Timers



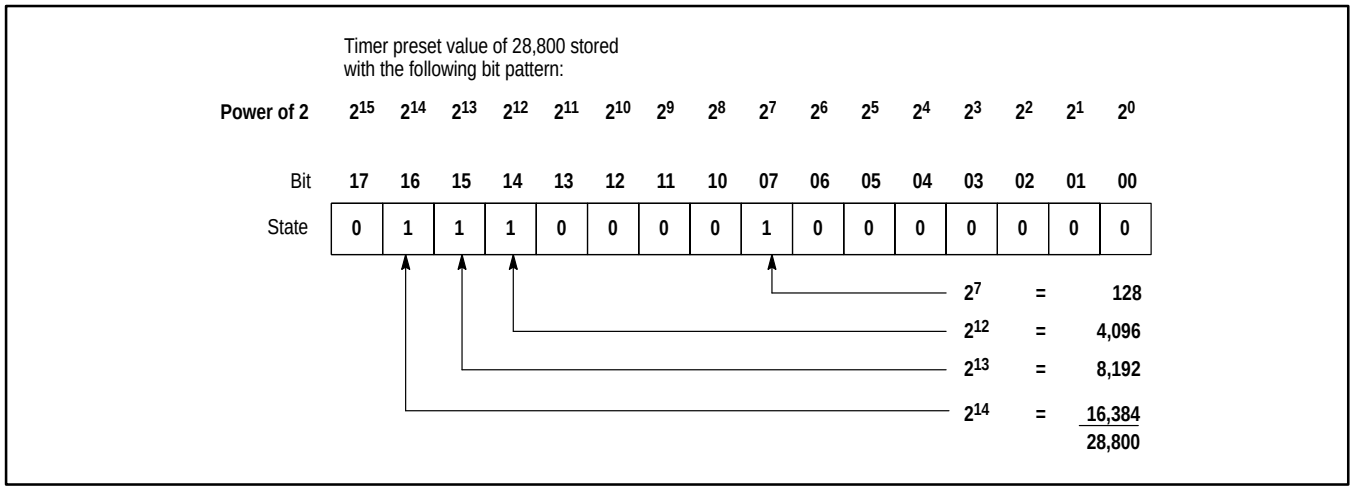
Control word (TCTL) contains the control bits that reflect the status of the timer instruction and can be examined in the ladder program:

- Timer enable – bit 17 (TE) shows that a timing operation has been enabled.
- Timer timing – bit 16 (TT) shows that a timing operation is in progress.

- Timer done – bit 15 (TD) shows that a timing operation is complete.

Preset value word (TPRE) contains a decimal value representing the time delay for the timer instruction. This value can range from 0 to 65,535 and is stored in 16-bit binary by the processor (Figure 5.2).

Figure 5.2
The Processor Stores Timer Preset and Accumulated Values in 16-bit Binary



Accumulated value word (TACC) contains a value representing the total number of time increments that have elapsed for a timer instruction since it was enabled. This value can range from 0 to 65,535 and is stored in 16-bit binary by the processor (Figure 5.2).

5.1.1 Selecting a Time Base

In general, you can use the following guidelines:

If you want to time a process for	Then use this time base
99 to 65,535 s	1.0 s
2.0 to 99.9 s	0.1 s
0.02 to 2.00 s	0.01 s

The maximum range of the timer is 65,535 times the time base. Actual range depends on the preset value and the time base. For example:

If the timer preset is	And the time base is	Then the timer times for
1000	1.0 s	1000 s
1000	0.1 s	100 s
1000	0.01 s	10 s

In many applications, timing with a 0.1-s time base provides accuracy comparable to or greater than typical electromechanical timing relays.

In addition to the three fixed time bases, you can use the CPU scan as the time base. When you do, the processor increments the timer one unit each time it scans the timer instruction.

5.1.2 Timer Accuracy

Timing accuracy described in this section refers to the length of time between the moment that the processor sets the timer-enable bit and the moment that the processor sets the timer-done bit. When writing the program, you should also account for the time required to turn on the output device once the processor sets the timer-done bit.

Timing accuracy depends on three factors:

- clock tolerance
- time base
- program scan time

The clock tolerance is ± 0.01%. Therefore, a timer could time out 0.01% early or late.

The time base that you select can also affect timer accuracy:

If you use this time base	The timer could time out up to
1.0 s	0.5 s earlier
0.1 s	0.1 s earlier
0.01 s	0.01 s earlier

When the processor executes a timer instruction, it maintains accurate time for a specific time interval:

If you use this time base	The timer remains accurate if examined every
1.0 s	1.0 s on average over two program scans (1.5 s max. for one scan)
0.1 s	0.3 s
0.01 s	0.15 s

In most cases, this time interval does not exceed the program scan time. However, if the program scan time does exceed the time interval, timer inaccuracy results. To prevent inaccuracy from this factor, program the same timer instruction repeatedly in the ladder program. Each time the processor executes the timer instruction, timer accuracy is maintained for the time interval.

WARNING: Program critical timers outside the MCR zone or jumped section of the ladder program to guard against invalid results. Refer to chapter 13 for detailed information on the jump and master-control-reset instructions.

5.2 Using Timer Instructions

The processor provides the following timer instructions:

- timer on-delay
- timer off-delay
- retentive timer on-delay
- timer one-shot

5.2.1 Timer On-delay (TON)

Required Parameters: Timer number, time base, timer preset value, and timer accumulated value.

Description: The timer-on-delay instruction turns outputs on or off after the timer has been on for a predetermined time interval. It starts accumulating time when the timer rung goes true and continues until one of the following conditions occur:

- Accumulated value equals the preset value.
- Rung condition goes false.
- Reset instruction resets the timer.

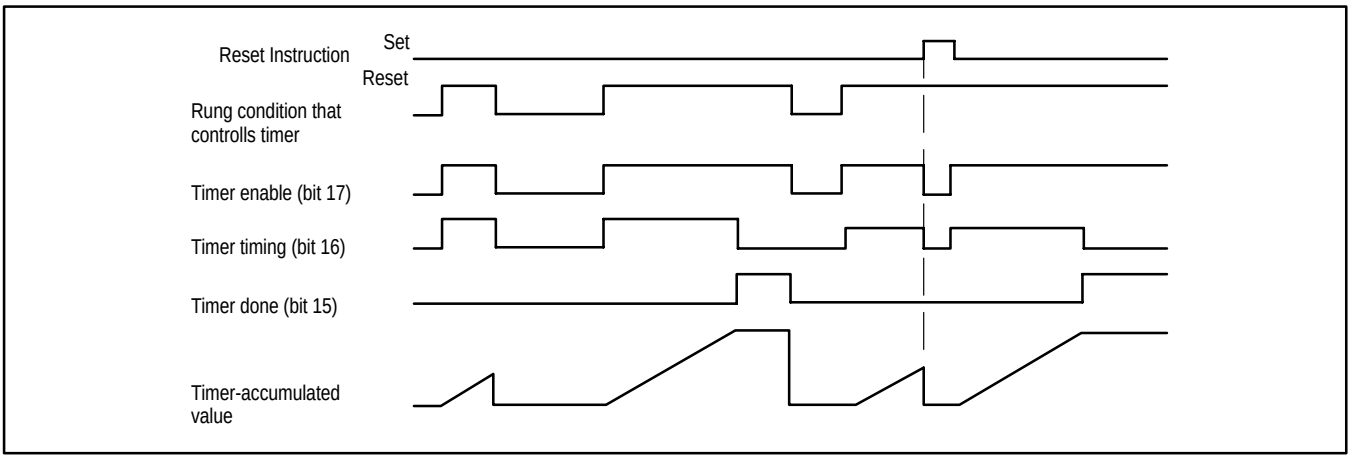
When the processor executes a rung containing a timer-on-delay instruction, status bits in the control word for the timer change states as follows (Figure 5.3):

- Bit 17, the timer enable bit (TE), is set when the timer rung goes true and remains set until the rung goes false, or a reset instruction resets the timer. When set, it shows that a timing operation is enabled.
- Bit 16, the timer timing bit (TT), is set when the rung goes true and remains set until the accumulated value equals the preset value, the rung condition goes false, or a reset instruction resets the timer. When set, it shows that a timing operation is in progress.

- Bit 15, the timer done bit (TD), is set when the accumulated value is equal to the preset value and remains set until the rung condition goes false, or a reset instruction resets the timer. When set, it shows that a timing operation is complete.

If the timer rung goes false, the timer resets itself by resetting the accumulated value to zero. This also occurs when you put the processor in the program load mode, or the processor loses power.

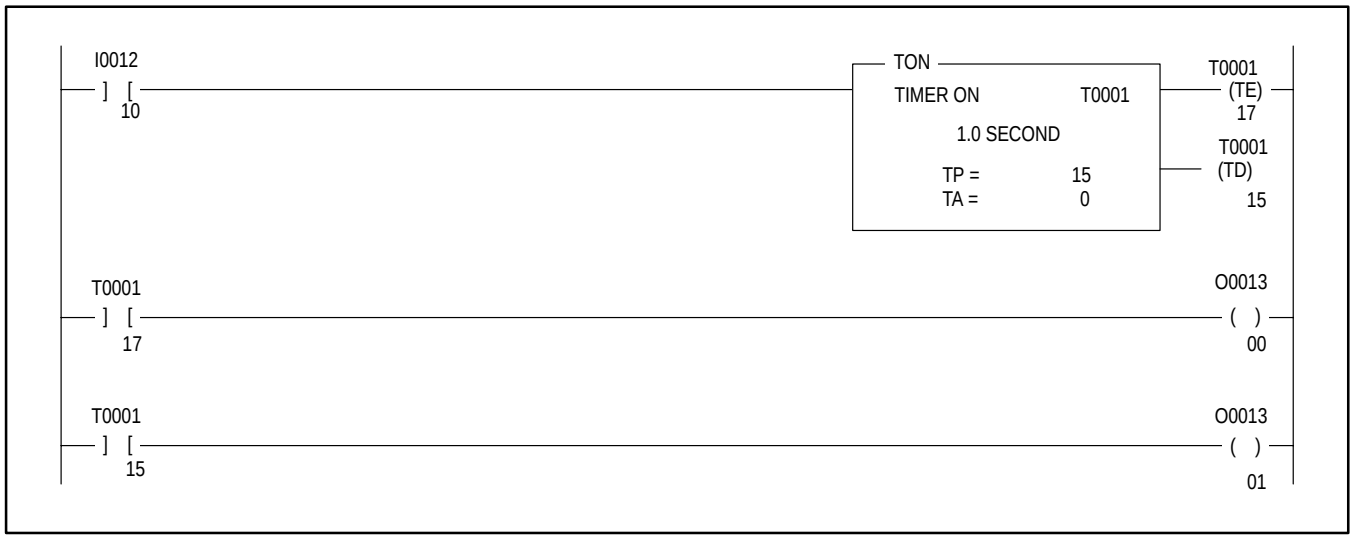
Figure 5.3
Timing Diagram for a Timer-on-delay Instruction



Example: Figure 5.4 shows rungs examining the enable and done bits of a timer on-delay instruction.

In the first rung, timer 1 begins to accumulate time when the processor sets input bit I0012/10. The time base is 1.0 second and the timer preset is 15. So timer 1 times for 15 seconds. In the second rung, output O0013/00 turns on when the processor sets the timer enable bit. In the third rung, output O0013/01 turns on when the processor sets the timer done bit.

Figure 5.4
Timing Rungs for a Timer-on-delay Instruction



5.2.2 Timer Off-delay (TOF)

Required Parameters: Timer number, time base, timer preset value, and timer accumulated value.

Description: The timer-off-delay instruction turns outputs on or off after the timer has been off for a predetermined time interval. It starts accumulating when the timer rung goes false and continues timing until one of the following conditions occur:

- Accumulated value equals the preset value.
- Rung condition goes true.
- Reset instruction resets the timer.

When the processor executes a rung containing a timer off-delay instruction, status bits in the control word for the timer change states as follows (Figure 5.5):

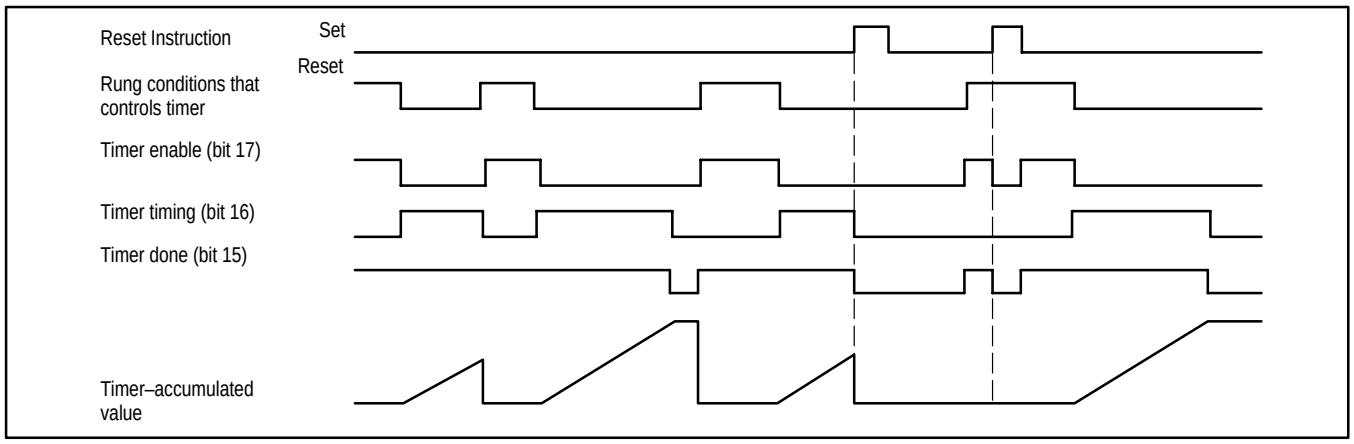
- Bit 17, the timer-enable bit (TE), is reset when the timer rung goes false and remains reset until the rung goes true. When reset, it shows that a timing operation has been enabled. If you use the reset instruction, the processor resets this bit.

- Bit 16, the timer-timing bit (TT), is set when the rung goes false and remains set until the accumulated value equals the preset value, the rung condition goes true or a reset instruction resets the timer. When set, it shows that a timing operation is in progress.

- Bit 15, the timer-done bit (TD), is reset when the accumulated value equals the preset value and remains reset until the rung condition goes true. When reset, it shows that a timing operation is complete. If you use the reset instruction, the processor resets this bit.

When the timer rung goes false initially, the timer resets itself by setting the accumulated value equal to the preset value. This also occurs when you put the processor in the program load mode, or the processor loses power.

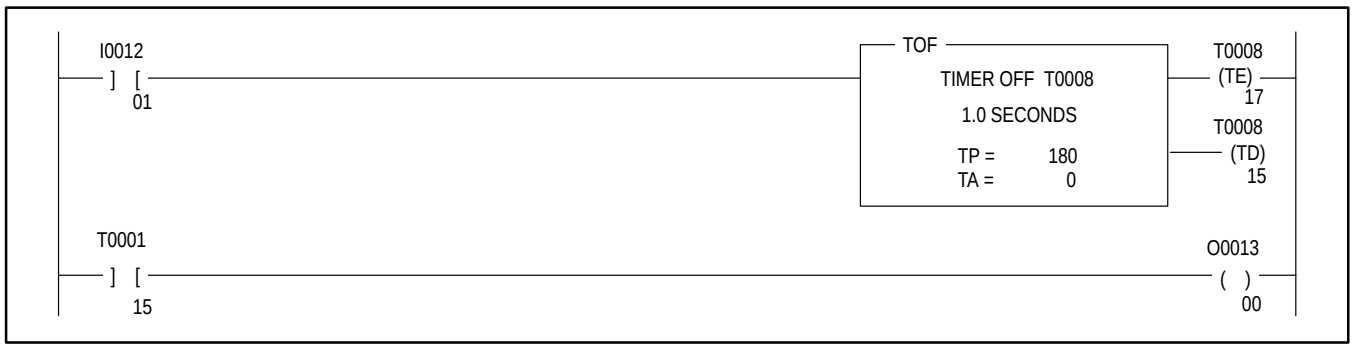
Figure 5.5
Timing Diagram for a Timer-off-delay Instruction



Example: Figure 5.6 shows rungs examining a control bit of a timer-off-delay instruction.

In the first rung, timer 8 begins timing when the input instruction is false. The time base is 1.0 s and the timer preset is 180. So timer 1 times for 180s or 3 min. In the second rung, output O0013/00 turns on when the processor sets the timer-done bit.

Figure 5.6
Example Rungs for a Timer-off-delay Instruction



5.2.3 Retentive Timer On-delay (RTO)

Required Parameters: Timer number, time base, timer preset value, and timer accumulated value.

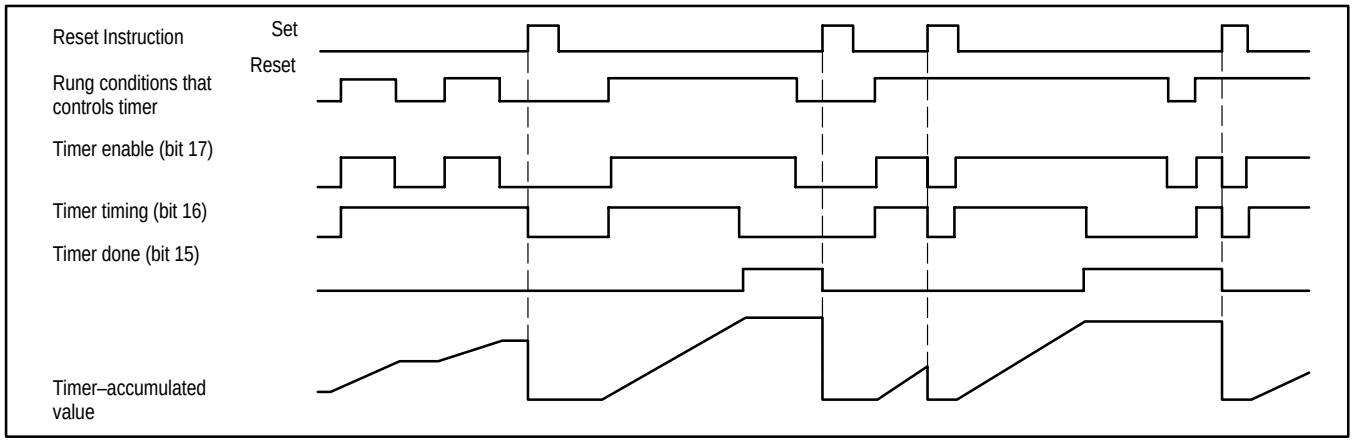
Description: The retentive-timer-on-delay instruction turns outputs on or off after the timer has been on for a predetermined time interval. The main difference between the retentive-timer-on-delay and the timer-on-delay instructions is that the retentive-timer-on-delay allows the timer to stop and start without resetting.

When the processor executes a rung containing a retentive timer, status bits in the control word for the timer change states as follows (Figure 5.7):

- Bit 17, the timer-enable bit (TE), is set when the timer rung goes true and remains set until the rung goes false, or a reset instruction resets the timer. When set, it shows that a timing operation has been enabled.
- Bit 16, the timer-timing bit (TT), is set when the rung goes true and remains set until the accumulated value equals the preset value, or a reset instruction resets the timer. When set, it shows that a timing operation is in progress.
- Bit 15, the timer-done bit (TD), is set when the accumulated value equals the preset value and can only be reset by a reset instruction. When set, it shows that a timing operation is complete.

The retentive-timer-on-delay instruction does not reset itself when the rung alternately goes true and false. This also occurs when you put the processor in program load mode or the processor loses power. To reset it, you use the reset instruction.

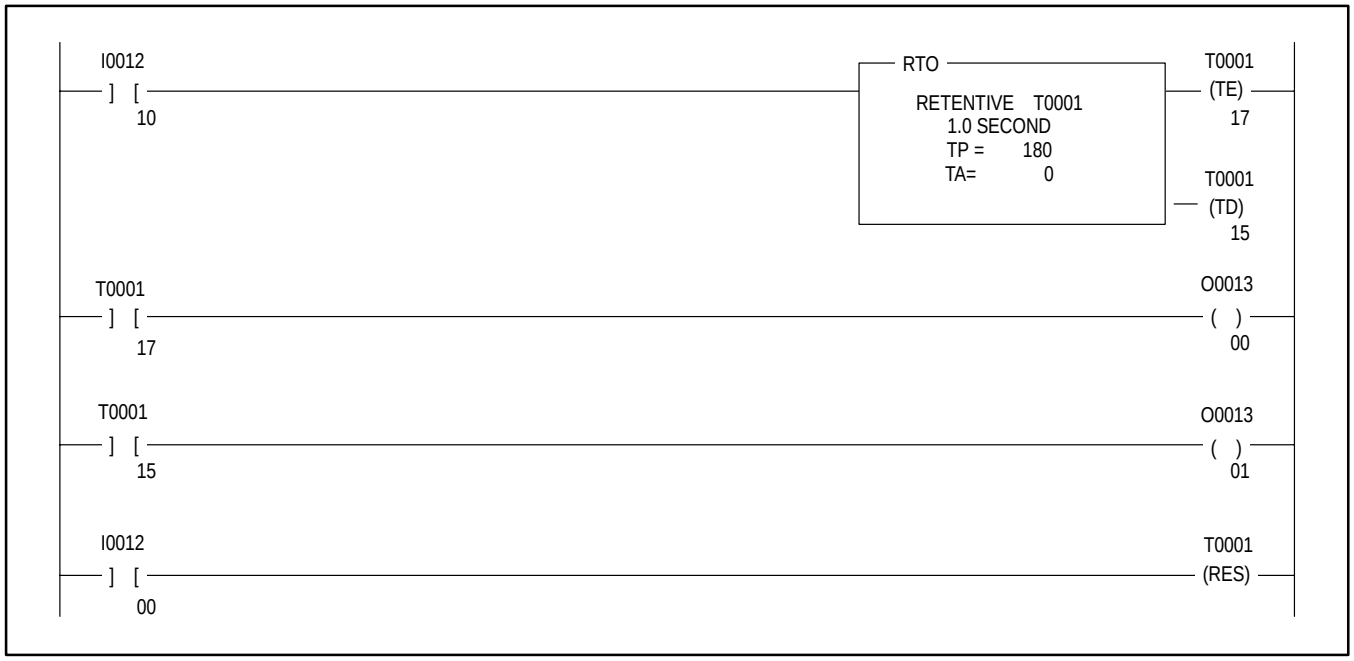
Figure 5.7
Timing Diagram for a Retentive-timer-on-delay Instruction



Example: Figure 5.8 shows rungs examining control bits of retentive-timer-on-delay instruction.

When a false-to-true rung transition occurs in the first rung, the processor sets the timer enable bit to start timer 1. The time base is 1.0 s and the timer preset is 180. So timer 1 times for 180 s or 3 min. In the second rung, output O0013/00 turns on when the processor sets the timer-enable bit. In the third rung, output O0013/01 turns on when the processor sets the timer-done bit. In the fourth rung, timer 1 resets when the processor sets input bit I0012/00.

Figure 5.8
Example Rungs for a Retentive-timer-on-delay Instruction



5.2.4 Timer One-shot (TOS)

Required Parameters: Timer number, time base, timer preset value, and timer accumulated value.

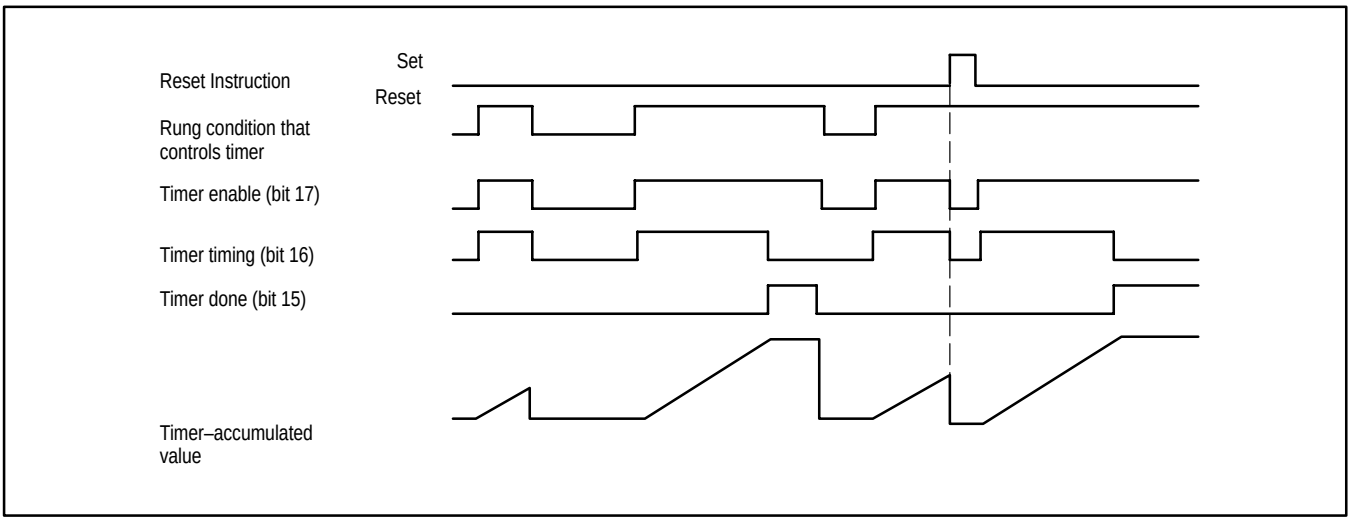
Description: The timer-one-shot instruction turns on an output based on a monitored input. It operates like the timer-on-delay instruction. The difference is that the timer-timing bit (TT) appears in the instruction format instead of the timer-done bit (TD).

When the processor executes a rung containing a timer-one-shot instruction, status bits in the control word for the timer change states as follows (Figure 5.9):

- Bit 17, the timer-enable bit (TE), is set when the timer rung goes true and remains set until the rung goes false, or a reset instruction resets the timer. When set, it shows that a timing operation is enabled.
- Bit 16, the timer-timing bit (TT), is set when the rung goes true and remains set until the accumulated value equals the preset value, the rung condition goes false, or a reset instruction resets the timer. When set, it shows that a timing operation is in progress.
- Bit 15, the timer-done (TD), is set when the accumulated value is equal to the preset value and remains set until the rung condition goes false, or a reset instruction resets the timer. When set, it shows that a timing operation is complete.

When the timer rung goes false, the timer resets itself by resetting the accumulated value to zero. This also occurs when you put the processor in the program load mode, or the processor loses power.

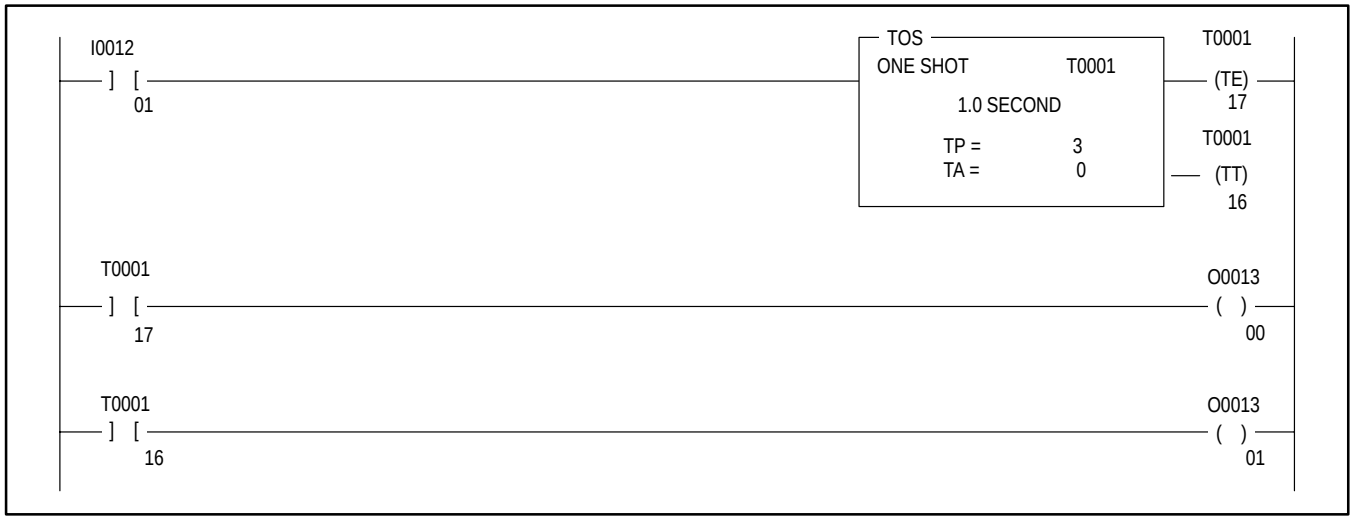
Figure 5.9
Timing Diagram for a Timer-one-shot Instruction



Example: Figure 5.10 shows rungs examining control bits of a timer one-shot instruction.

In the first rung, timer 1 begins timing when the input I0012/01 is on. The time base is set at 1.0 second and the timer preset is set at 3. So timer times for 3 seconds. In the second rung, output O0013/00 turns on when the processor sets the timer enable bit. In the third rung, output O0013/01 turns on when the processor sets the timer timing bit.

Figure 5.10
Example Rungs for a Timer-one-shot Instruction



5.3 Using Counters

You can use counters in applications where you want to keep track of a number of events or control outputs based on a count value. Counter instructions are output instructions that increments or decrements an accumulated value base on a false-to-true rung transition. Each counter instruction has two 5-digit values associated with it:

- **present value** – value that the counter must reach before the processor turns on or off its outputs. You specify this value when loading the instruction onto a ladder rung. When the accumulated value equals the preset value, the processor sets a status bit. You can use this bit to control an output device.
- **accumulated value** – value that represents the current count. The processor generates this value when executing the ladder program.

The range for these values is -32,768 to 32, 767 and the processor stores them in binary format. If you are using a move instruction to transfer data to or from these words, the data must follow this format (see chapter 6).

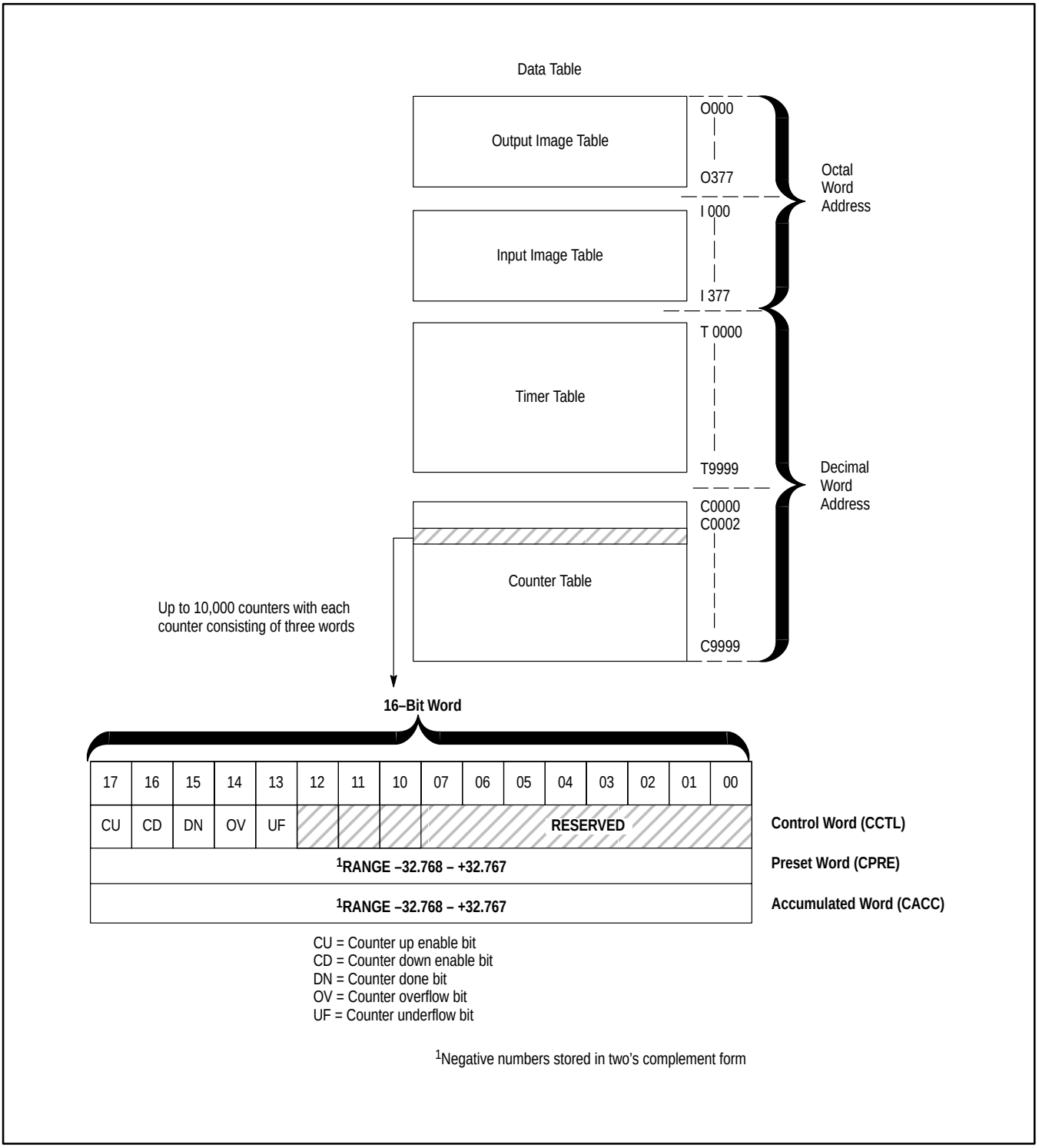
WARNING: Do not use a counter assigned to a file instruction (see chapter 7) for any other purpose. Unexpected operation could result in damage to equipment and/or injury to personnel.

Figure 5.11 shows the counter section of the data table. You can program up to 10,000 counters (C0 to C9999) with each counter instruction requiring three words in the data table:

Control word (CCTL) contains control bits that reflect the status of the counter instruction. You can examine these control bits in the ladder program:

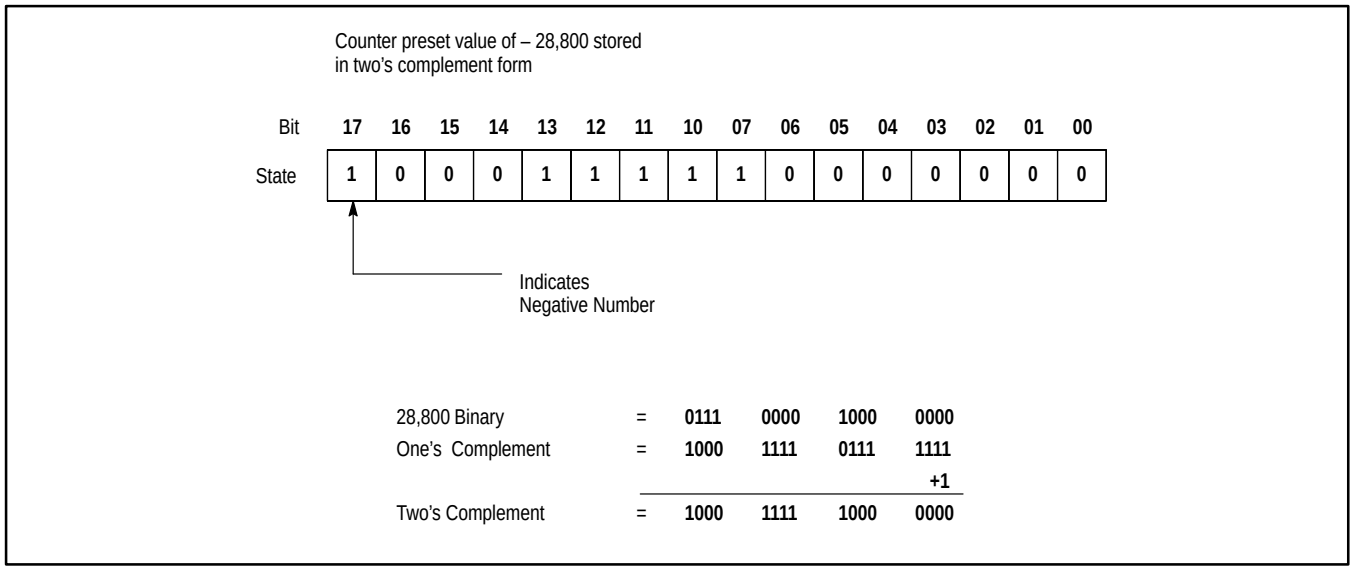
- Counter-up enable - bit 17 (CU) shows that a count up operation has been enabled.
- Counter-down enable - bit 16 (CD) shows that a count down operation has been enabled.
- Counter-done - bit 15 (DN) shows that a counter operation is complete.
- Counter-overflow - bit 14 (OV) shows that a counter has reached the upper limit of 32,767.
- Counter-underflow - bit 13 (UF) shows that a counter has reached the lower limit of -32,768.

Figure 5.11
Memory Storage for Counters



Preset value word (CPRE) contains an integer representing the target value for the counter instruction. This value can range from -32,768 to 32,767. The processor stores negative numbers in two's compliment form (Figure 5.12).

Figure 5.12
The Processor Stores Negative Preset and Accumulated Values in Two's Complement Form



Accumulated value word (CACC) contains a value representing the current count of events that have occurred for counter instruction since the last reset. This value can range from -32,768 to 32,767. The processor stores negative numbers in two's complement form (Figure 5.12).

WARNING: Program critical counters outside the MCR zone or jumped section of the ladder program to guard against invalid results. Refer to chapter 13 for detailed information on the jump and master-control-reset instructions.

5.4 Using Counter Instructions

The processor provides the following counter instructions:

- counter up
- counter down

5.4.1 Counter Up (CTU)

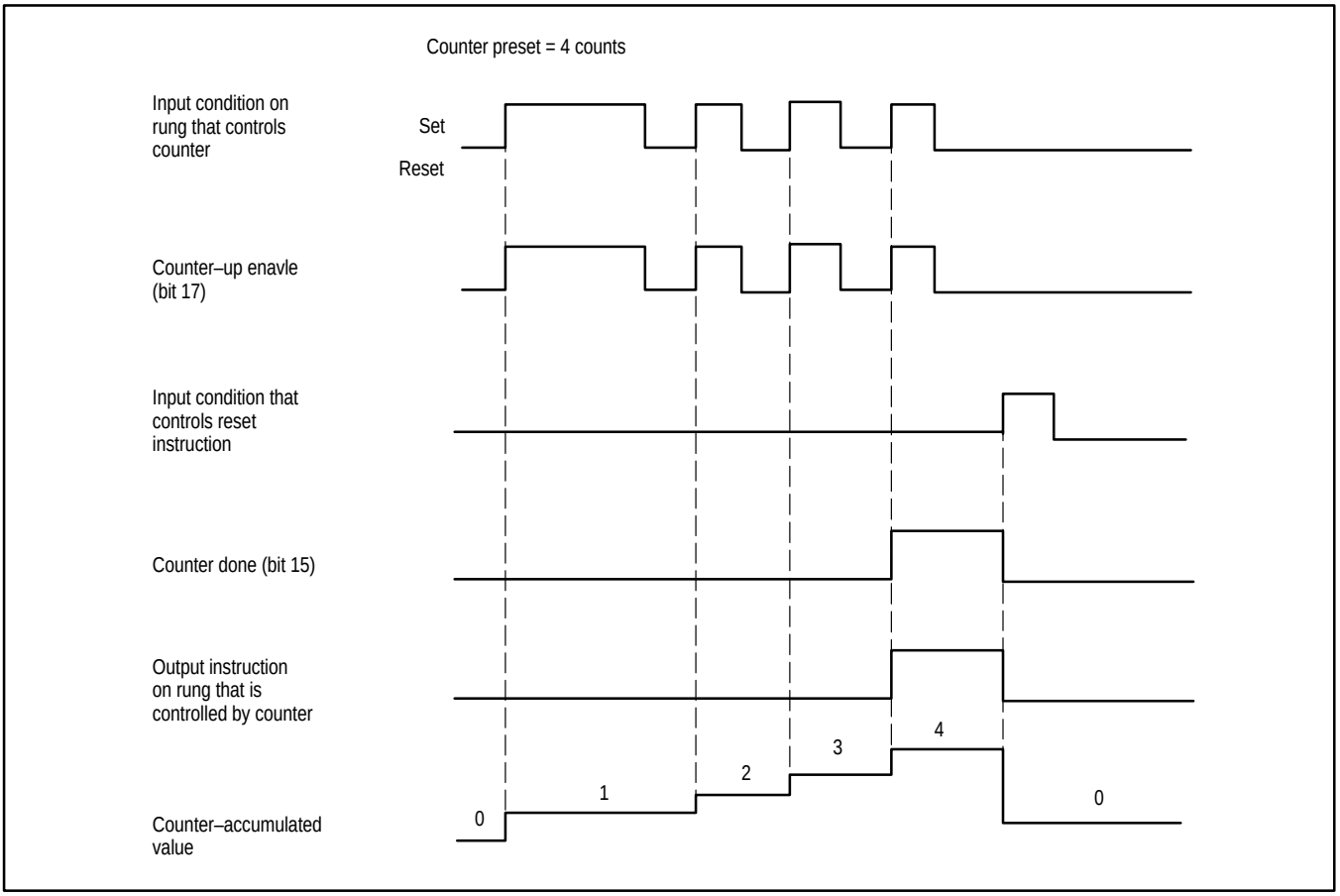
Required Parameters: Counter number, counter preset value, and counter accumulated value.

Description: When the processor executes a rung containing a counter-up instruction, bits in the control word change states as follows (Figure 5.13):

- Bit 17, the counter-up-enable bit (CU), is set when the counter rung goes true. The processor resets this bit when the rung goes false. When set, it shows that a count operation has been enabled.
- Bit 15, the done bit (DN), is set when the accumulated value is equal to or greater than the present value.
- Bit 14, the counter-overflow bit (OV), is set when the accumulated value increments past its upper limit of 32,767.

If the counter rung goes false, the counter does not reset itself. This also occurs if you put the processor in the program load mode, or the processor loses power. To reset the counter-up instruction, you use the reset instruction.

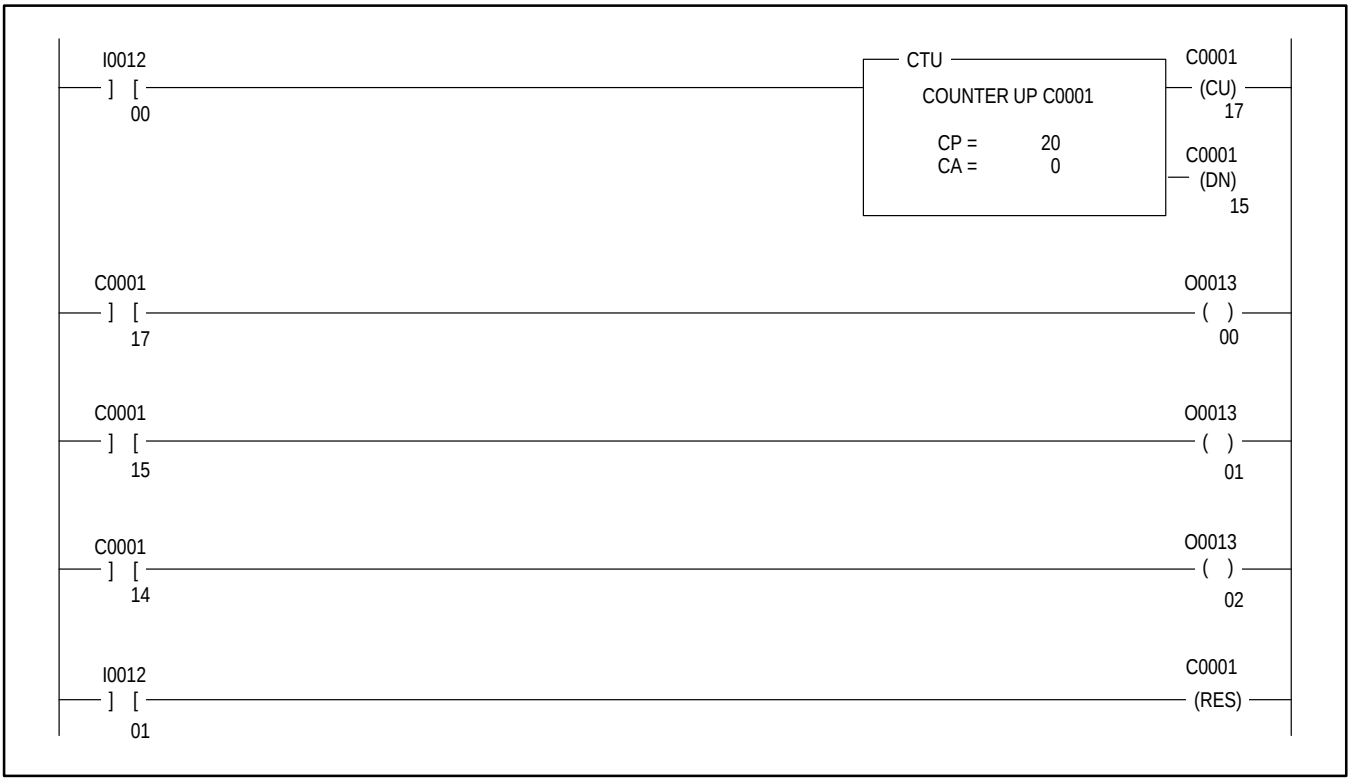
Figure 5.13
Timing Diagram for a Counter-up Instruction



Example: Figure 5.14 shows rungs examining control bits of a counter-up instruction.

In the first rung, when a false-to-true rung transition occurs, the processor sets the counter-enable bit (CU), and counter 1 increments. The counter preset is 20. Each time the input toggles from off to on, the counter increments. So when counter 1 reaches a count of 20, the processor sets bit 15, counter done (DN). In the second rung, output O0013/00 turns on when the processor sets the counter-enable bit. In the third rung, output O0013/01 turns on when the processor sets the counter-done bit. In the fourth rung, output O0013/02 turns on when the processor sets the overflow bit. In the fifth rung, when input I0012/01 turns on, the processor resets the counter.

Figure 5.14
Example Rungs for a Counter-up Instruction



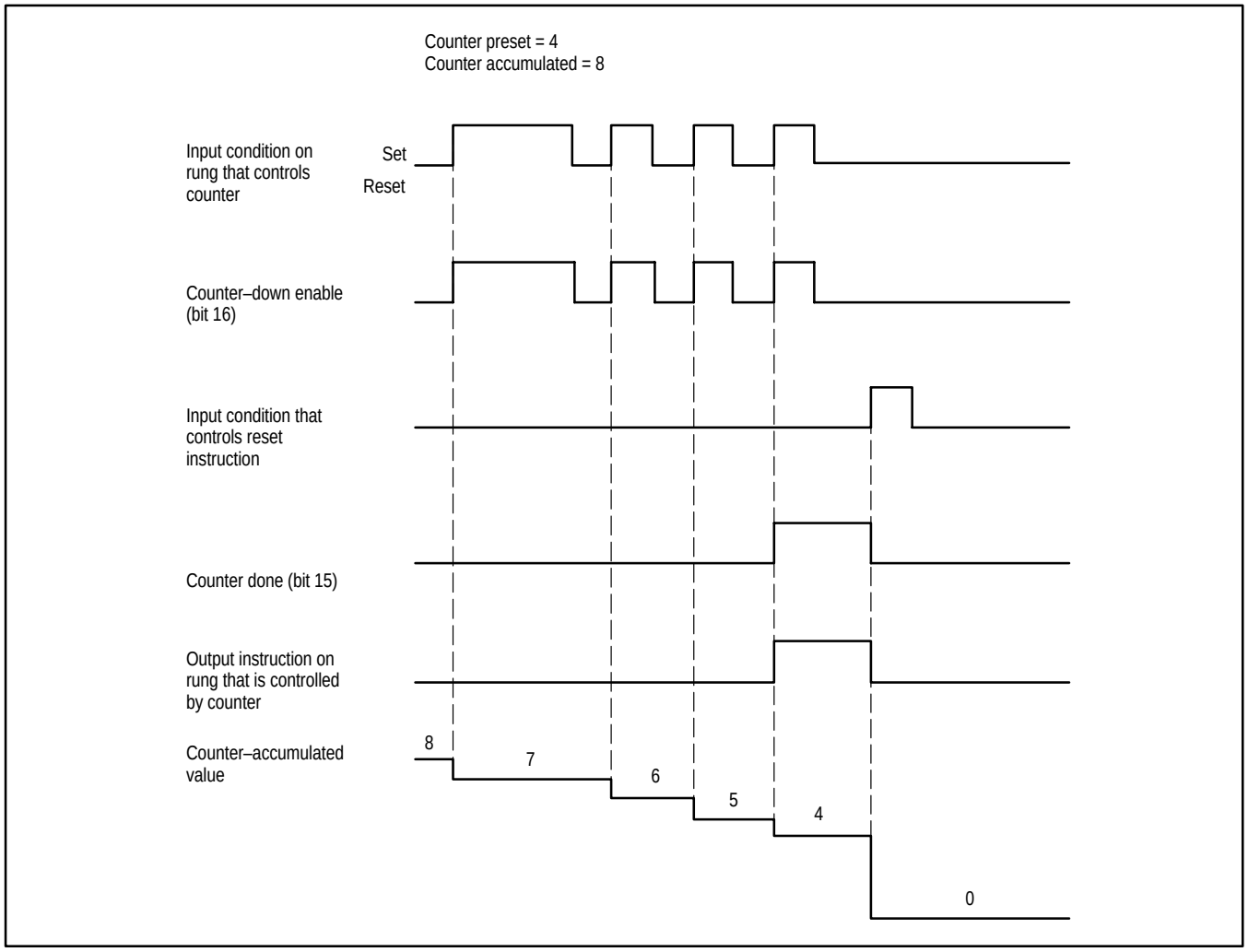
5.4.2 Counter Down (CTD)

Required Parameters: Counter number, counter preset value, and counter accumulated value.

Description: When the processor executes a rung containing a counter-down instruction, status bits in the control word change states as follows (Figure 5.15):

- Bit 16, the counter-down-enable bit (CD), is set when the counter rung goes true. The processor resets this bit when the rung goes false. When set, it shows that a count operation has been enabled.
- Bit 15, the done bit (DN), is set when the accumulated value is equal to or greater than the preset value.
- Bit 13, the counter-underflow (UF), is set when the accumulated value decrements below its lower limit of -32,768.

Figure 5.15
Timing Diagram for a Counter-down Instruction



If the counter rung goes false, the counter does not reset itself. This also occurs if you put the processor in the program load mode, or the processor loses power. To reset the counter down instruction, you use the reset instruction.

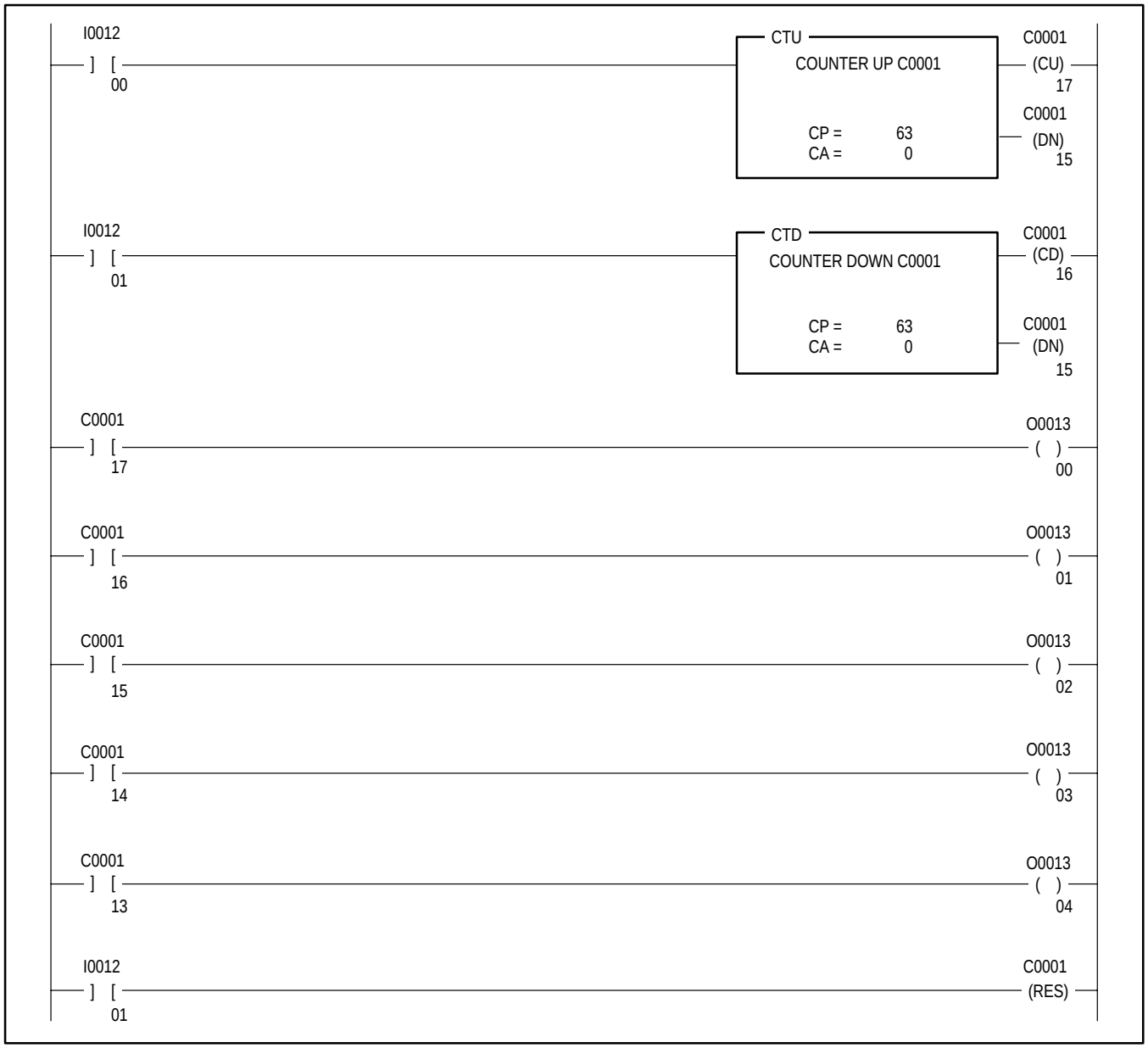
Example: Figure 5.16 shows rungs examining control bits of counter-up and down instructions.

In the first rung, when a false-to-true rung transition occurs, the processor sets the counter up enable bit (CU) and counter 1 increments. The counter preset is 63. Each time the input bit toggles, counter 1 increments. So when counter 1 reaches a count of 63, the processor sets bit 15, counter done (DN). In the second rung, when the rung goes true, the processor sets the counter done bit (CD), and counter 1 decrements. The counter preset is 63.

Each time the input bit toggles, the counter decrements. So when counter 1 counts down to a count of 20, the processor sets bit 15, counter done (DN).

In the third rung, output O0013/00 turns on when the processor sets the counter-down-enable bit. In the fourth rung, output O0013/02 turns on when the processor sets the counter-done bit. In the fifth rung, output O0013/01 turns on when the processor sets the underflow bit. In the sixth rung, when the processor sets input bit I0012/01, counter 1 resets.

Figure 5.16
Example Rungs for Up- and Down-counter Instructions



5.5 Resetting Timers and Counter s (RES)

Required Parameters: Timer or counter number.

Description: The reset instruction is an output instruction that you enter on a separate rung to reset a timer or counter. Like any other output instruction, the reset instruction executes only when the rung is true.

When executing a reset instruction for a	The processor resets the
timer	accumulated-value, timer-done, and timer-timing bits
counter	accumulated-value, overflow, underflow, and counter-done bits

Example: Refer to the retentive-timer-on-delay and counter instruction sections.

5.6 Cascading Timers and Counters

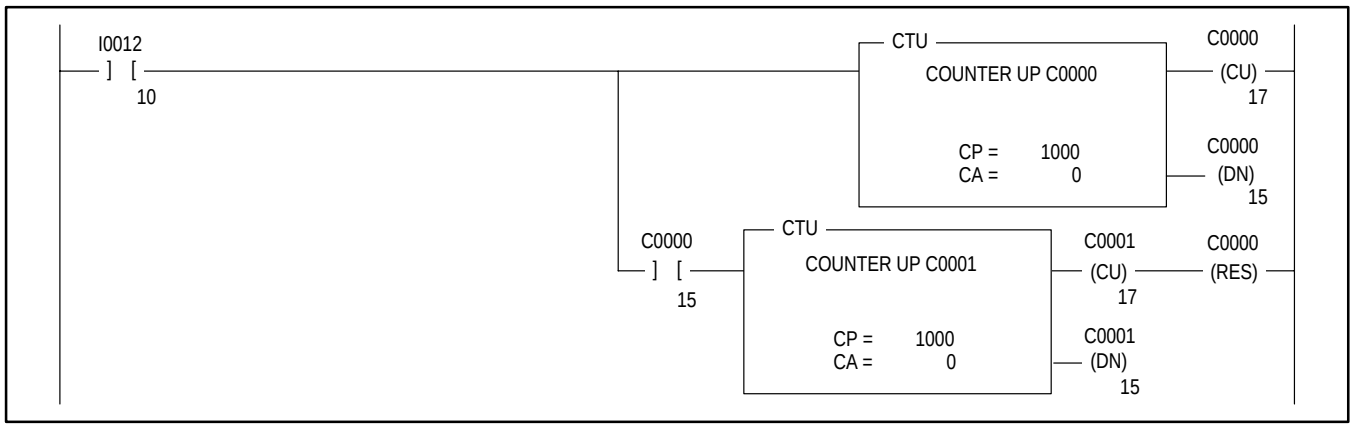
The range for timer and counter instructions is 0 to 65,353 and -32,768 to 32,767 respectively. By cascading two or more timers or counters, you can multiply the range of these instructions.

To cascade timers or counters, each timer or counter is assigned a different word address. The first timer or counter executes and the second timer or counter operates on the done bit of the first timer or counter. That is, the second timer or counter starts each time the first timer or counter reaches its preset value and sets its done bit.

Figure 5.17 shows an example for cascading counter up instructions.

As input I0012/10 toggles, counter 0 begins counting up. When the accumulated value reaches the preset value 1000, the processor sets the done bit. The done bit is the input condition that initiates counter 1. The reset instruction resets the counter 0.

Figure 5.17
Example Rung for Cascading Counter-up Instructions



Chapter 6

Using Data-manipulation Instructions

6.0 Chapter Objectives

After reading this chapter, you should understand how the processor uses data-manipulation instructions to transfer, compare, or compute data in the data table.

6.1 Data Manipulation

Data-manipulation instructions transfer, compare, or compute arithmetic or logic functions. Some sources of data to be manipulated include:

- timer preset and accumulated values
- counter preset and accumulated values
- thumbwheel input values
- analog input values
- values you enter through the keyboard
- encoder inputs

The processor stores the data that you want to manipulate in the various sections of the data table. You can use data-manipulation instructions to manipulate:

- words or portions of words (bits) between data table sections.
- entire files or portions of files (words) between data table sections.

A file is a group of consecutive words that can be accessed as a unit. The data table is composed of files that contain up to 10,000 data words per file. Files are addressed 0 to 999.

Figure 6.1 shows the data table map with file organization for the various data table sections. We describe how to use files in chapter 7. The examples given in this chapter show you how data-manipulation instructions work on words between data table sections.

To address words in the data table sections, use the prefix W followed by the section specifier (Table 6.A). For example, the word address of a floating-point value in word 198 is WF198.

Figure 6.1
Data Table Map

Section Number	Title	Maximum Size	Address Range
1	Output Image Table	4,096 values	$O0000_8$ to $O0377_8$
2	Input Image Table	4,096 values	$I0000_8$ to $I0377_8$
3	Timer Table (3 words/timer)	10,000 timers	T0 to T9999
4	Counter Table (3 words/counter)	10,000 counters	C0 to C9999
5	Integer Table (1 word/value)	10,000 values	N000:0000 to N000:9999
6	Floating Point Table (2 words/value)	10,000 values	F000:0000 to F000:9999
7	Decimal Table (1 word/ 4 BCD values)	10,000 values	D000:0000 to D000:9999
8	Binary Table (1 word/value)	10,000 values	B000:0000 to B000:9999
9	ASCII Table (2 characters/word)	20,000 characters	A000:0000 to A000:9999
10	High–order–integer Table (2 words/value)	10,000 values	H000:0000 to H000:9999
12	Pointer Table	10,000 addresses	P000:0000 to P000:9999
13	Status Table	10,000 values	S000:0000 to S000:9999

Table 6.A
Section Specifiers, Data Types, and Acceptable Ranges for Values Stored in the Data Table

Data Table Section	Section Specifier	Type of Data Stored in Section	Range	
			Low Limit	High Limit
Output image	O	Unsigned binary	0	65,535
Input image	I	Unsigned binary	0	65,535
Timer	T	Unsigned binary	0	65,535
Counter	C	Binary[1]	-32,768	32,767
Integer	N	Binary[1]	-32,768	32,767
Floating point	F	Floating point	±2.939 E-39	±1.701 E+38
Decimal	D	Binary coded decimal	0	9,999
Binary	B	Unsigned binary	0	65,535
ASCII[2]	A	Unsigned binary	– – – – –	– – – – –
High order integer	H	Binary[1]	-2,147,483,648	2,147,483,647
Pointer	P	Unsigned binary[3]	– – – – –	– – – – –
Status	S	Unsigned binary[3]	– – – – –	– – – – –

[1]The processor stores positive numbers in straight binary and negative numbers in two's complement form.

[2]The ASCII table can store ASCII characters as defined by ASCII (ANSI X3.4).

[3]The processor treats data in the pointer and status sections as unsigned binary, although these sections are intended to store non-numeric data.

We group the data-manipulation instructions into four types:

- data-transfer instructions
- data-comparison instructions
- arithmetic instructions
- logic instructions

We describe these types in the rest of this chapter.

6.2 Data-transfer Instructions

Data-transfer instructions move one word of data from one location to another in the same or different data table sections. By using data-transfer instructions, you can:

- copy timer or counter control words, preset or accumulated values
- copy one word to another through a mask
- copy data from a read-only data table section, such as system status into another

Data-transfer instructions are output instructions that the processor executes when a rung is true.

In programming data-transfer instructions, you specify addresses that tell the processor the source and destination for data that you want to move:

- Source address (A) tells the processor where to read the data.
- Destination address (R) tells the processor where to transfer the data.

When the processor executes a rung containing a data-transfer instruction:

If the rung is	Then the processor
true	executes the instruction and copies data from the source to the destination
false	does not copy the data; the destination retains its last value

If you program the transfer of data between data table sections with different data types, the processor automatically converts the data into the proper type. For example, if the processor moves a word from the binary to the decimal section, it automatically converts binary to BCD.

However, the processor does not convert the data to and from the input or output sections. To convert the data, you can first transfer the data from the input or output section into another section. For example, if the data is BCD, you can transfer the data into the decimal section. Then transfer the data from the decimal section to the desired section.

If a value in the source or destination address is not a storable data type or not within the limits of the corresponding data table section, the processor sets the instruction fault (bit 17) in status file 0, word 0 and either the overflow (bit 13), underflow (bit 12), or conversion fault (bit 10) bits in status file 0, word 0 (see chapter 14).

You can use the following data-transfer instructions:

- move
- move with mask
- move status

6.2.1 Move (MOV)

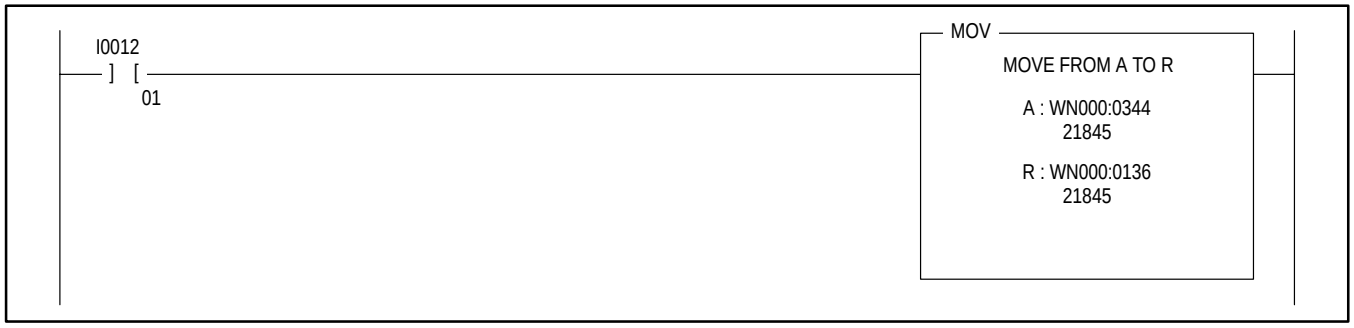
Required Parameters: Source (A) and destination (R) addresses

Description: When a rung containing a move instruction is true, the processor copies the information from the source to the destination. To load values into word locations, you can enter the data upon loading the instruction or by using the data monitor. Refer to the PLC-3 Industrial Terminal User's Manual (publication 1770-6.5.15) for detailed information on these methods.

Example: Figure 6.2 shows a rung containing a move instruction.

If the rung is true, the processor copies the data from the source (integer word 344) to the destination (integer word 136).

Figure 6.2
Example Rung for a Move Instruction



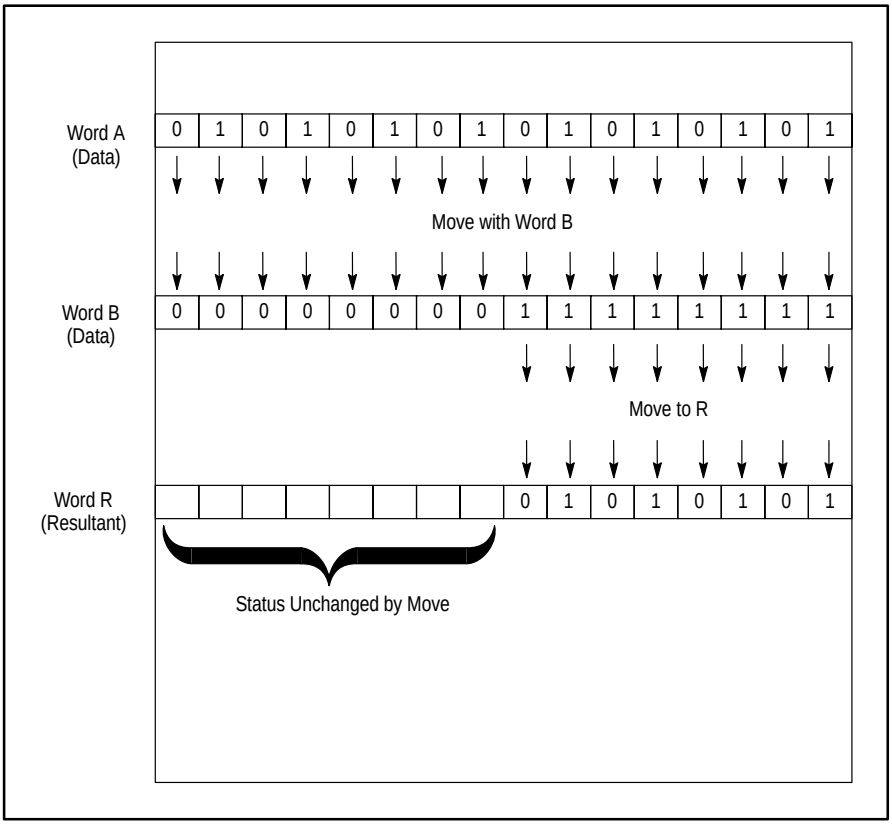
6.2.2 Move with Mask (MVM)

Required Parameters: Source (A), mask (B), an destination (R) addresses.

Description: When a rung containing a move-with-mask instruction is true, the processor copies the information from the source through the mask to the destination.

Only bits in the source that correspond to set bits in the mask move to the destination (Figure 6.3). The most likely use of the move-with-mask instruction is with the input image, output image, binary, and decimal tables. Data formats for values in the source and destination should be the same. Values for the mask should be binary.

Figure 6.3
Move-with-mask Instruction Operation

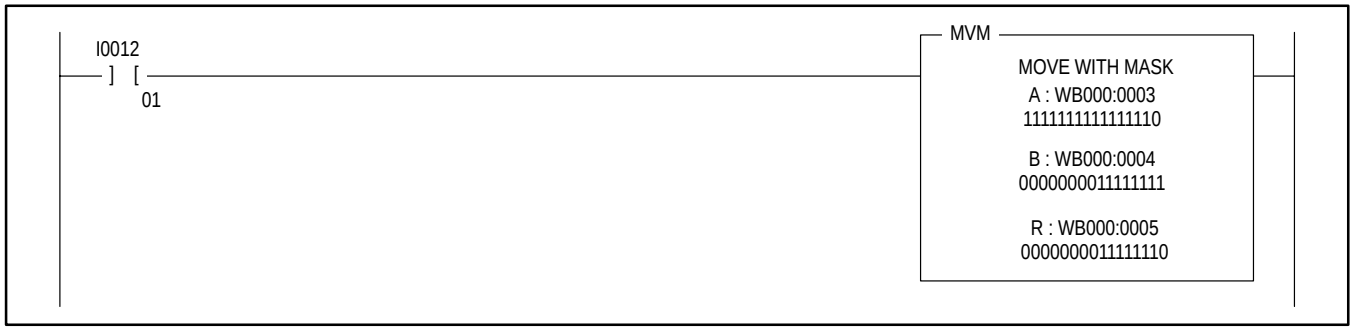


To pass data through the mask, you must set bits in the mask word. To load values into source or mask-word locations, you can enter the data upon loading the instruction or by using the data monitor. Refer to the PLC-3 Industrial Terminal User's Manual (publication 1770-6.5.15) for detailed information on these methods.

Example: Figure 6.4 shows a rung containing a move-with-mask instruction.

If the rung is true, the processor copies the bits from the source (binary word 3) that correspond to set bits in the mask word (binary word 4), to the destination (binary word 5).

Figure 6.4
Example Rung for a Move-with-mask Instruction



6.2.3 Move Status (MVS)

Required Parameters: Source (extended address for memory location) and destination addresses.

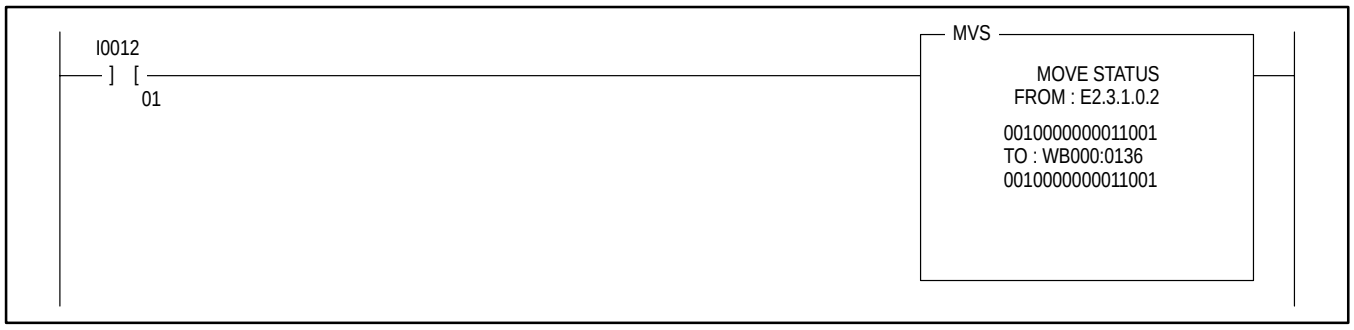
Description: When a rung containing a move-status instruction is true, the processor copies the information from the source to the destination. With this instruction, you can move information into the data table from areas such as system status or module status.

In specifying an address outside the data table, you enter an extended address. Refer to chapter 14 for detailed information on extended addressing.

Example: Figure 6.5 shows a rung containing a move instruction.

If the rung is true, the processor copies the data from the module status area to binary word 136.

Figure 6.5
Example Rung for a Move-status Instruction



6.3 Data-comparison Instructions

Data-comparison instructions are input instructions that determine whether or not an output turns on. These instructions operate on integer and/or floating-point data and can compare values stored in any data table section. The upper and lower limits of the data being compared depend on the section where data are stored.

When data-comparison instructions compare data between data table sections having different data types, the processor automatically converts the data to the proper data type. For example, if the processor compares binary value 1101 (13 in binary) to BCD value 13, the processor considers both values equal, even through they are stored in different data types.

You can use the following data-comparison instructions:

- equal to, not equal to
- greater than, greater than or equal to
- less than, less than or equal to
- limit

6.3.1 Equal To (EQU)

Required Parameters: Sources (A and B) addresses.

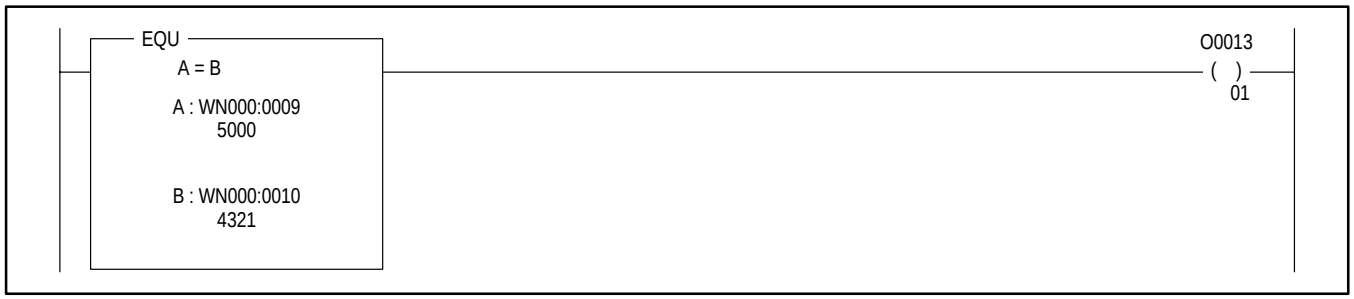
Description: The equal-to instruction tells the processor to compare a value at source (A) with the value at source (B):

If source	Then the rung is
A=B	true
A≠B	false

Example: Figure 6.6 shows a rung containing an equal-to instruction.

The processor compares the source values in integer words 9 and 10. If the values are equal (A = B), the processor turns on output O0013/01.

Figure 6.6
Example Rung for an Equal-to Instruction



6.3.2 Not Equal To (NEQ)

Required Parameters: Sources (A and B) addresses.

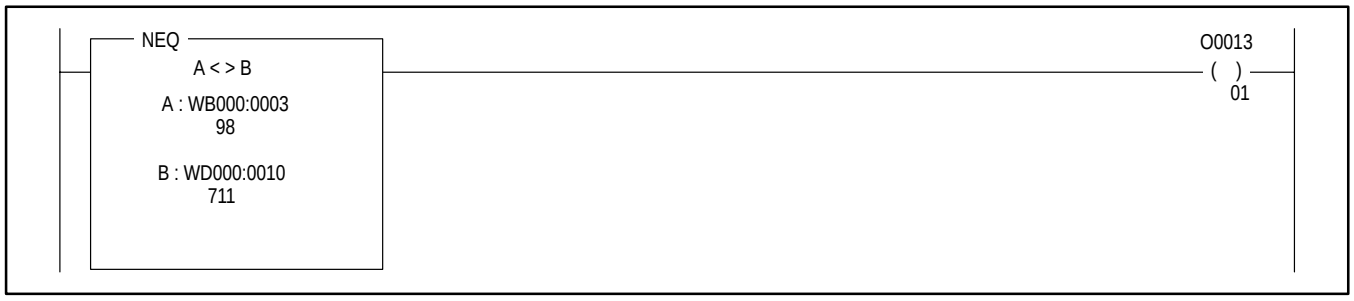
Description: The not-equal-to instruction tells the processor to compare a value at source (A) with the value at source (B):

If source	Then the rung is
A≠B	true
A=B	false

Example: Figure 6.7 shows a rung containing a not-equal-to instruction

The processor compares the source value in binary word 3 to the source value in decimal word 10. If the values are not equal (A ≠ B), the processor turns on output O0013/01.

Figure 6.7
Example Rung for a Not-equal-to Instruction



6.3.3 Greater Than (GRT)

Required Parameters: Sources (A and B) addresses.

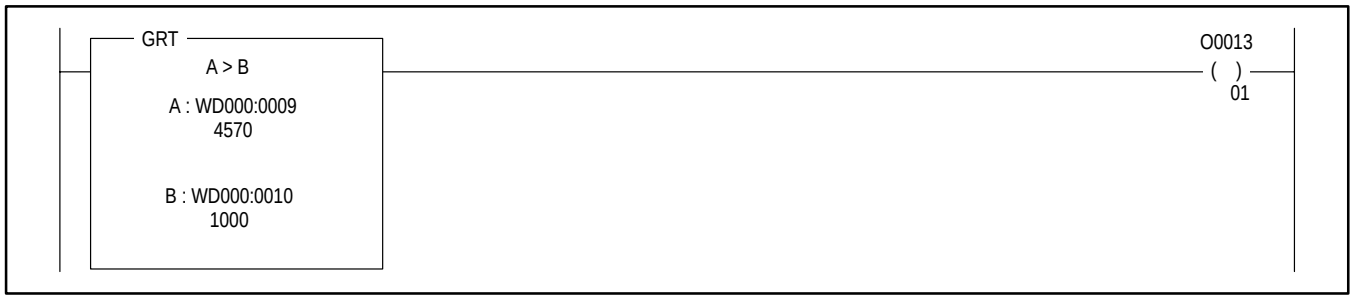
Description: The greater-than instruction tells the processor to compare a value at source (A) with the value at source (B):

If source	Then the rung is
A>B	true
A≤B	false

Example: Figure 6.8 shows a rung containing a greater-than instruction.

The processor compares the source values in decimal words 9 and 10. If the value in decimal word 9 is greater than the value in decimal word 10 (A > B), the processor turns on output O0013/01.

Figure 6.8
Example Rung for a Greater-than Instruction



6.3.4 Greater Than or Equal To (GEQ)

Required parameters: Sources (A and B) addresses.

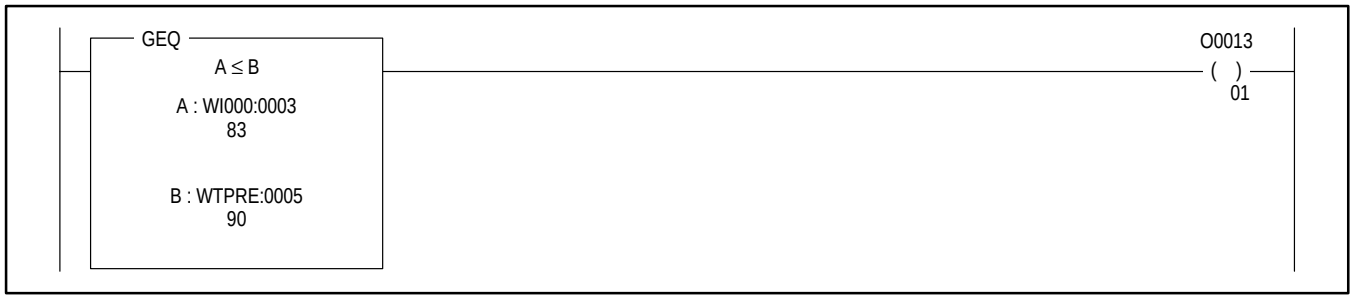
Description: The greater-than or equal-to instruction tells the processor to compare a value in source (A) with the value at source (B):

If source	Then the rung is
A≥B	true
A<B	false

Example: Figure 6.9 shows a rung containing a greater-than or equal-to instruction.

The processor compares the source value in input word 3 to the source value in the timer preset word for timer 5. If the value in input word 3 is greater than or equal to the value in the timer preset word for timer 5 (A ≥ B), the processor turns on output O0013/01.

Figure 6.9
Example Rung for a Greater-than-or-equal-to Instruction



6.3.5 Less Than (LES)

Required Parameters: Sources (A and B) addresses.

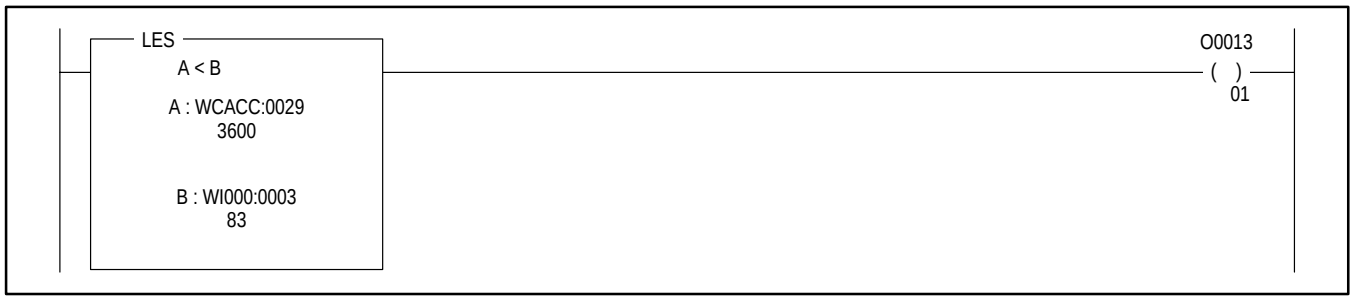
Description: The less-than instruction tells the processor to compare a value at source (A) with the value at source (B):

If source	Then the rung is
A<B	true
A≥B	false

Example: Figure 6.10 shows a rung containing a less-than instruction.

The processor compares the source value in the accumulated value word for counter 29 to the source value in input word 3. If the value in the accumulated value word for counter 29 is less than the value in input word 3 (A < B), the processor turns on output O0013/01.

Figure 6.10
Example Rung for a Less-than Instruction



6.3.6 Less Than or Equal To (LEQ)

Required Parameters: Sources (A and B) addresses.

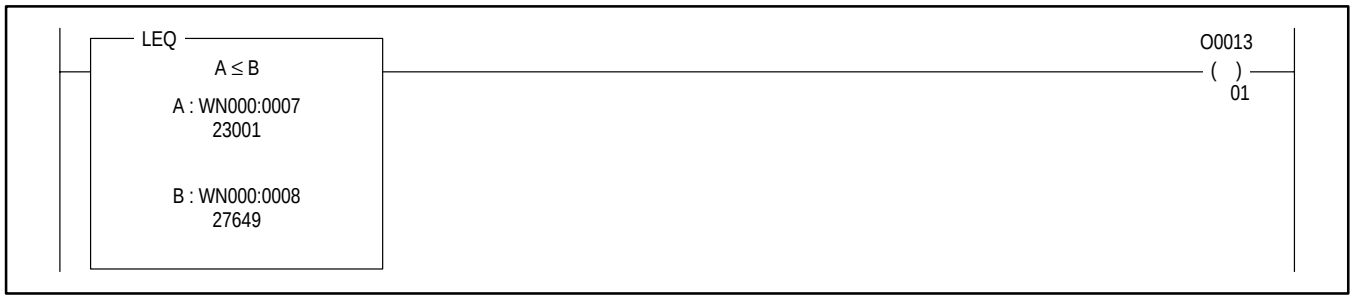
Description: The less-than-or-equal-to instruction tells the processor to compare a value at source (A) with the value at source (B):

If source	Then the rung is
A≤B	true
A>B	false

Example: Figure 6.11 shows a rung containing a less-than-or-equal-to instruction.

The processor compares the source values in integer words 7 and 8. If the value in integer word 7 is less than or equal to the value in integer word 8 (A ≤ B), the processor turns on output O0013/01.

Figure 6.11
Example Rung for a Less-than-or-equal-to Instruction



6.3.7 Limit (LIM)

Required Parameters: Source (A, B, and C) addresses.

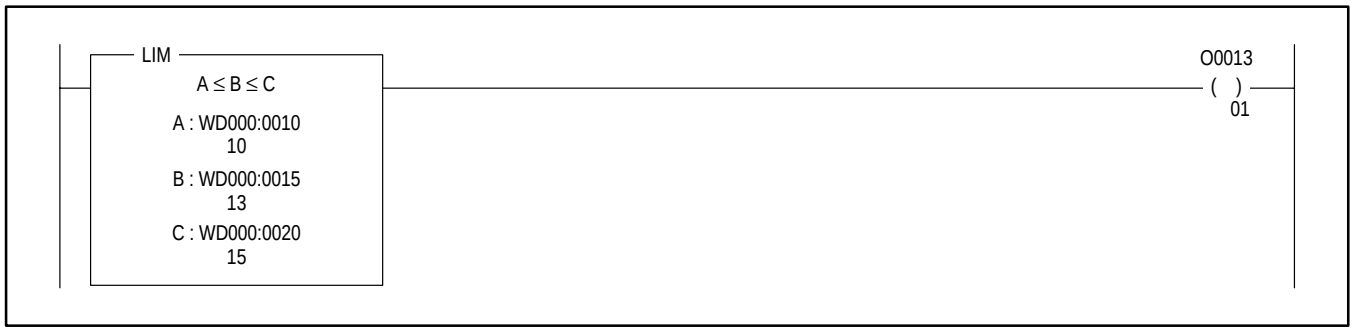
Description: The limit instruction tells the processor to compare values at sources A, B, and C:

If source	Then the rung is
A≤B≤C	true
A>B or B>C	false

Example: Figure 6.12 shows a rung containing a limit instruction.

The processor compares the source values in decimal words 10, 15, and 20. If the value in decimal word 10 is less than the values in decimal words 15 and 20, and the value in decimal word 15 is less than the value in decimal word 20, the processor turns on output O0013/01.

Figure 6.12
Example Rung for a Limit Instruction



6.4 Arithmetic Instructions

Arithmetic instructions are output instructions that operate on integer and/or floating-point data. The data can be stored in any data table section. The upper and lower limits of the data depend on the section where the data are stored.

Table 6.A lists the type of data stored and the range of values for the data table sections. If the result of an arithmetic instruction exceeds the limits of the data table section that stores the result, an error occurs. You can determine the specific error by monitoring the following bits in status file 0 word 0:

- arithmetic fault – bit 17
- arithmetic overflow – bit 15
- arithmetic underflow – bit 14

We recommend that you monitor these bits in your ladder program so that an arithmetic fault does not cause invalid data to be used. Refer to chapter 14 for detailed information on the status files.

When executing arithmetic instructions on integer values, if fractional values result in the final answer, the processor truncates or rounds the final result:

If the resultant remainder is	Then the processor
less than 0.5	truncates or drops the remainder
0.5 or greater	rounds the remainder to the next higher whole number

For example, the processor rounds 3.5 to 4 and truncates 3.2 to 3. If you do not want the processor to truncate or round the result, use values from the floating-point section of the data table.

You can use the following arithmetic instructions:

- add
- subtract
- multiply
- divide
- square root
- negate

6.4.1 Add (ADD)

Required Parameters: Sources (A and B) and destination (R) addresses.

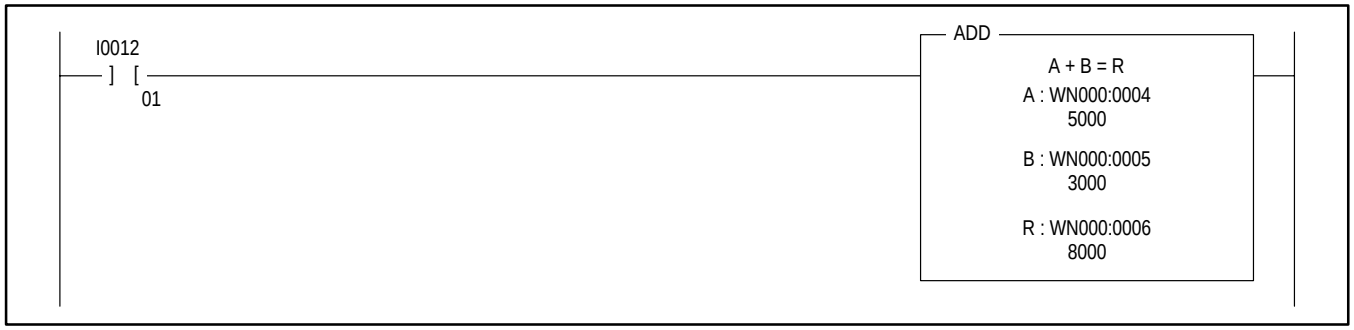
Description: When a rung containing an add instruction is true, the processor adds the value in source (A) to the value in source (B) and puts the result in destination (R). If the rung is false, the processor does not execute the addition instruction, and the destination retains its last value.

If the sources contain values with different numeric formats, the processor automatically converts the data. You can use the add instruction to add positive and negative values: however, if the result is negative, the destination address must be for a data table section that supports negative values. Otherwise an arithmetic fault occurs.

Example: Figure 6.13 shows a rung containing an add instruction.

If the rung is true, the processor adds the source values in integer words 4 and 5 and puts the result in the destination (integer word 6).

Figure 6.13
Example Rung for an Add Instruction



6.4.2 Subtract (SUB)

Required Parameters: Sources (A and B) and destination (R) addresses.

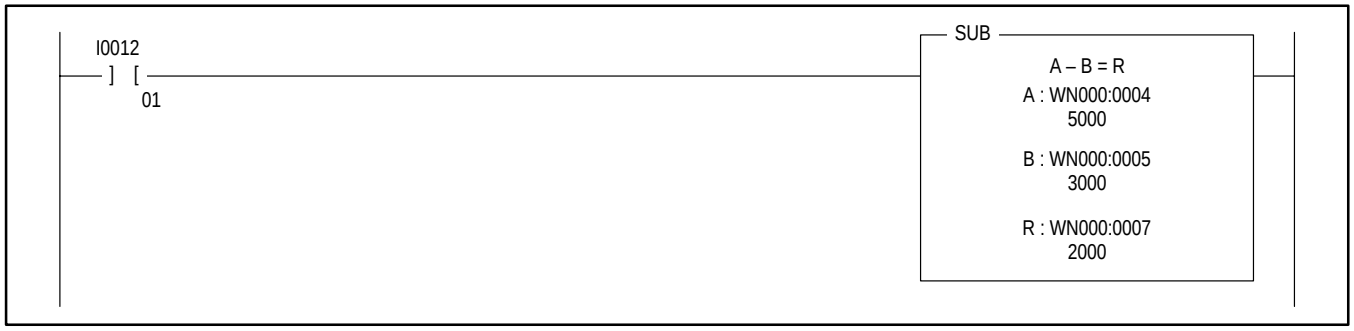
Description: When a rung containing a subtract instruction is true, the processor subtracts the value in source (B) from the value in source (A) and puts the result in destination (R). If the rung is false, the processor does not execute the subtract instruction, and the destination retains its last value.

If the sources contain values with different numeric formats, the processor automatically converts the data. You can use the subtract instruction to subtract positive and negative values; however, if the result is negative, the destination address must be for a data table section that supports negative values. Otherwise an arithmetic fault occurs.

Example: Figure 6.14 shows a rung containing a subtract instruction.

If the rung is true, the processor subtracts the source value in integer word 5 from the source value in integer word 4 and puts the result in the destination (integer word 7).

Figure 6.14
Example Rung for a Subtract Instruction



6.4.3 Multiply (MUL)

Required Parameters: Sources (A and B) and destination (R) addresses.

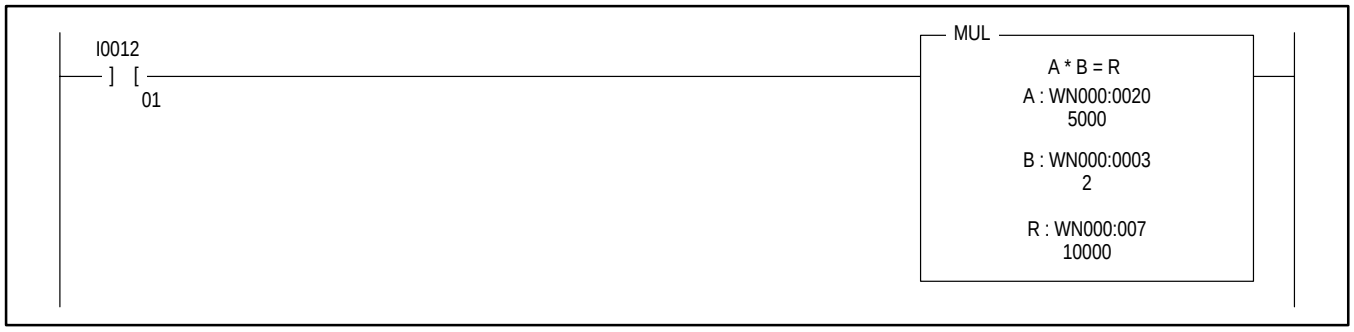
Description: When a rung containing a multiply instruction is true, the processor multiplies the value in source (A) by the value in source (B) and puts the result in destination (R). If the rung is false, the processor does not execute the multiply instruction, and the destination retains its last value.

If the sources contain values with different numeric formats, the processor automatically converts the data. You can use the multiply instruction to multiply positive and negative values; however, if the result is negative, the destination address must be for a data table section that supports negative values. Otherwise an arithmetic fault occurs.

Example: Figure 6.15 shows a rung containing a multiply instruction.

If the rung is true, the processor multiplies the source value in integer word 20 by the source value in integer word 3 and puts the result in the destination (integer word 7).

Figure 6.15
Example Rung for a Multiply Instruction



6.4.4 Divide (DIV)

Required Parameters: Sources (A and B) and destination (R) addresses.

Description: When a rung containing a divide instruction is true, the processor divides the value in source (A) by the value in source (B) and puts the result in destination (R). If the rung is false, the processor does not execute the divide instruction, and the destination retains its last value.

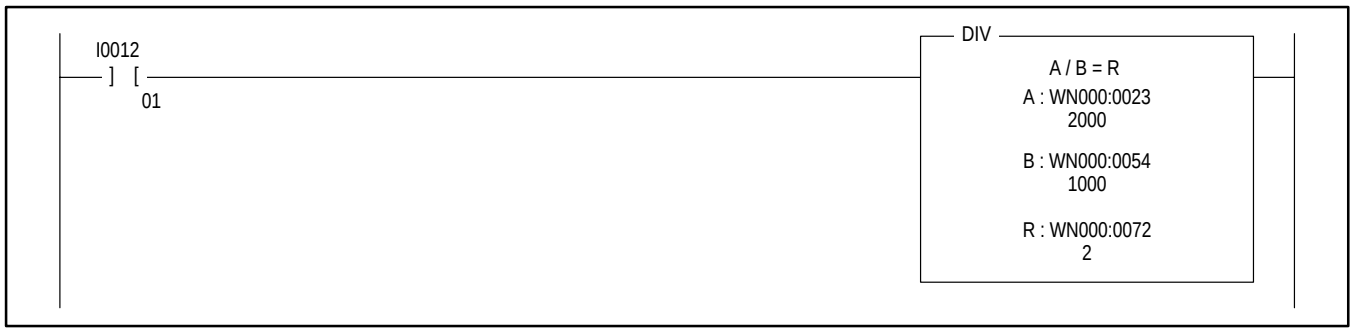
If the sources contain values with different numeric formats, the processor automatically converts the data. You can use the divide instruction to divide positive and negative values; however, if the result is negative, the destination address must be for a data table section that supports negative values. Otherwise an arithmetic fault occurs.

If source (B) contains 0, the processor declares an arithmetic fault and sets status bit 11 in status file 0, word 0 because you cannot divide a value by 0.

Example: Figure 6.16 shows a rung containing a divide instruction.

If the rung is true, the processor divides the source value in integer word 23 by the source value in decimal word 54 and puts the result in the destination (integer word 72).

Figure 6.16
Example Rung for a Divide Instruction



6.4.5 Square Root (SQR)

Required Parameters: Source (A) and destination (R) addresses.

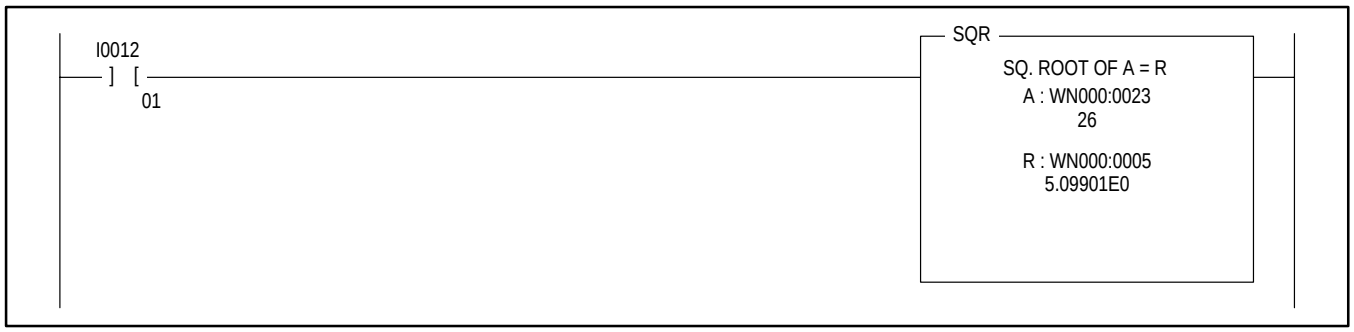
Description: When a rung containing a square-root instruction is true, the processor takes the square root of the value in source (A) and puts the result in destination (R). If the rung is false, the processor does not execute the square-root instruction, and the destination retains its last value.

If source (A) contains 0 or a negative value, the processor declares an arithmetic fault and sets status bit 11 in status file 0, word 0 because you cannot take the square root of 0 or a negative value.

Example: Figure 6.17 shows a rung containing a square-root instruction.

If the rung is true, the processor takes the square root of the source value in integer word 23 and puts the result in the destination (floating-point word 5).

Figure 6.17
Example Rung for a Square-root Instruction



6.4.6 Negate (NEG)

Required Parameters: Source (A) and destination (R) addresses.

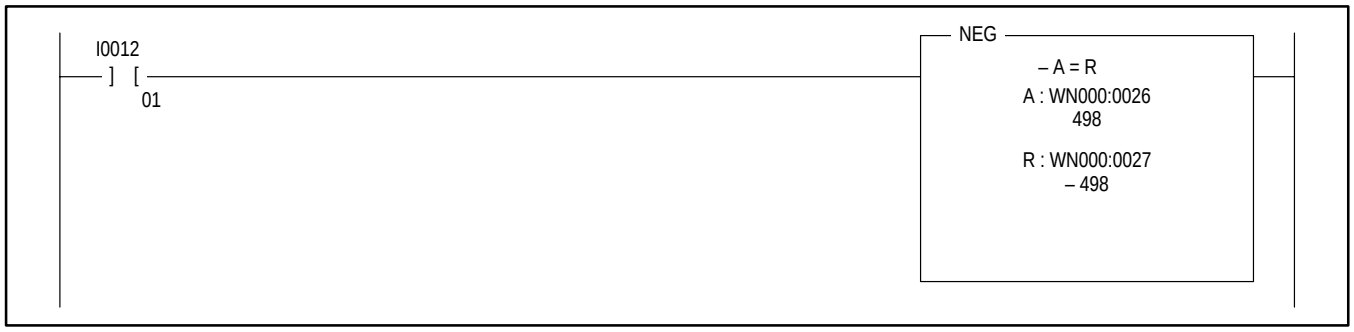
Description: When a rung containing a negate instruction is true, the processor changes the sign of the value in source (A) and puts the result in destination (R). If the rung is false, the processor does not execute the negate instruction, and the destination retains its last value.

Make sure that the destination address is for a data table section that can store negative values. Otherwise the processor declares an arithmetic fault and sets bit 10 in status file 0, word 0.

Example: Figure 6.18 shows a rung containing a negate instruction.

If the rung is true, the processor changes the sign of the source value in integer word 26 and puts the result in the destination (integer word 27).

Figure 6.18
Example Rung for a Negate Instruction



6.5 Logic Instructions

Logic instructions are output instructions that perform 16-bit or one-word logical operations on binary data. In logic operations, the processor considers data in the sources to be binary and performs the operation bit by bit. For example:

```
    11001010
AND 10101010
    --------
    10001010
```

The addresses for sources and the destination are usually from the binary section of the data table. However, you can specify other sections. If you do specify other sections, the processor does not convert data formats. It bases logic operations strictly on the bit pattern of each word.

The values stored in the sources and destination must fall within the limits for the particular data table section used (Table 6.A).

Important: The floating-point and high-order-integer sections of the data table store data in 32-bit words. If you specify a 32-bit word as a source, the destination address must also be a 32-bit word.

If you specify a 16-bit word data table section and a 32-bit word data table section for a logic instruction, the processor executes a 32-bit logical operation by adding 16 zeros to the upper byte of the 16-bit word. The original word is now stored in the lower 16 bits.

You can use the following logic instructions:

- AND
- OR
- XOR
- NOT

6.5.1 AND (AND)

Required Parameters: Sources (A and B) and destination (R) addresses.

Description: When a rung containing an AND instruction is true, the processor ANDs the values in sources (A and B) and puts the result in destination (R). If the rung is false, the processor does not execute the AND instruction, and the destination retains its last value. Table 6.B shows a truth table for a logical-AND operation.

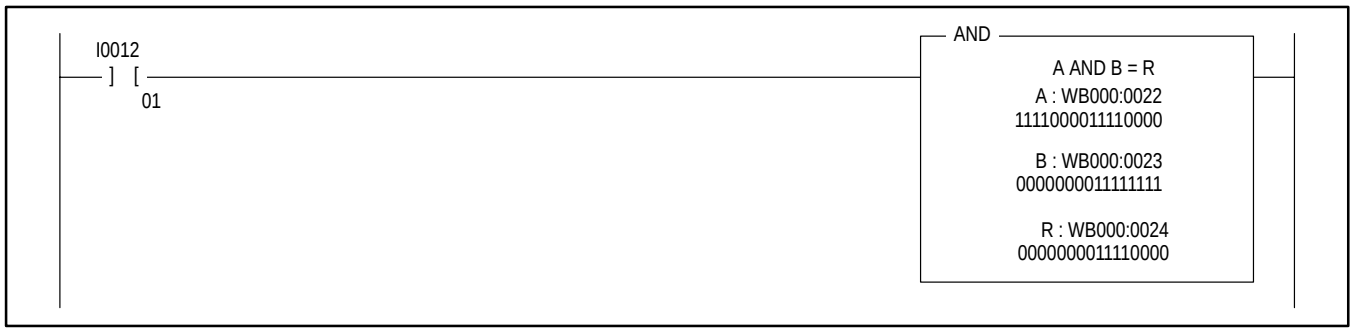
Table 6.B
Truth Table for a Logical-AND Operation

A	B	R
0	0	0
1	0	0
0	1	0
1	1	1

Example: Figure 6.19 shows a rung containing and AND instruction.

If the rung is true, the processor ANDs the source values in binary words 22 and 23 and puts the result in the destination (binary word 24).

Figure 6.19
Example Rung for an AND Instruction



6.5.2 OR (OR)

Required Parameters: Sources (A and B) and destination (R) addresses.

Description: When a rung containing an OR instruction is true, the processor ORs the values in sources (A and B) and puts the result in destination (R). If the rung is false, the processor does not execute the OR instruction, and the destination retains its last value. Table 6.C shows a truth table for a logical-OR operation.

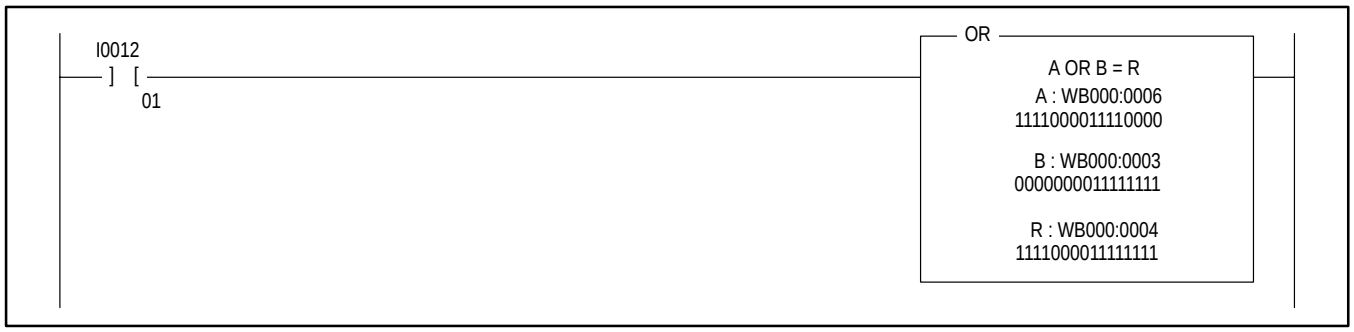
Table 6.C
Truth Table for a Logical-OR Operation

A	B	R
0	0	0
1	0	1
0	1	1
1	1	1

Example: Figure 6.20 shows a rung containing an OR instruction.

If the rung is true, the processor ORs the source values in binary words 6 and 3 and puts the result in the destination (binary word 4).

Figure 6.20
Example Rung for an OR Instruction



6.5.3 XOR (XOR)

Required Parameters: Sources (A and B) and destination (R) addresses.

Description: When a rung containing an XOR instruction is true, the processor XORs the values in sources (A and B) and puts the result in destination (R). If the rung is false, the processor does not execute the XOR instruction, and the destination retains its last value. Table 6.D shows a truth table for a logical-exclusive-OR operation.

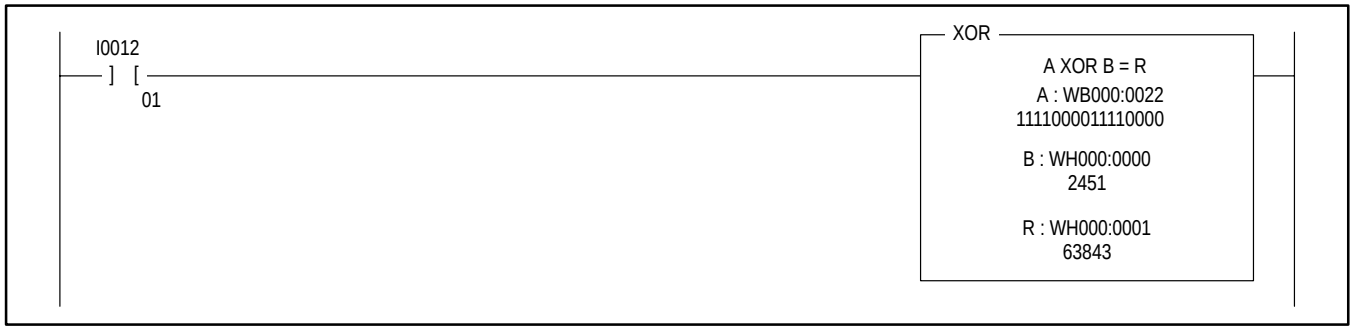
Table 6.D
Truth Table for a Logical-exclusive-OR Operation

A	B	R
0	0	0
1	0	1
0	1	1
1	1	0

Example: Figure 6.21 shows a rung containing XOR instruction.

If the rung is true, the processor XORs the source values in binary word 22 and high order integer word 0 and puts the result in the destination (high order integer word 1).

Figure 6.21
Example Rung for an XOR Instruction



6.5.4 NOT (NOT)

Required Parameters: Source (A) and destination (R) addresses.

Description: When a rung containing a NOT instruction is true, the processor NOTs the value in source (A) and puts the result in destination (R). If the rung is false, the processor does not execute the NOT instruction, and the destination retains its last value. Table 6.E shows a truth table for a logical-NOT operation.

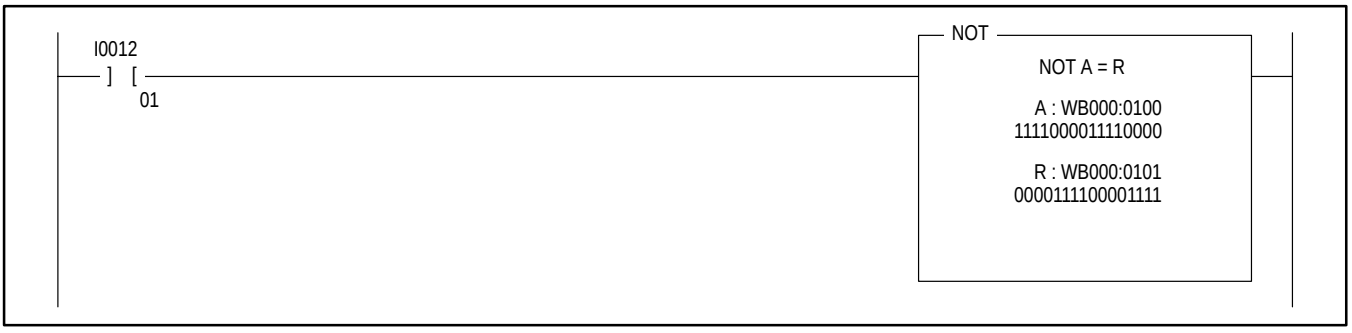
Table 6.E
Truth Table for a Logical-NOT Operation

A	R
0	1
1	0

Example: Figure 6.22 shows a rung containing a NOT instruction.

If the rung is true, the processor NOTs the source value in binary word 100 and puts the result in the destination (binary word 101).

Figure 6.22
Example Rung for a NOT Instruction



Chapter 7

Using Files

7.0 Chapter Objectives

Up to this point, we have described how you can use instructions to alter individual bits and words in the data table. In this chapter we describe how to combine files and instructions to alter groups of words. After reading this chapter you should understand:

- the definition of a file
- how to create and address files in memory
- how to address words within files
- how to address files within files
- how the processor executes file instructions

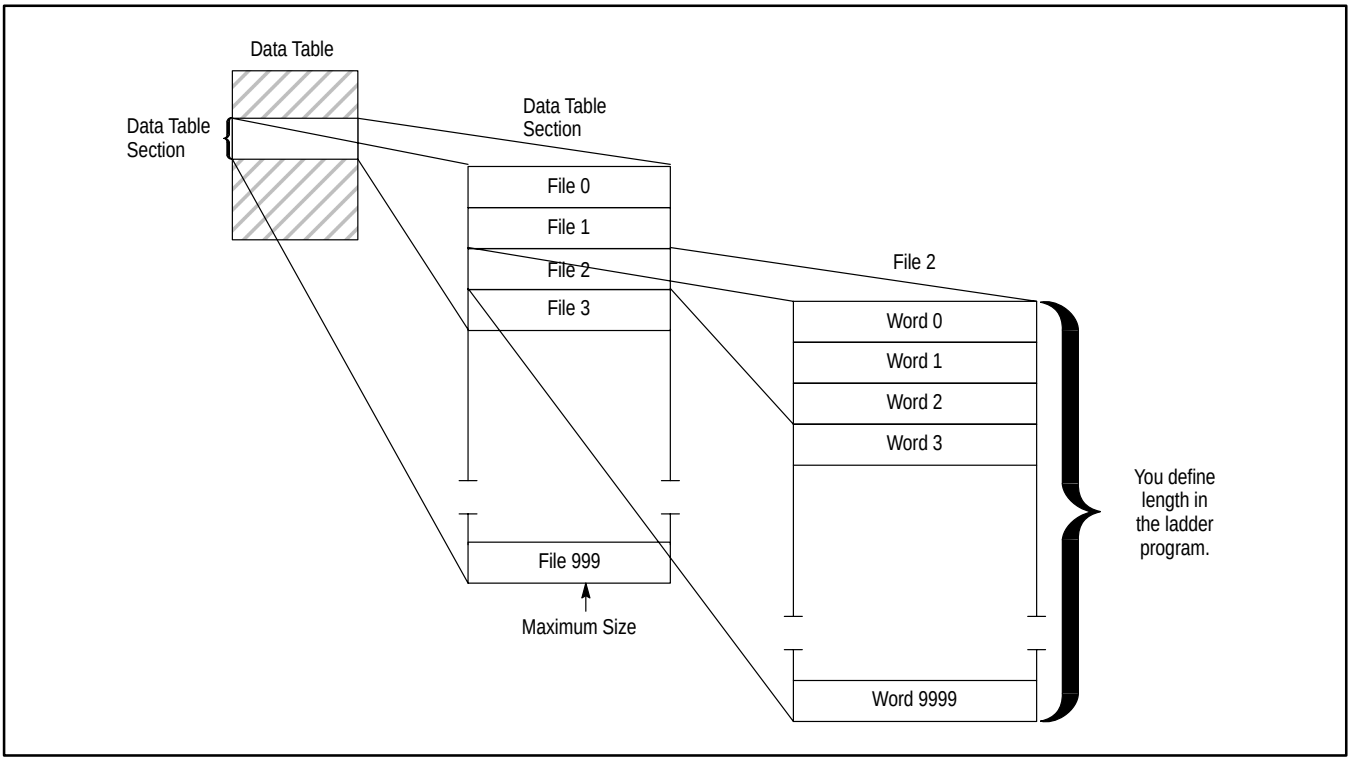
7.1 Defining a File

A **file** is a group of consecutive words from a given data table section. The ladder program can access a file as a complete unit (Figure 7.1). Each data table section can have up to 1,000 distinct files; each file containing up to 10,000 words.

The processor numbers files starting with 0 and expresses word numbers in:

- octal for the input and output image tables
- decimal for all other data table sections

Figure 7.1
Data Table Sections Consist of Accessible Units Called Files



7.2 Creating and Addressing Files

Before executing a ladder program that accesses files, you must create the file in memory. You cannot do this in a ladder program. Instead you create a file with a special command from the program loader, much as you modify the data table through the data monitor. For details, refer to the user's manual for your program loader.

In the ladder program, you can address files in instructions by entering F for the file followed by the section specifier (Table 7.A) and file number. The file delimiter (F) tells the processor that you want to address a group of data table words.

Table 7.A
Data Table Section Specifiers

Section	Specifier	Section	Specifier
output image	O	binary	B
input image	I	ASCII	A
integer	N	high-order integer	H
floating point	F	status	S
decimal	D		

For example, some legitimate file addresses include:

Address	Meaning
F10	The lowest possible address file in the input image table.
FN999	The highest possible address file in the integer section.
FB123	An intermediate address file in the binary section.

Figures 7.2 and 7.3 show an example file address for integer file 3 and integer file organization. You specify the number of words in the file when you create through the program loader.

File numbers need not be consecutive, nor do they need to be created in any particular order. You might, for example, create only the three files shown above, if that suited your programming needs.

However, the processor does allocate three words per file of overhead for unused file numbers between zero and the highest created file number for the data table section. Therefore, for efficient use of memory, we do recommend that you create files consecutively (e.g. FN1, FN2, FN3, etc.). Also, the processor executes instructions on file number 0 to 19 faster than file numbers greater than 19.

Furthermore, you can delete some files without affecting existing files. As an example, if you had created files FF2, FF3, FF4, and FF5, you could delete FF3, leaving FF2, FF4, and FF5.

Figure 7.2
Addressing Data in File Storage

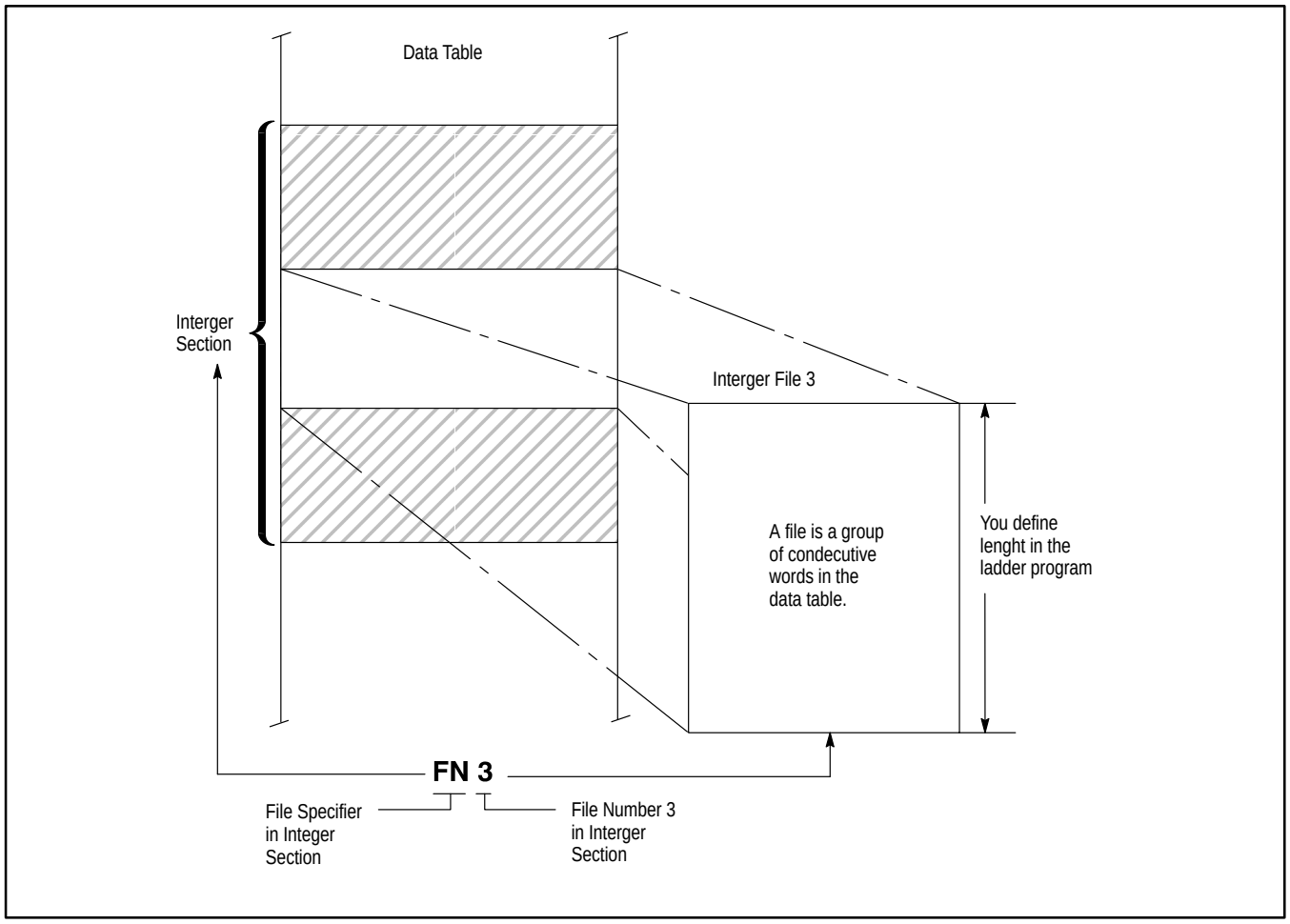
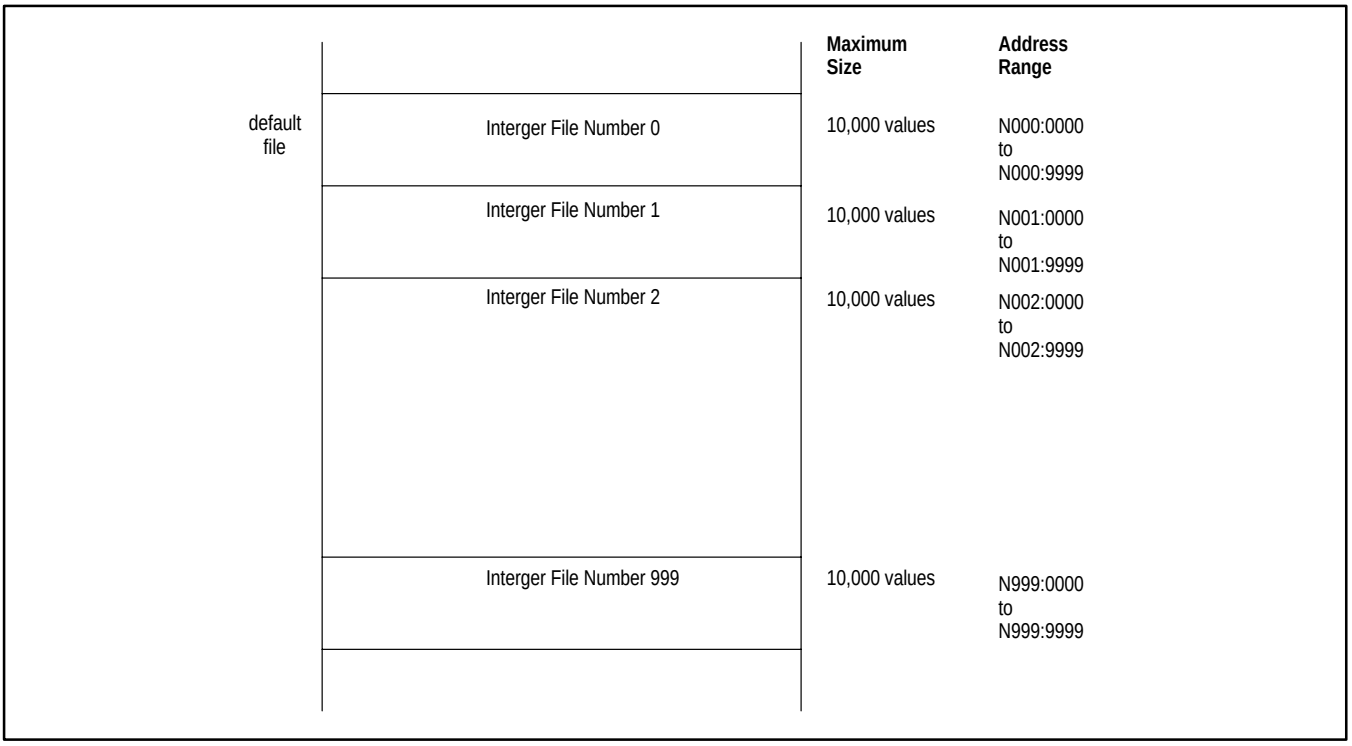


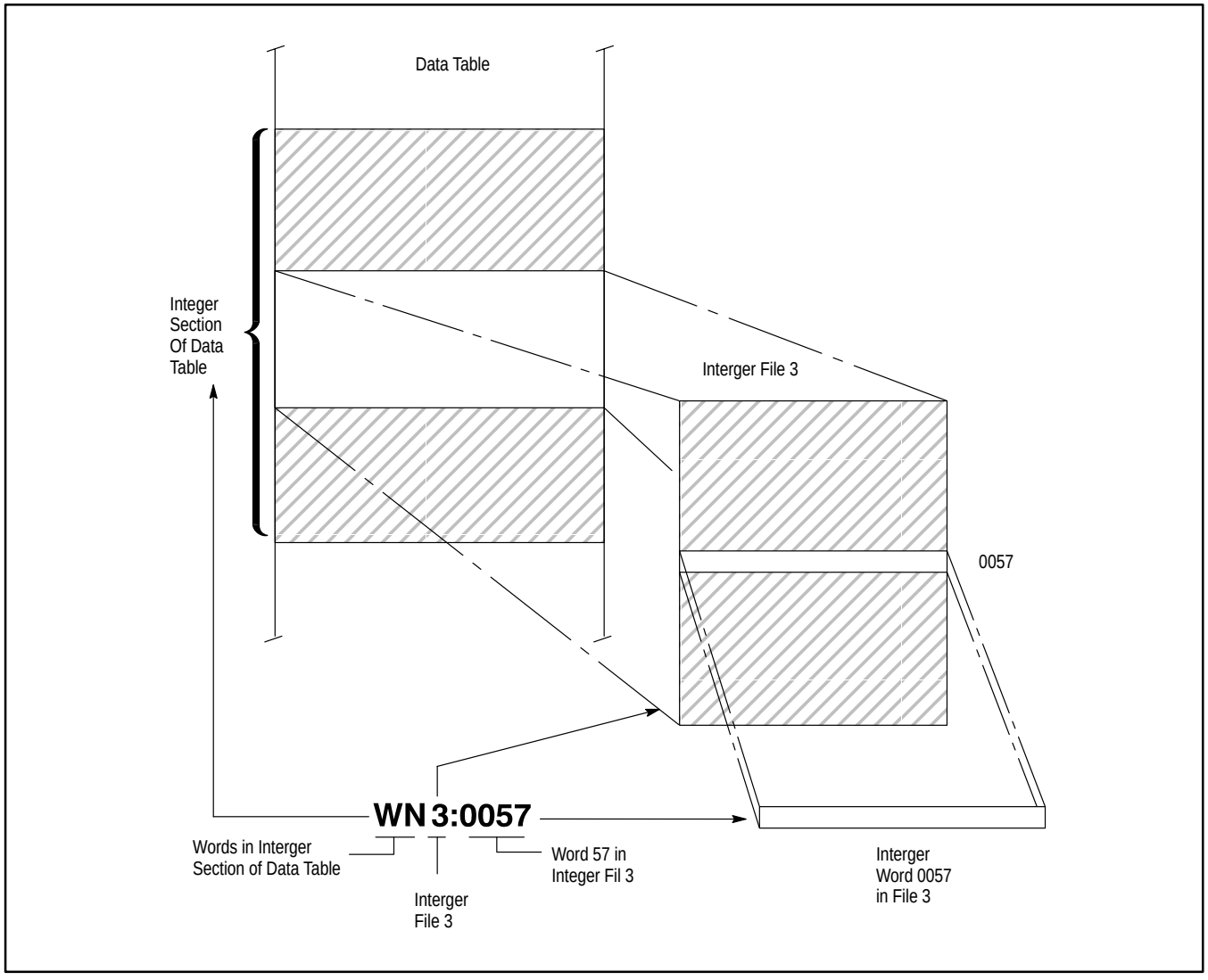
Figure 7.3
File Organization Within a Data Table Section



7.2.1 Addressing a Word within a File

To address the words that make up a file, enter W for word followed by the file address, a colon (:) and the word number. The word delimiter (W) tells the processor that you want to address one data table word. Figure 7.4 shows an example file address for word 57 in integer file 3.

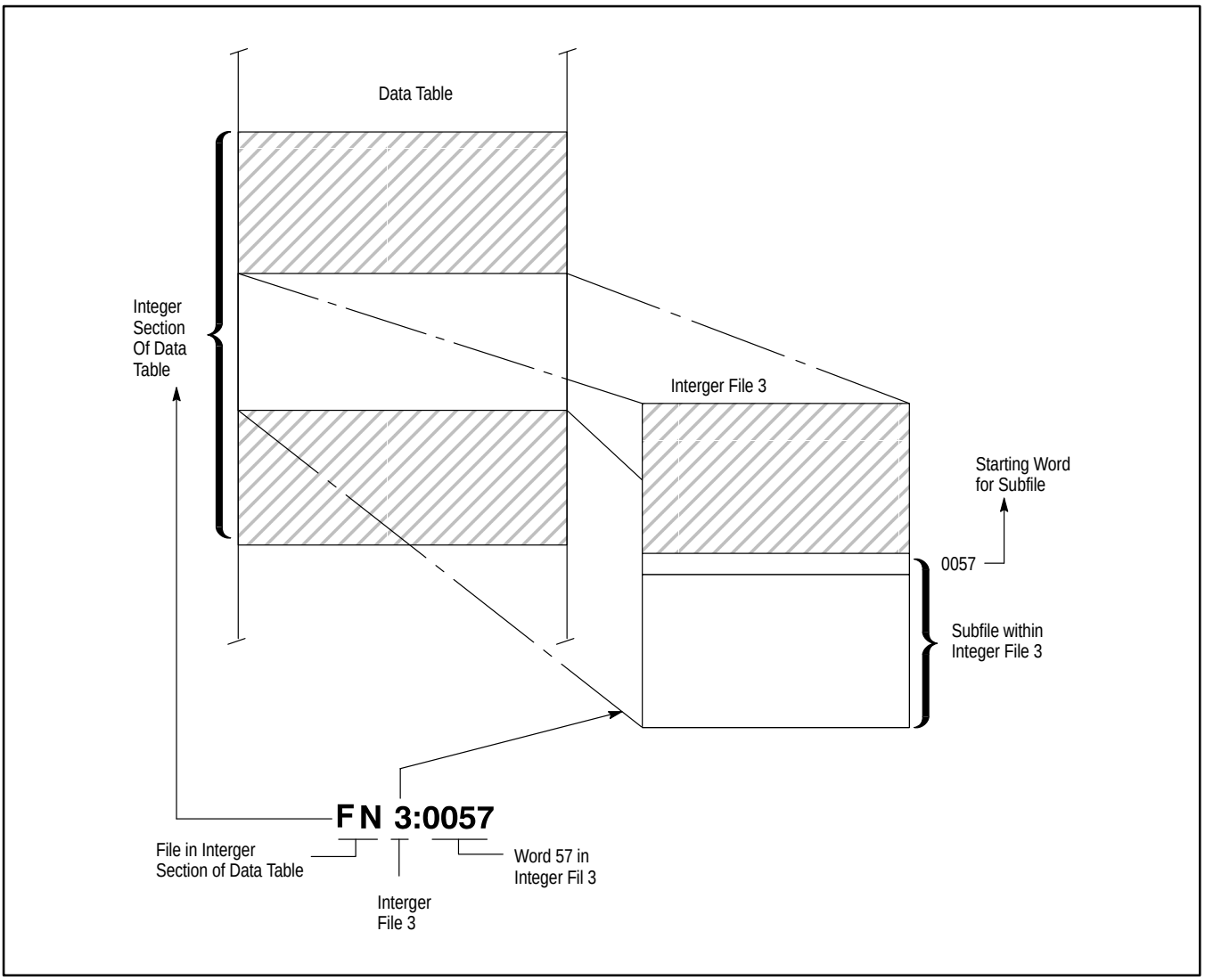
Figure 7.4
Addressing a Word within a File



7.2.2 Addressing a Group of Words within a File

Once you have created a file, you can also have the processor access a portion of the file or a subfile by specifying a word within a file to act as the starting word for the subfile. Figure 7.5 shows an example subfile address. Notice that the address is the same as in Figure 7.4 except that the file specifier is used. For this example, the subfile begins at word 57 and ends with the last word in integer file 3. You specify the number of words for the file in the instruction that manipulates the file.

Figure 7.5
Addressing a File within a File



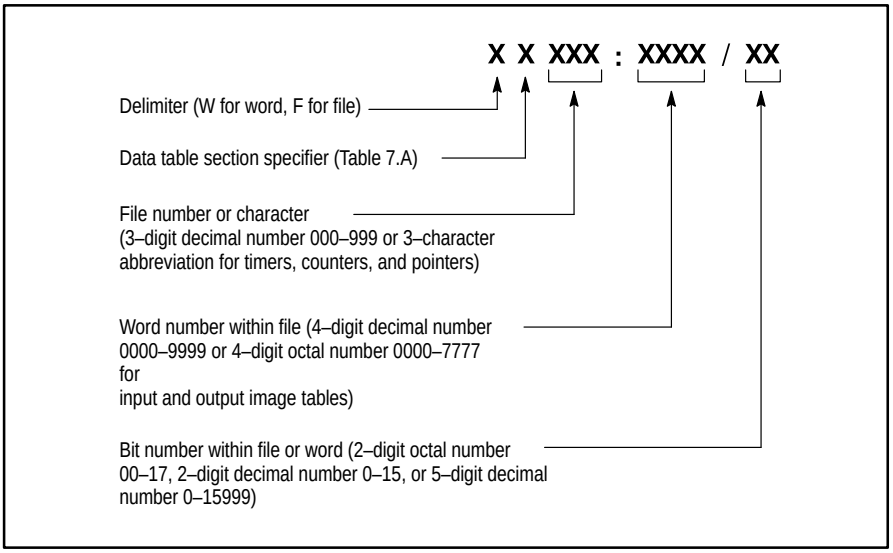
7.2.3 Addressing a Bit within a File

You can address an individual bit within a file in three ways:

- with octal values for the word and bit addresses
- with decimal values for the word and bit addresses
- with a decimal value for the bit alone

These addressing techniques are shown in Figures 7.6 and 7.7.

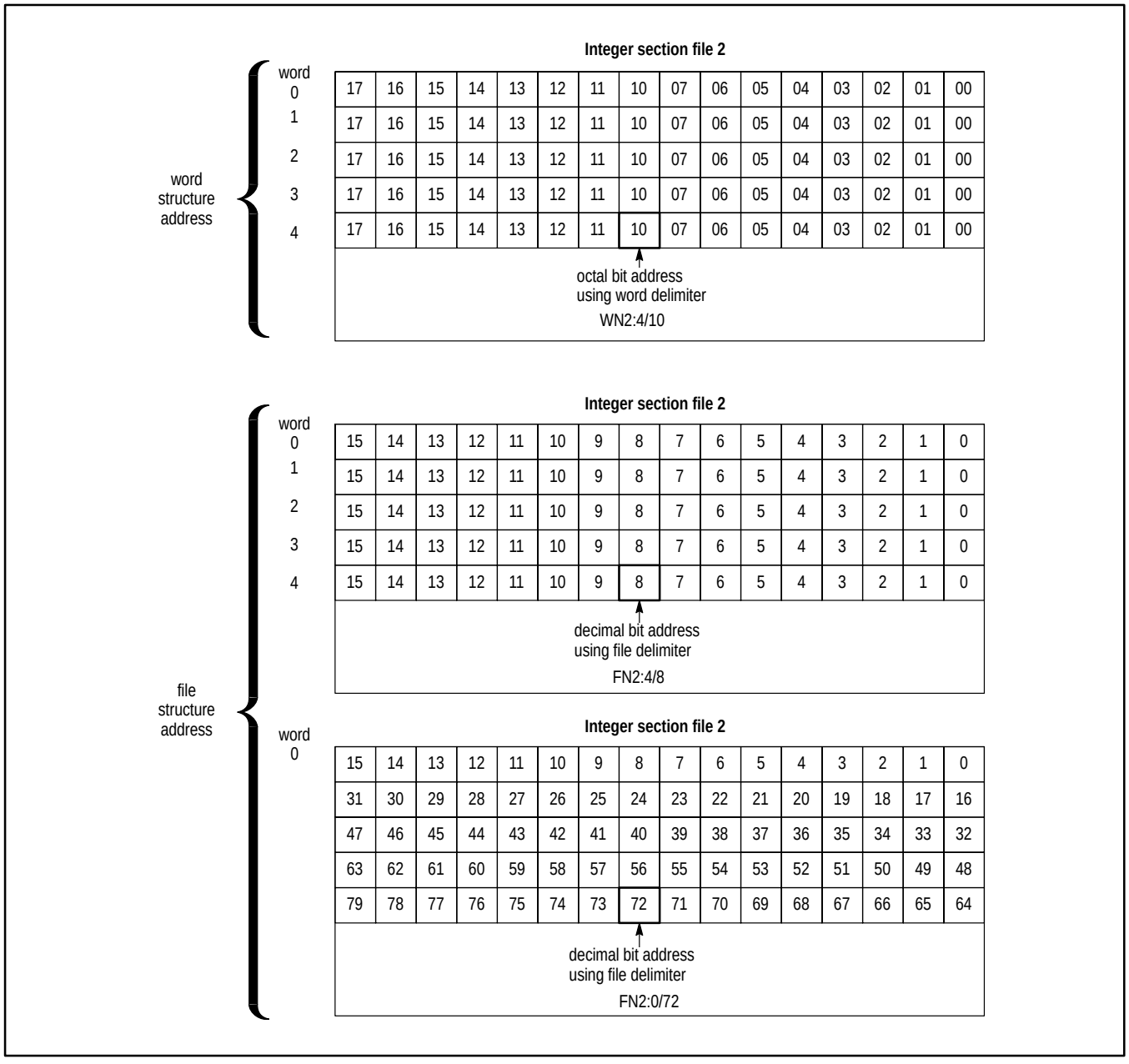
Figure 7.6
Addressing Bits within Files



If you are addressing bits within words:

If you are using the	Then bits are numbered in
word delimiter (W)	octal (00 to 07 and 10 to 17)
file delimiter (F)	decimal (0 to 15) and bit numbers are one less than the bit position. That is, the first bit in a file word is number 0, the second bit is number 1, the third number 2, and so on.

Figure 7.7
Example Bit Addresses



If you are addressing bits within the file itself:

- Bit numbering begins at 0 from the word specified as the starting word. If you begin the file at word 10, then the first bit in word 10 is 0, the second is 1, and so on until the end of the file.
- Bit numbers are one less than the bit position. The first bit with a file is number 0. If you set the number of bits within the file to 48, then the last bit number is 47.

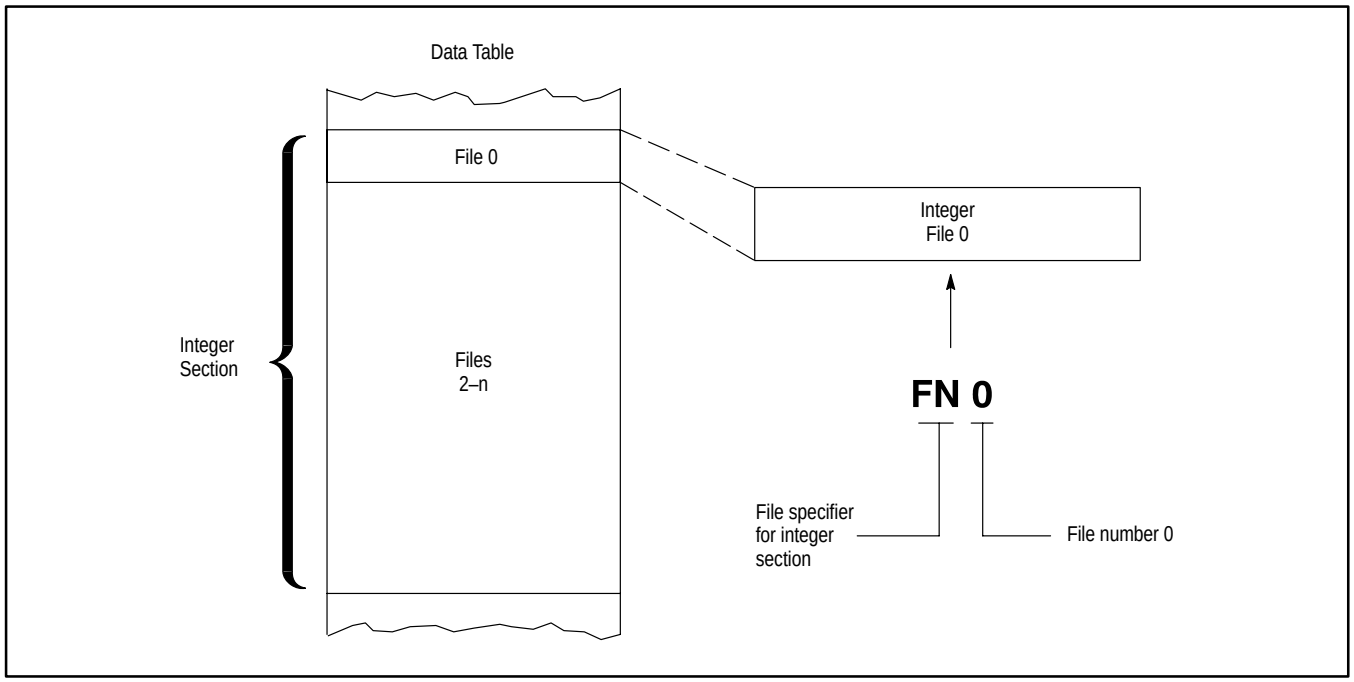
CAUTION: If you attempt to access a bit number that is not within the file, the processor declares a bad address major fault and shuts down.

7.2.4 Addressing File 0

The processor creates file 0 for each data table section when you initiate communications between the program loader and the controller. In chapters 2 through 6, the example instructions address file 0 for a data table section. Figure 7.8 shows an example address for integer file 0.

If you do not specify a file number when programming an instruction, the processor defaults to file 0. For example, the address I3/01 is the same as I0:3/01.

Figure 7.8
Addressing File 0



7.2.5 Addressing Timers, Counters, and Pointers Using Files

In the timer, counter, and pointer sections of the data table, you can create files for:

- timer or counter control words
- timer or counter preset values
- timer or counter accumulated values
- pointer section, file, or word designations

Figure 7.9 shows the word organization for the timer section of the data table. Each timer uses three words, one for the control, one for the preset value, and one for the accumulated value.

Figure 7.9
Addressing a File within the Timer and Counter Sections

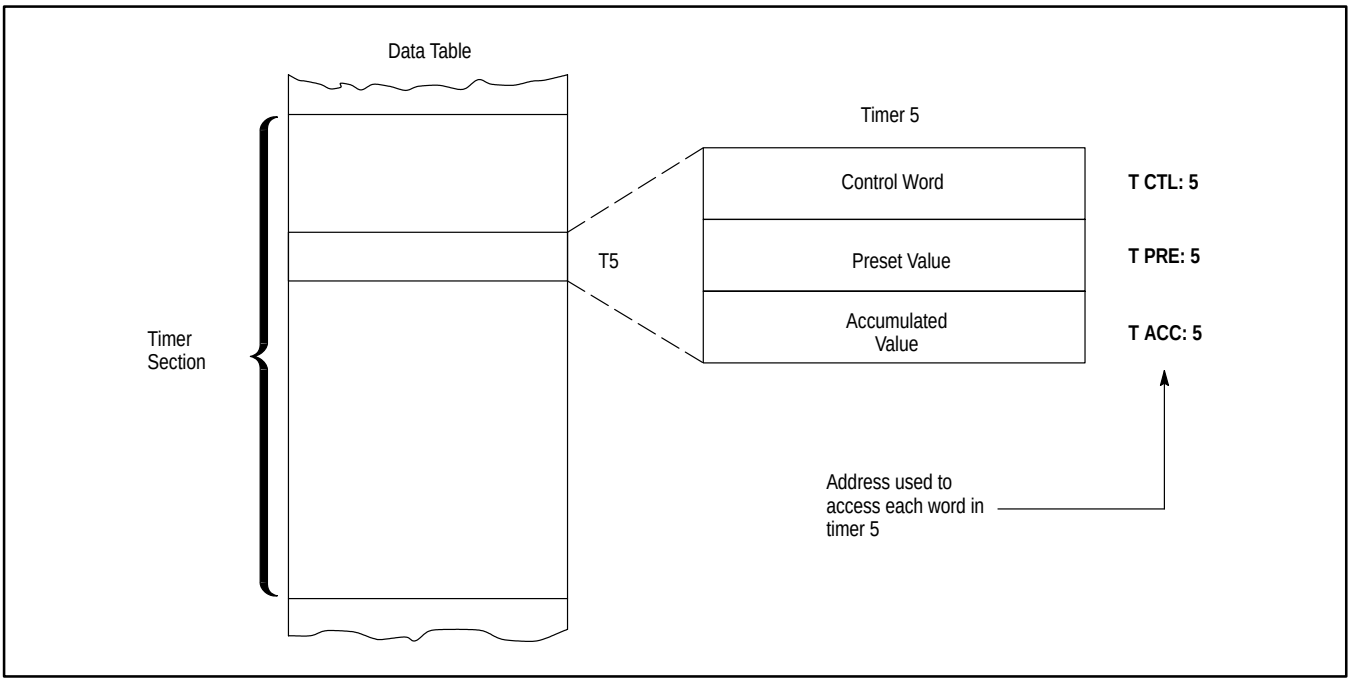
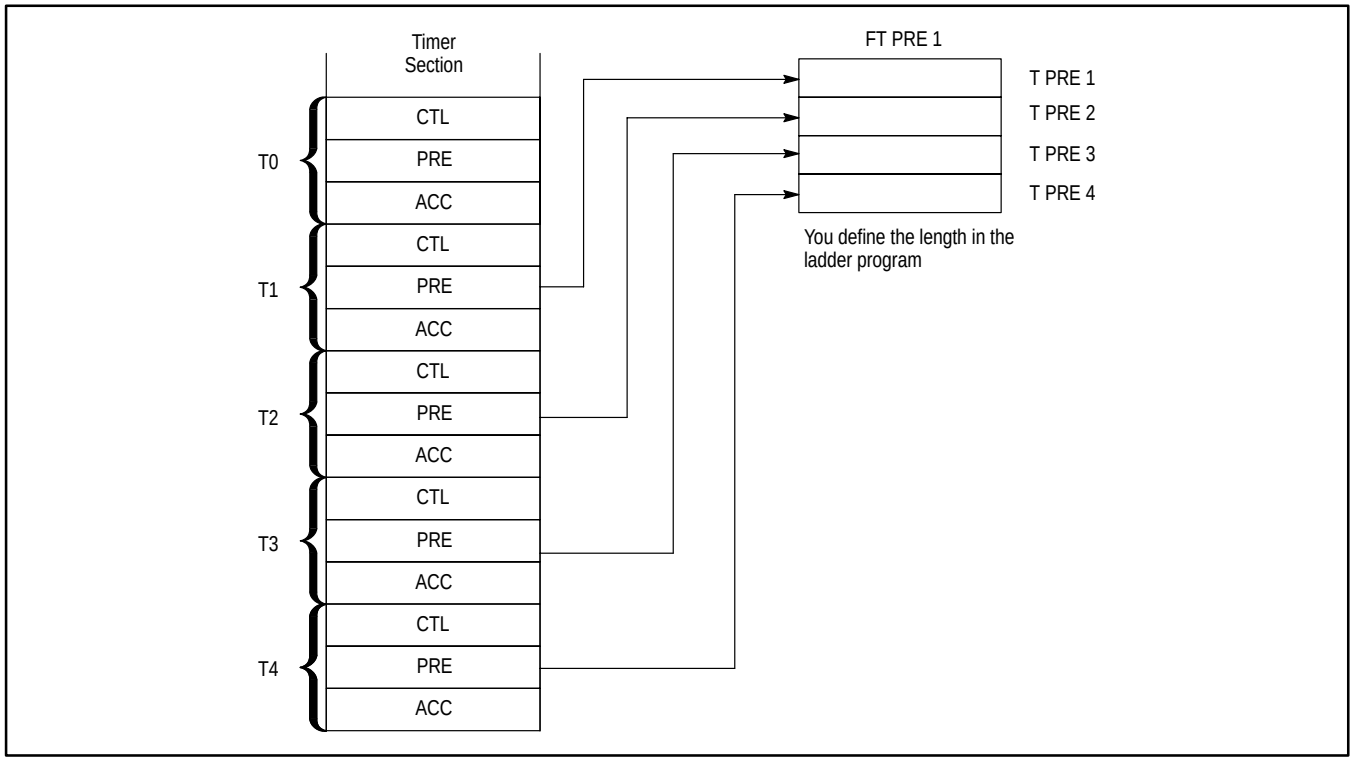


Figure 7.10 shows an example file consisting of timer preset values. When storing and retrieving data in this file, the processor scans the timer section accessing the timer preset words as through they were a file.

Figure 7.10
Example File Consisting of Timer Preset Values



You can program file structures of this type for the counter and pointer sections as well. Refer to chapter 11 for detailed information on pointers.

7.3 File Operation

The controller features the following types of instructions that operate on files:

- data transfer
- data comparison
- arithmetic
- logic

You can use these instructions on entire files of data or individual words within a file. We describe the file instructions in the following chapters.

In programming file instructions, you need to provide the processor with the following information.

- address or addresses for the file(s) in the data table that tell the processor where to find or store the data
- counter that the processor uses to locate data for the file

WARNING: Do not use a counter assigned to a file instruction for any other purpose. Unexpected operation could result in damage to equipment and/or injury to personnel.

- file length (the number of words executed) for the file instruction
- file position (the word location) in the file that the processor is currently accessing. You generally enter a zero to start at the beginning of the file.
- file mode of operation that tells the processor how to execute the file operation

7.3.1 Counter Operation for File Instructions

When used with a file instruction, the words that make up the counter store the following information (Figure 7.11):

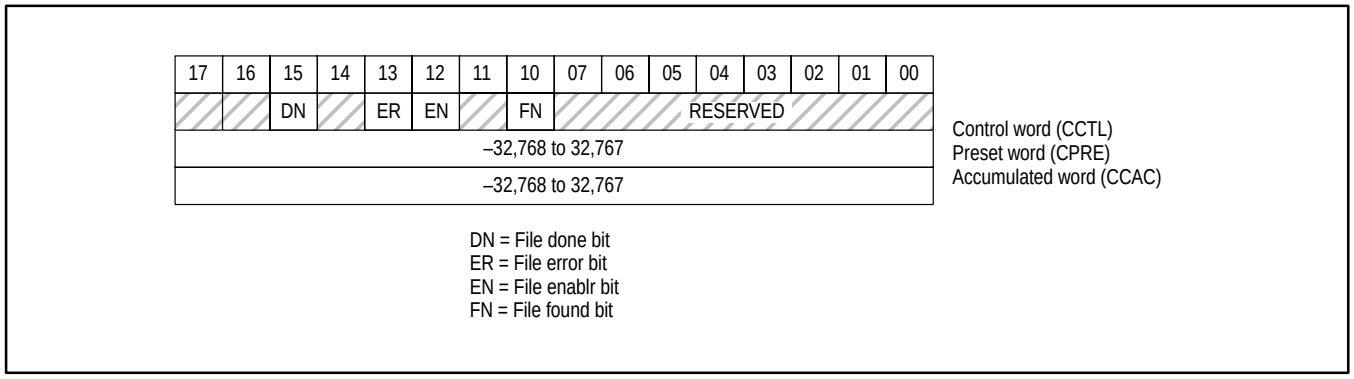
Control word (CCTL) stores the control bits that reflect the status of the file instruction:

- File Done – Bit 15 (DN) shows that a file operation is complete
- File Error – Bit 13 (ER) shows that an error has occurred during the file operation. File errors include overflow, underflow, operand fault, conversion fault. You can find out the error by monitoring word 0 in the status section (S0:0) (refer to chapter 14). If an error occurs, the processor stops executing the file instruction and stores the file word number that caused the error in the counter accumulated value word. To restart the file operation , you can reset the:
  - counter using the reset instruction to restart the entire file operation
  - error bit to restart the file operation from the point that the error occurred
- File Enable – Bit 12 (EN) shows that a file operation is in progress
- File Found – Bit 10 (FN) shows that a file data comparison condition has been met.

Preset value word (CPRE) stores the file length or the number of words in the file.

Accumulated value word (CACC) stores the file position or the word in the file that the processor is currently accessing.

Figure 7.11
Memory Storage for a Counter Working with a File Instruction



7.3.2 File Mode Operation

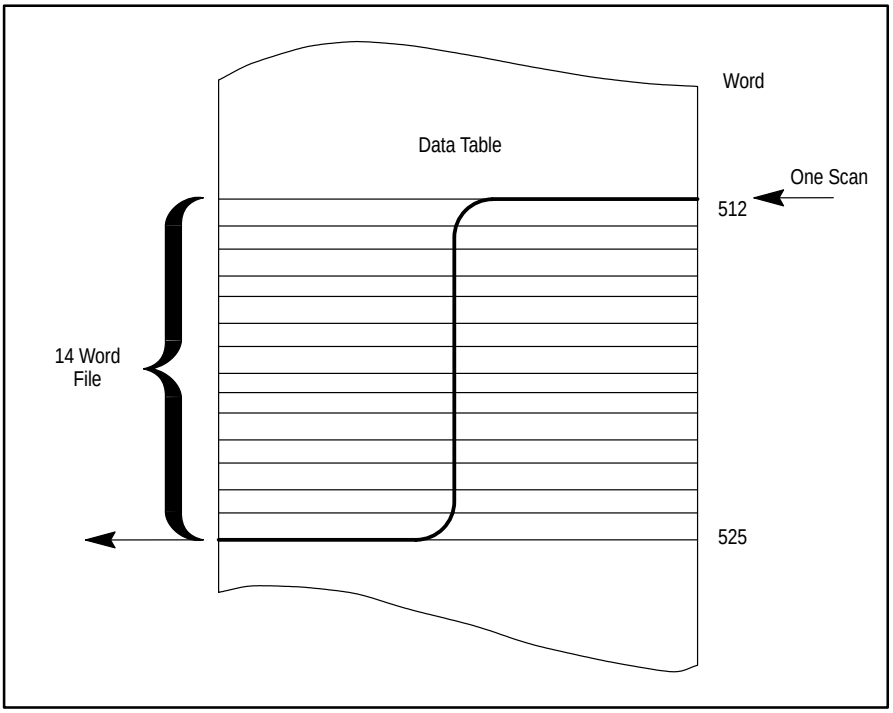
The file mode that you select in programming the file instruction tells the processor how to execute the file operation. You can select one of the following file modes:

- all
- numeric
- increment
- none

All Mode

All mode tells the processor to execute the entire file operation in one program scan (Figure 7.12).

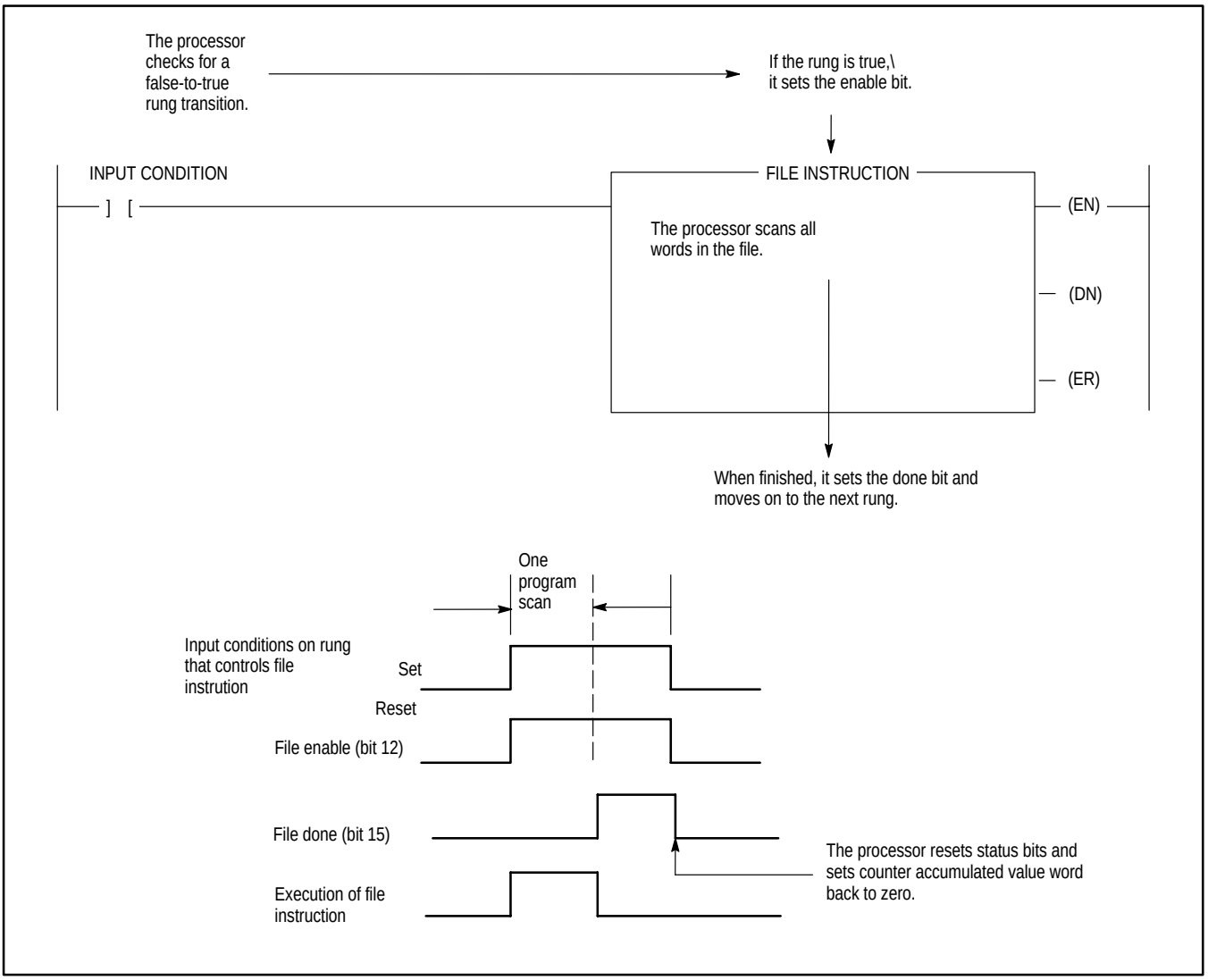
Figure 7.12
Operating File Instructions in All Mode



File instructions are output instructions. When a rung containing a file instruction goes from false to true, the processor begins the file operation and stops when either the file operation is complete, or an error occurs.

Figure 7.13 summarizes how the processor executes a file instruction programmed for the all mode and shows a timing diagram.

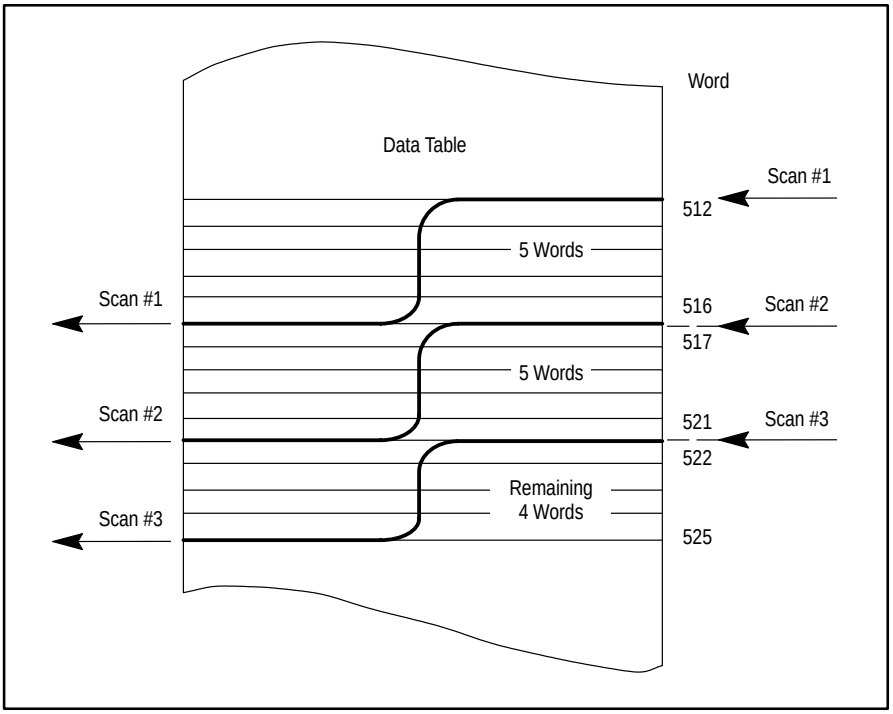
Figure 7.13
Sequence of Operations and Timing Diagram for All Mode



Numeric Mode

Numeric mode tells the processor to execute the entire file operation over multiple program scans (Figure 7.14).

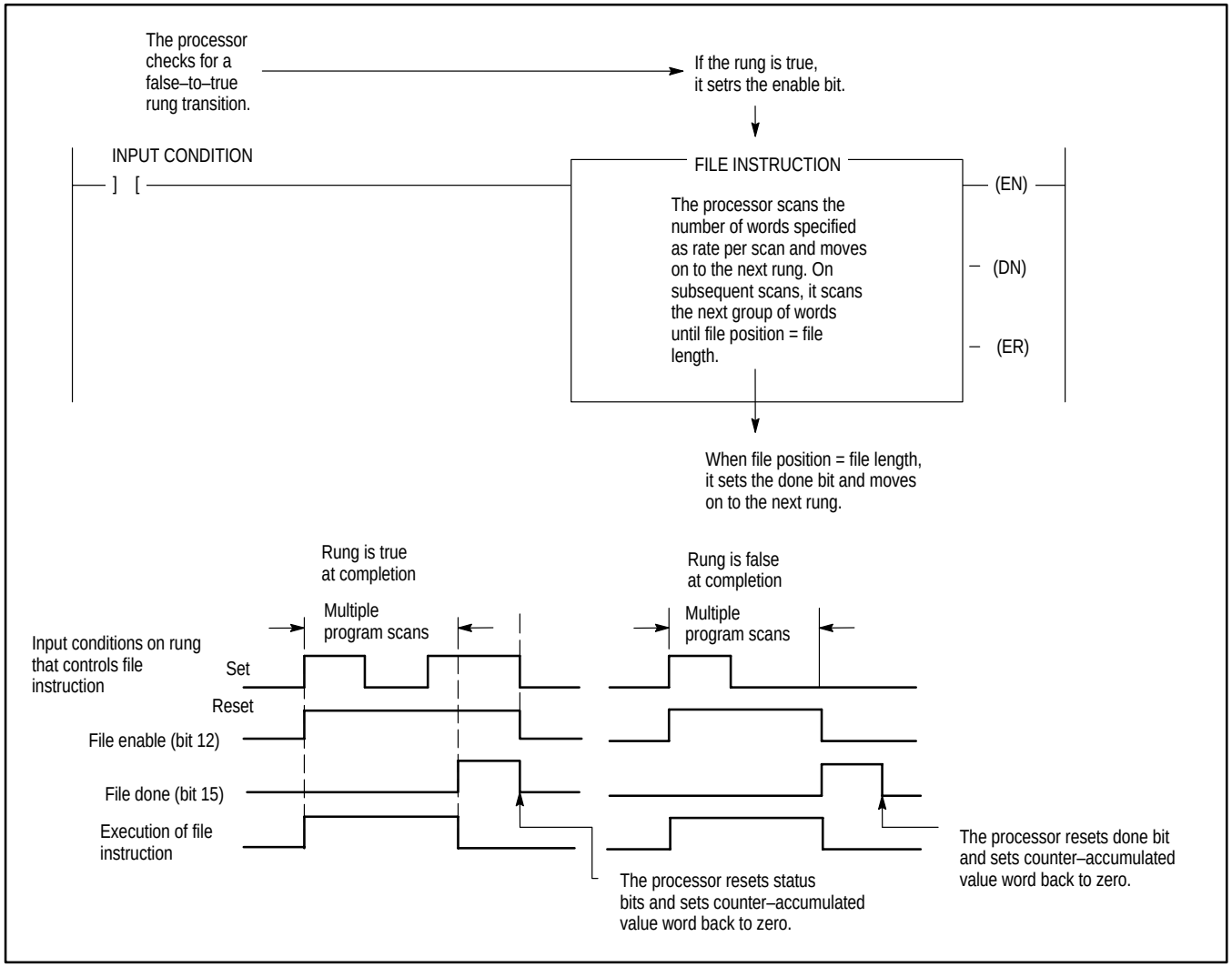
Figure 7.14
Operating File Instructions in Numeric Mode



In the file numeric mode, you specify the rate per scan or the number of words that the processor executes on each program scan. The rate per scan must be less than the file length. When a rung containing a file instruction goes from false to true, the processor begins the file operation and scans the number of words specified as the rate per scan. Then it moves on to the next rung(s) in the ladder program. On the following scan, the processor again scans the next specified number of words in the file. This process repeats until the processor scans all the words in the file, and either the file operation is complete, or an error occurs.

Figure 7.15 summarizes how the processor executes a file instruction programmed for the file numeric mode and shows a timing diagram.

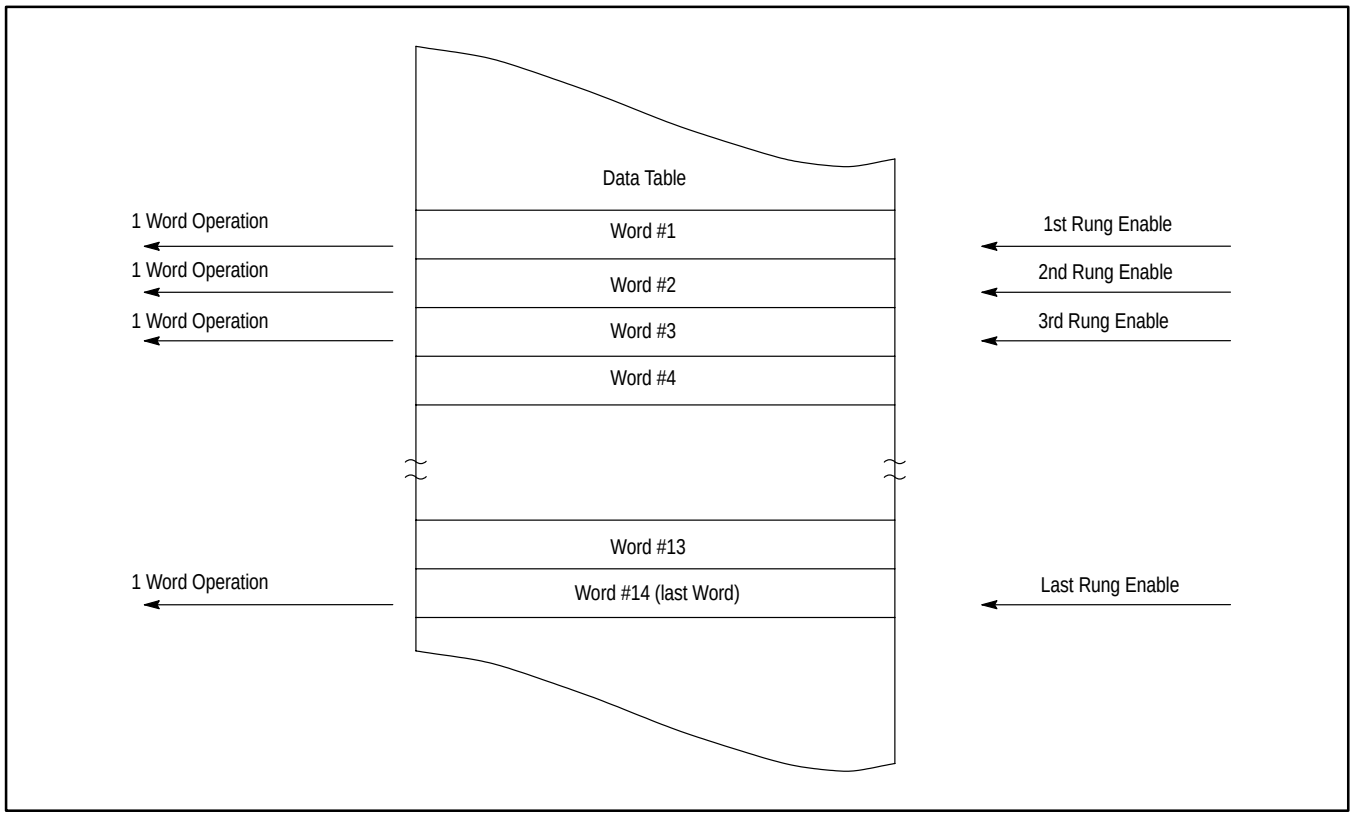
Figure 7.15
Sequence of Operations and Timing Diagram for All Mode



Increment Mode

Increment mode tells the processor to execute the file operation one word at a time for each false-to-true rung transition (Figure 7.16).

Figure 7.16
Operating File Instructions in Increment Mode

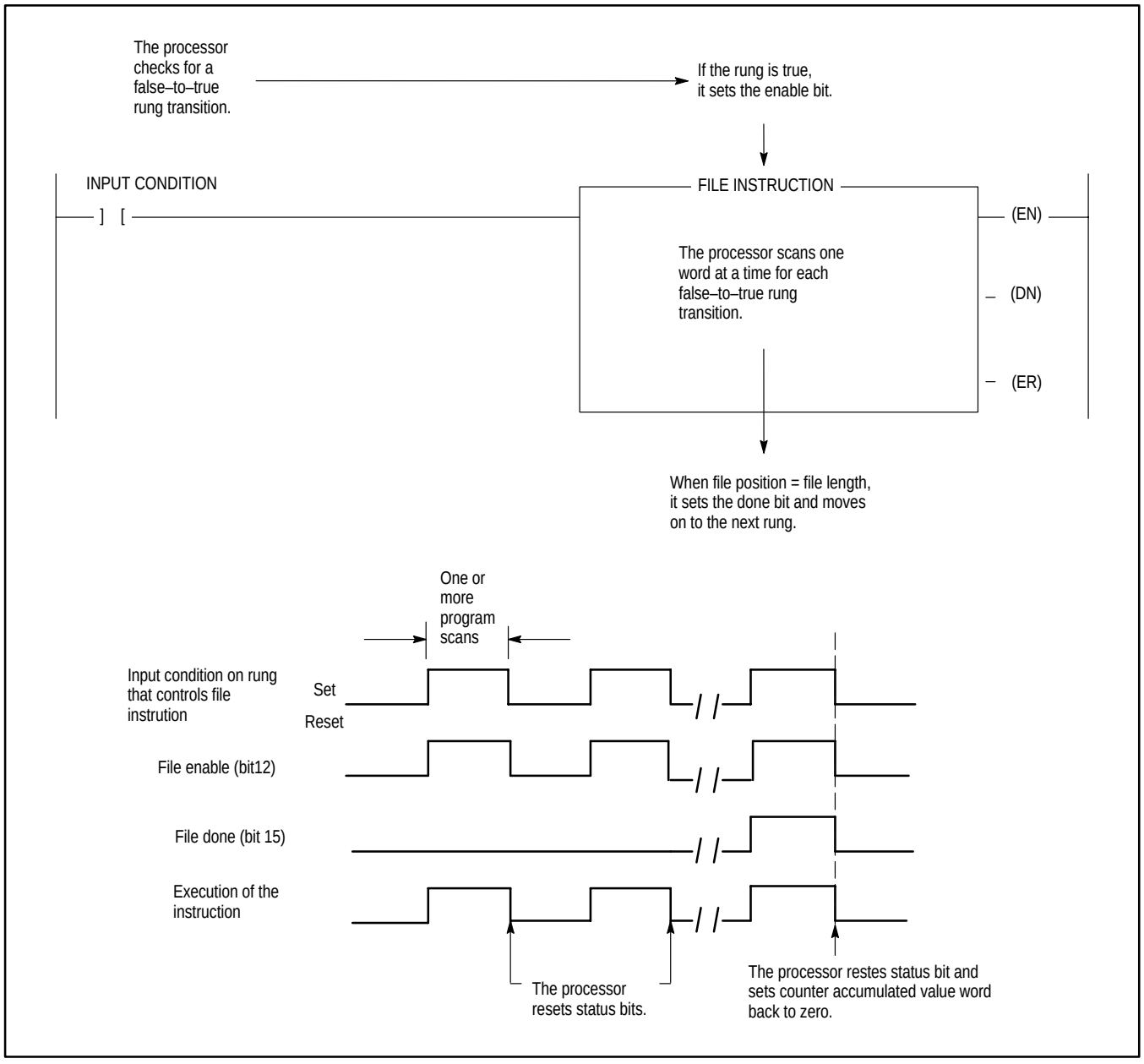


When a rung containing a file instruction is true, the processor executes the file operation on the word pointed to by the counter accumulated value. Then the counter accumulated value increments to the next word in the file.

Increment mode is similar to the numeric mode with rate per scan set to 1. However, there is one important difference. In numeric mode, one false-to-true rung transition tells the processor to continue scanning the file until completion. In increment mode, multiple false-to-true rung transition are necessary for the processor to scan the entire file.

Figure 7.17 summarizes how the processor executes a file instruction programmed for increment mode and shows a timing diagram.

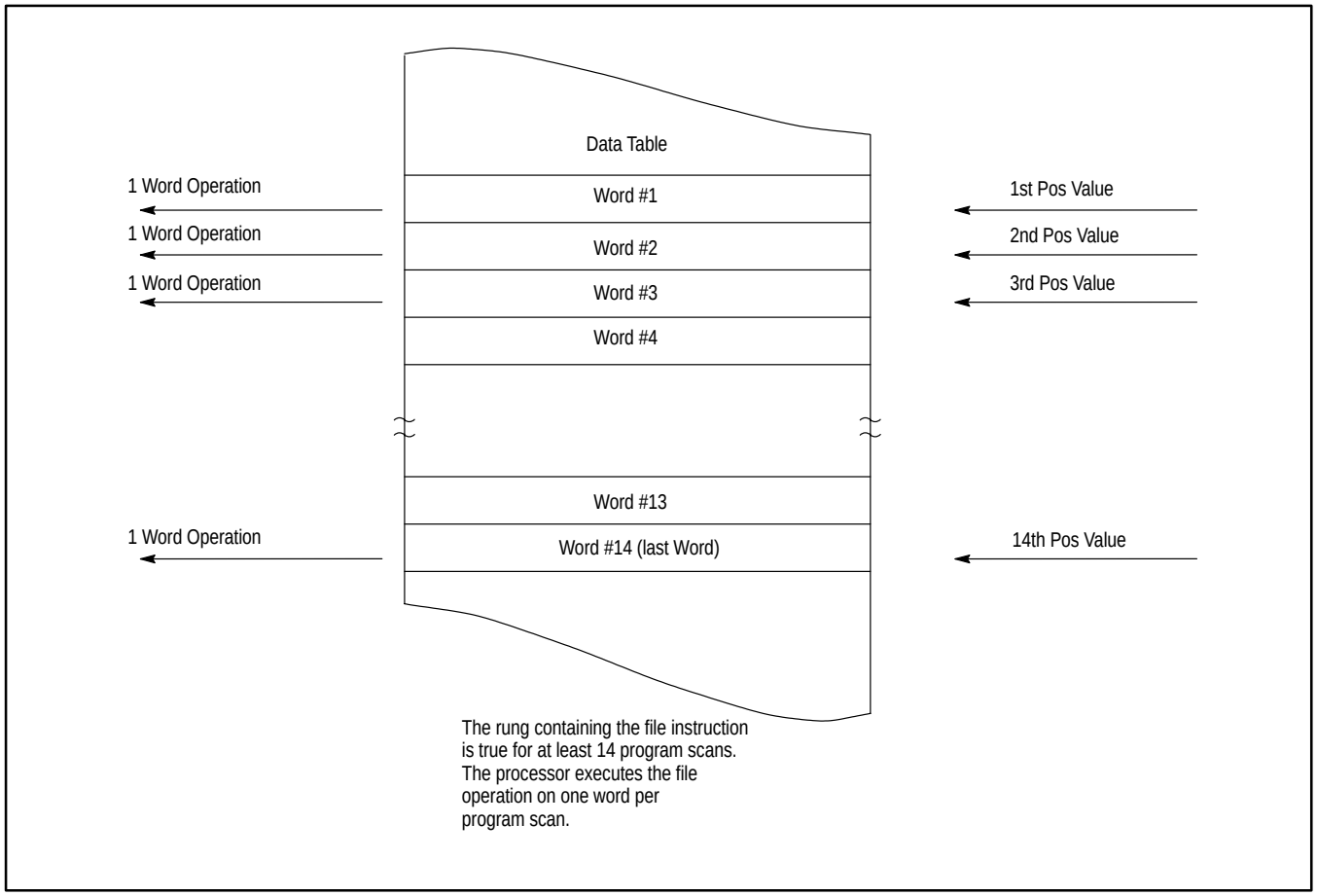
Figure 7.17
Sequence of Operations and Timing Diagrams for Increment Mode



None Mode

None mode tells the processor to execute the file operation on the word specified by the file position (Figure 7.18).

Figure 7.18
Operating File Instructions in None Mode



None mode has an important difference from the other file operating modes. In none mode, the file instruction does not increment. Instead, you use ladder logic to change the position to any desired word location. You must change the location before the file instruction rung goes true. You can use a :

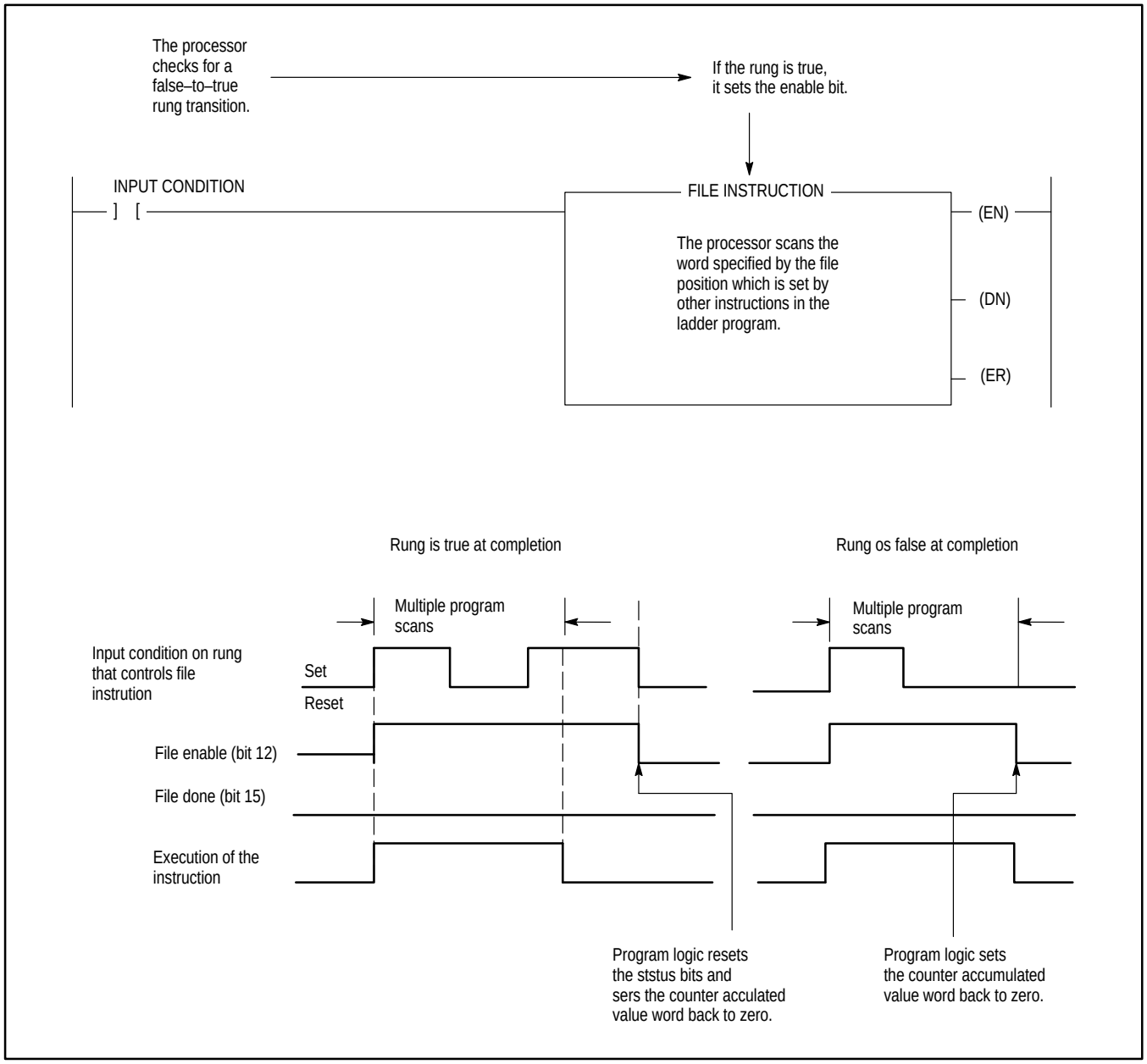
- move instruction to increment the file position or
- counter instruction to increment the file position sequentially

In either case, you assign the move or counter instruction the same address as the counter accumulated value of the file instruction.

When a rung containing a file instruction goes from false to true, the processor operates on the file word specified by the counter accumulated value. As long as the rung remains true, the processor executes the same file operation during subsequent program scans.

Figure 7.19 summarizes how the processor executes a file instruction programmed for the non mode and shows a timing diagram.

Figure 7.19
Sequence of Operation and Timing Diagram for None Mode



CAUTION: In none mode the file instruction does not control the file position value. So, your ladder program could move the file position beyond the file boundary. If the processor detects this condition, it responds with a major fault and displays the BAD ADDRESS fault message on the front panel.

Chapter 8

Using Data-manipulation Instructions with Files

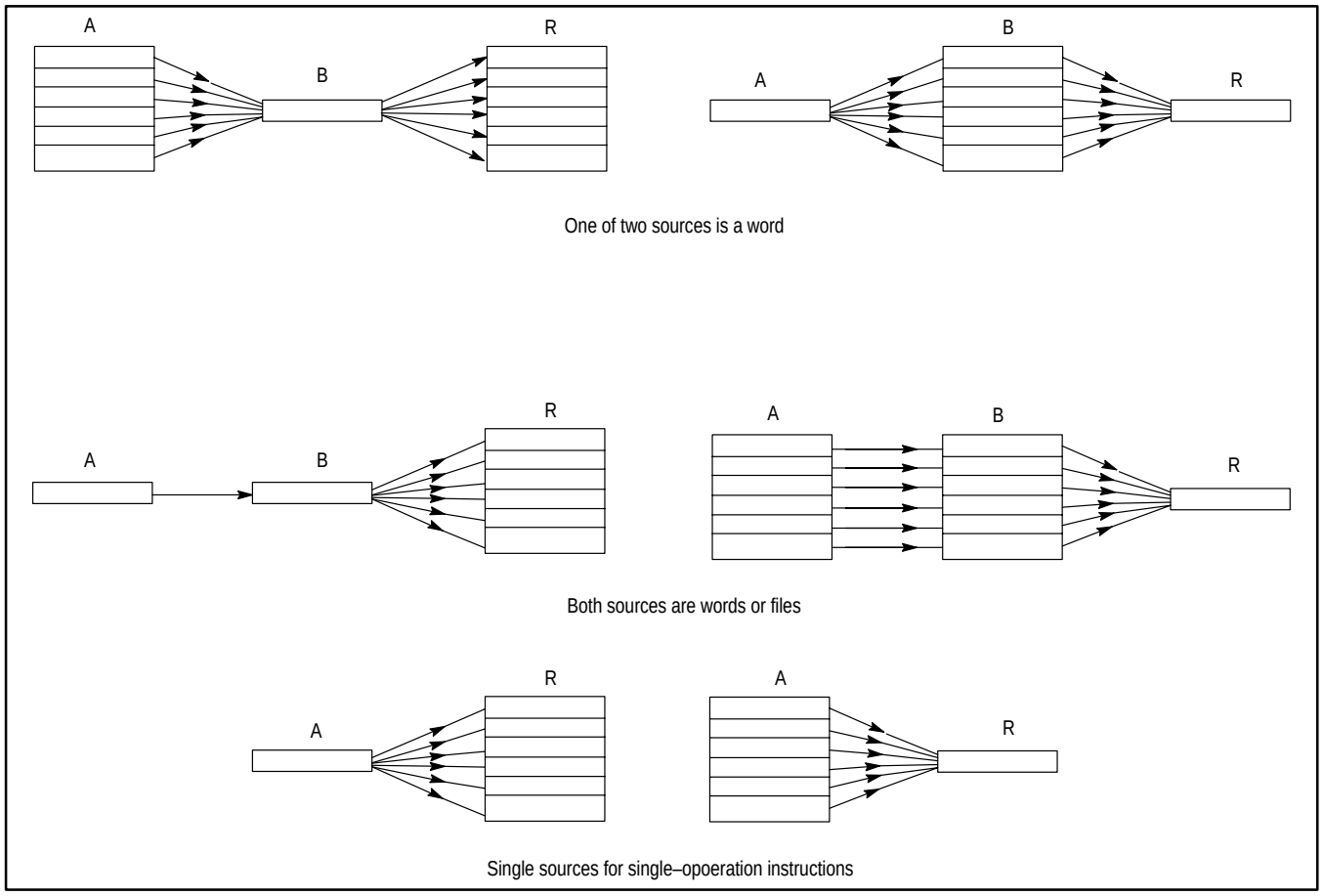
8.0 Chapter Objectives

In the preceding two chapters, we described data manipulation instructions and introduced you to the concept of files. This chapter describes data manipulation instructions that you can use with files.

8.1 Data Manipulation with Files

With files, you can use data manipulation instructions to manipulate entire files or portions of files between data table sections (Figure 8.1).

Figure 8.1
Manipulating Entire or Portions of Files



You specify the files or words to be manipulated by entering the appropriate data table addresses as sources for the file instruction as we described in chapter 7:

If you want to access a	Then enter this delimiter before the address
single word	W (word)
group words	F (file)

If you do not enter a delimiter, the processor defaults to the word delimiter.

Important: If you specify a word address as a source for a file instruction, the processor reads that word only when it first executes the instruction.

Figure 8.2 shows the data table map with file organization for the various data table sections. Table 8.A lists the section specifiers, data types, and value ranges for each section.

We group the data manipulation instructions for files into four types:

- file-data-transfer instructions
- file-data-comparison instructions
- file-arithmetic instructions
- file-logic instructions

We describe these types in the rest of this chapter.

Figure 8.2
Data Table Map

Section Number	Title	Maximum Size	Address Range
1	Output Image Table	4,096 values	$O0000_8$ to $O0377_8$
2	Input Image Table	4,096 values	$I0000_8$ to $I0377_8$
3	Timer Table (3 words/timer)	10,000 timers	T0 to T9999
4	Counter Table (3 words/counter)	10,000 counters	C0 to C9999
5	Integer Table (1 word/value)	10,000 values	N000:0000 to N000:9999
6	Floating-Point Table (2 word/value)	10,000 values	F000:0000 to F000:9999
7	Decimal Table (1 word/ 4BCD values)	10,000 values	D000:0000 to D000:9999
8	Binary Table (1 word/value)	10,000 values	B000:0000 to B000:9999
9	ASCII Table (2 characters/word)	20,000 characters	A000:0000 to A000:9999
10	High–order–integer Table (2 words/value)	10,000 values	H000:0000 to H000:9999
12	Pointer Table	10,000 addresses	P000:0000 to P000:9999
13	Status Table	10,000 values	S000:0000 to S000:9999

Table 8.A
Section Specifiers, Data Types, and Acceptable Ranges for Values Stored in the Data Table

Data Table Section	Section Specifier	Type of Data Stored in Section	Range	
			Low Limit	High Limit
Output image	O	Unsigned binary	0	65,535
Input image	I	Unsigned binary	0	65,535
Timer	T	Unsigned binary	0	65,535
Counter	C	Binary[1]	-32,768	32,767
Integer	N	Binary[1]	-32,768	32,767
Floating point	F	Floating point	±2.939 E-39	±1.701 E+38
Decimal	D	Binary coded decimal	0	9,999
Binary	B	Unsigned binary	0	65,535
ASCII[2]	A	Unsigned binary	– – – – –	– – – – –
High order integer	H	Binary[1]	-2,147,483,648	2,147,483,647
Pointer	P	Unsigned binary[3]	– – – – –	– – – – –
Status	S	Unsigned binary[3]	– – – – –	– – – – –

[1]The processor stores positive numbers in straight binary and negative numbers in two's complement form.

[2]The ASCII table can store ASCII characters as defined by ASCII (ANSI X3.4).

[3]The processor treats data in the pointer and status sections as unsigned binary, although these sections are intended to store non-numeric data.

8.2 File-data-transfer Instructions

File-data-transfer instructions move a file or a portion of a file from one location to another. In programming these instruction, you specify addresses that tell the source and destination for data that you want to move:

- Source address (A) tells the processor where to read the data.
- Destination address (R) tells the processor where to transfer the data.

When the processor executes a rung containing a file-data-transfer instruction:

If a false-to-true rung transition	Then the processor
occurs	executes the instruction and copies the data from the source to the destination
does not occur	does not copy the data; the destination retains its last value

If you program the transfer of data between data table sections with different data types, the processor automatically converts the data into the proper type. For example, if the processor moves a file from the binary to the decimal section, it automatically converts binary to BCD.

However, the processor does not convert the data to and from the input or output sections. To convert the data, you can first transfer the data from the input or output section into another section. For example, if the data is BCD, you can transfer the data into the decimal section. Then transfer the data from the decimal section to the desired section..

If a value in the source or destination address is not a storable data type or not within the limits of the corresponding data table section, the processor sets:

- file error bit (bit 13) in the counter control word that controls the file instruction
- instruction fault (bit 17) and either the overflow (bit 13), underflow (bit 12), or conversion fault (bit 10) bits in the status file 0, word 0 (refer to chapter 14).

You can use the following file-data-transfer instructions:

- file move
- file move with mask

8.2.1 File Move (MVF)

Required Parameters: Source (A) and destination (R) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-move instruction goes from false to true, the processor copies the information from the source to the destination. Operating like the move instruction, the file-move instruction allows you to program the processor to transfer data from file to file, file to word, and word to file. You can enter data into word locations when loading the instruction by using the data monitor. Refer to the PLC-3 Industrial Terminal User's Manual (publication 1770-6.5.15) for detailed information.

Examples: Figure 8.3 shows a rung that executes a file-to-file move with both files starting at the same word number.

If the rung goes from false to true, the processor copies the data from the source (integer file 3 starting at word 0) to the destination (integer file 4 starting at word 0). In this file-move instruction:

This parameter	Tells the processor
counter (C24)	what counter controls file instruction operation
file position (POS=0)	to start at the first word (word 0) in integer file 3
file length (LEN=6)	to transfer 6 words
mode (ALL/SCAN)	to execute the entire file operation in one program scan

Figure 8.3
Example Rung for a File-to-File Move Instruction

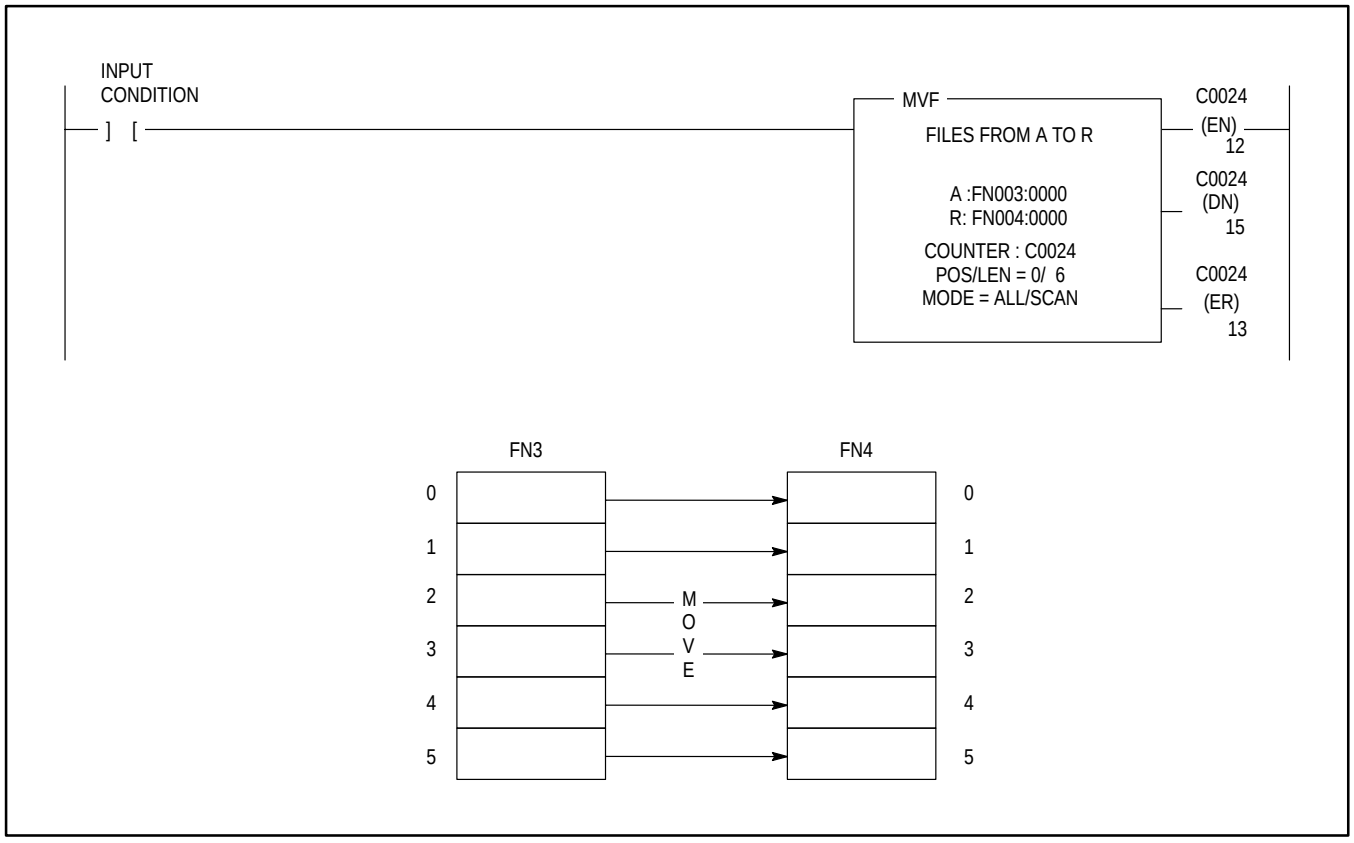


Figure 8.4 shows a rung that executes a file-to-file move with files starting at different word numbers.

If the rung goes from false to true, the processor copies integer file 3 word by word starting at word 3 and places the copy in integer file 4 starting at word 1. In this file-move instruction:

This parameter	Tells the processor
counter (C24)	what counter controls file instruction operation
file position (POS=0)	to start the move at word 3 in integer file 3
file length (LEN=6)	to transfer 6 words
mode (ALL/SCAN)	to execute the entire file operation in one program scan

Figure 8.4
Example Rung for a File-to-File Move Instruction

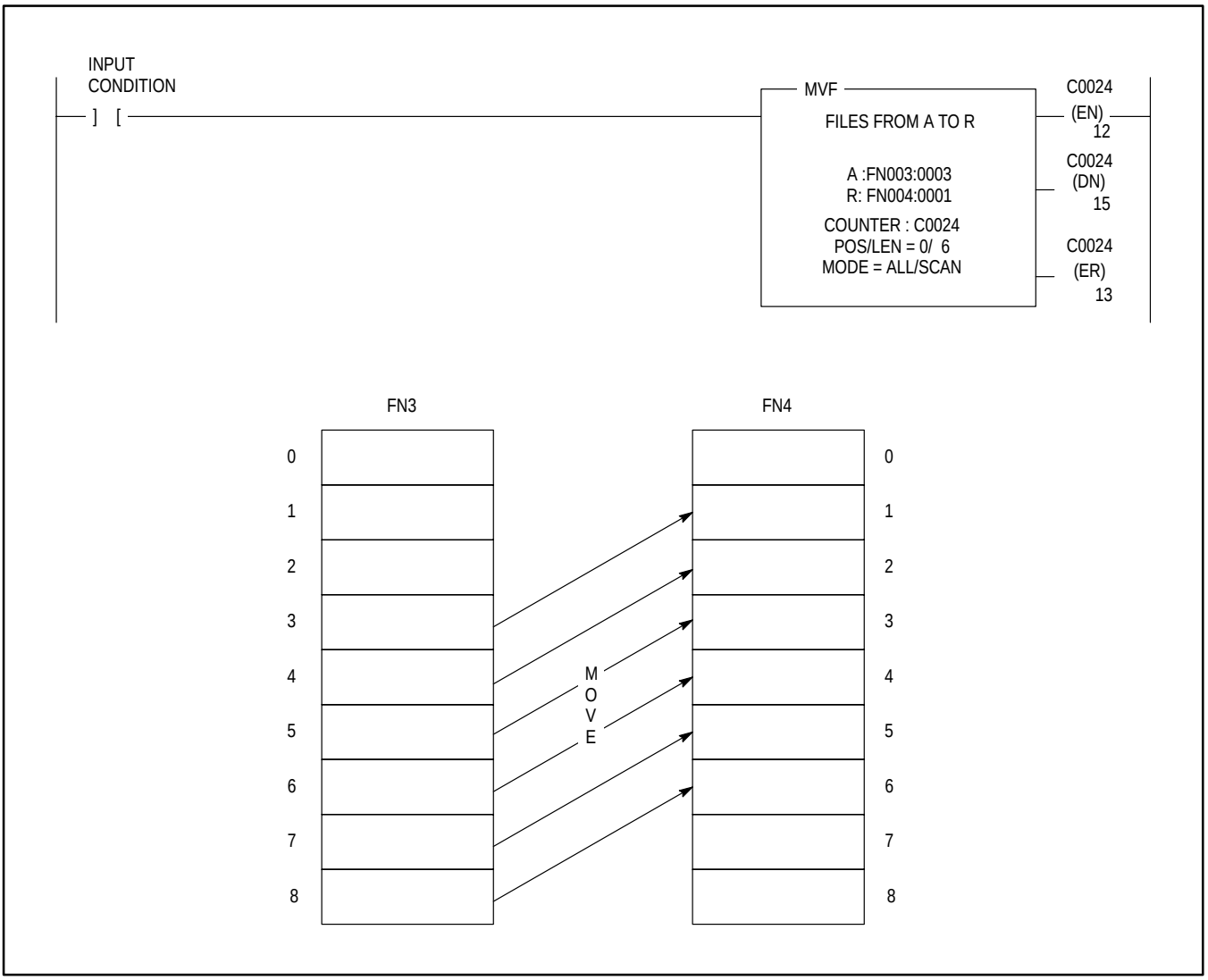


Figure 8.5 shows a rung that executes a word-to-file move.

If the rung goes from false to true, the processor copies integer file 3, word 2 and places the copy in integer file 4 starting at word 2. In this file-move instruction:

This parameter	Tells the processor
counter (C24)	what counter controls file instruction operation
file position (POS=0)	to start the move at word 2 in integer file 3
file length (LEN=6)	to transfer the source word into 6 words of the destination file
mode (ALL/SCAN)	to execute the entire file operation in one program scan

Figure 8.5
Example Rung for a Word-to-File Move Instruction

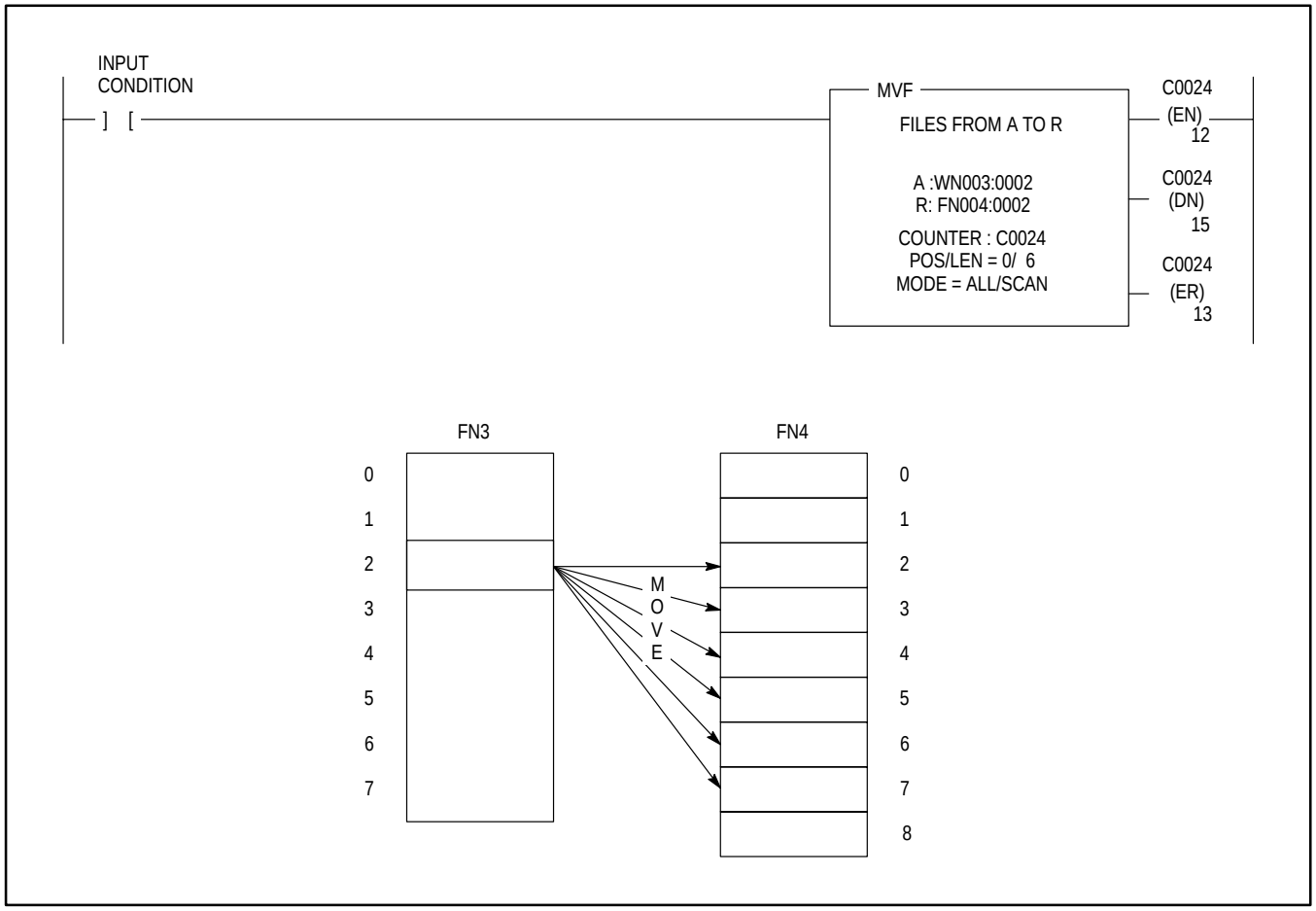


Figure 8.6 shows a rung that executes a file-to-word move.

If the rung goes from false to true, the processor copies one word from integer file 3 per program scan and places the copy in integer rile 4 word 5. In this file-move instruction:

This parameter	Tells the processor
counter (C24)	what counter controls file instruction operation
file position (POS=0)	to start the first word (word 0) in integer file 3
file length (LEN=6)	to transfer 6 words
mode (1/SCAN)	to execute the file operation on one word per program scan

Important: For a file-to-word-data transfer, you can use increment mode, none mode, or numeric mode with the rate per scan set to one. You cannot use all mode.

Figure 8.6
Example Rung for a File-to-Word Move Instruction

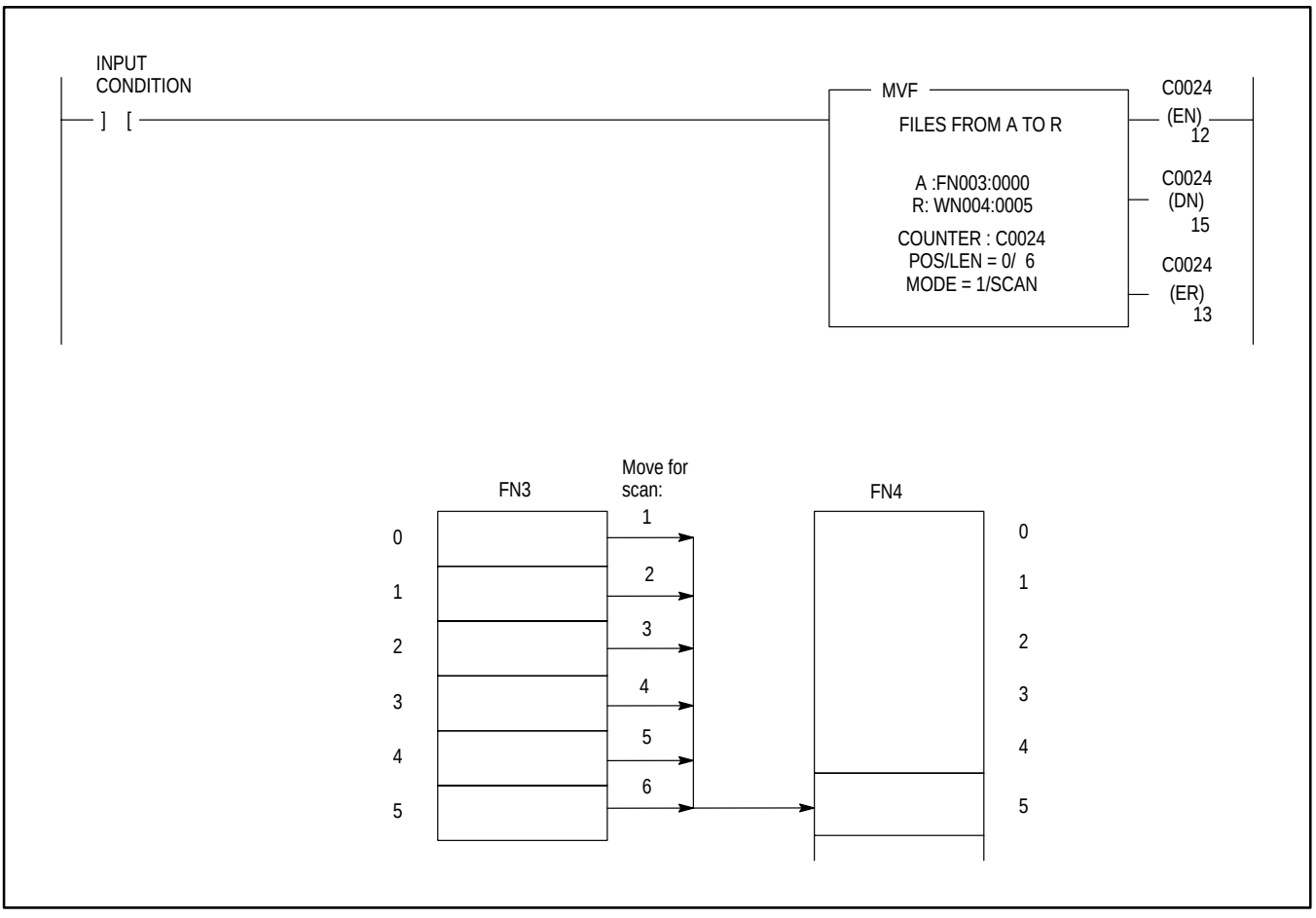
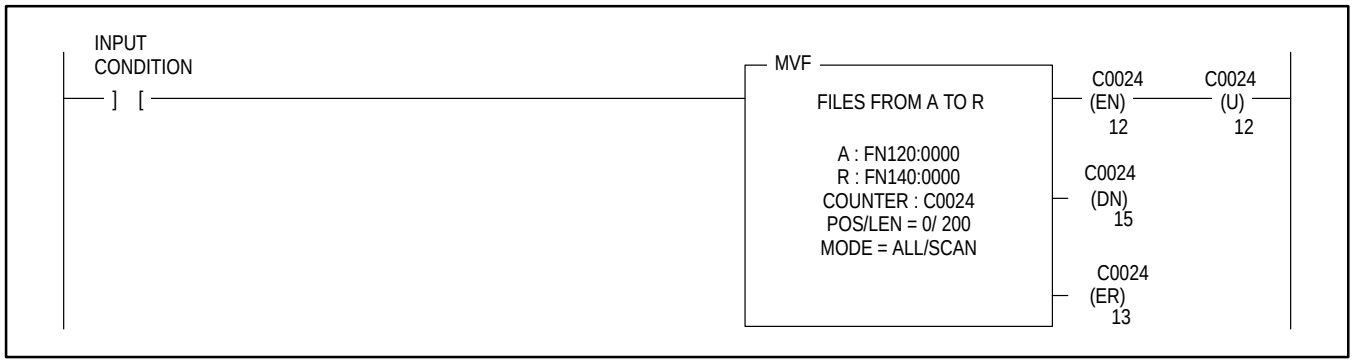


Figure 8.7 shows a rung that executes a file-move instruction continuously.

If the rung goes from false to true, the processor copies 200 words of integer file 120 and places the copy in integer file 140. The rung also unlatches the file enable bits to that the next time the processor scans the rung, it executes the file instruction if the rung remains true.

Figure 8.7
Example Rung for Executing a File-move Instruction Continuously



8.2.2 File Move with Mask (MMF)

Required Parameters: Source (A), mask (B), and destination (R) addresses, counter number, starting word (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-move-with-mask instruction goes from false to true, the processor copies the information from the source through the mask to the destination. Only bits in the source that correspond to set bits in the mask move to the destination.

Operating similarly to the move instruction, the file-move-with-mask instruction allows you to program data to mask data through a separate word or file. Data formats for source (A) and the destination should be the same. Source (B) should be binary data.

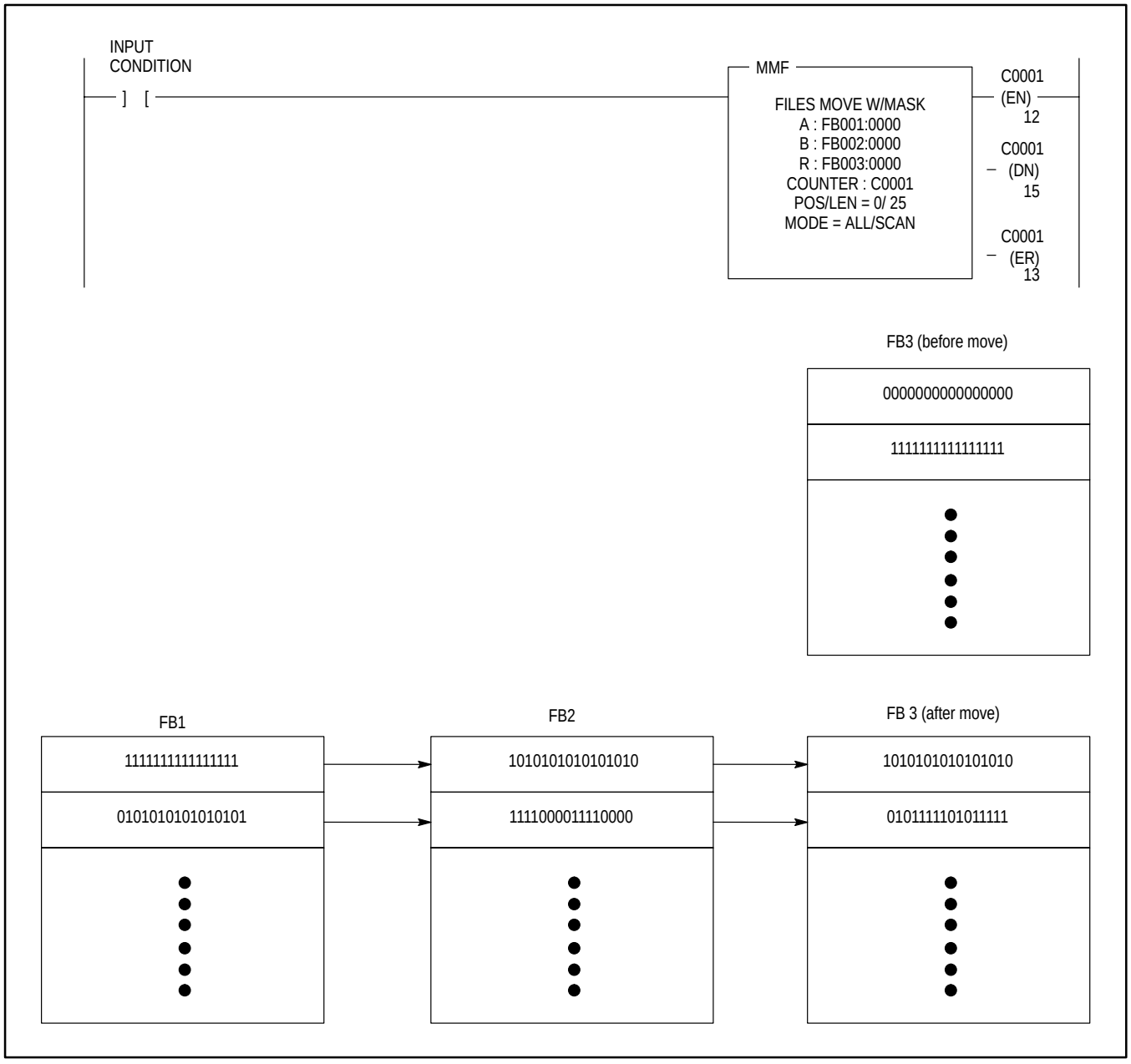
To pass data through the mask, you must set bits in the mask word or file. To load values into source or mask file locations, you can enter the data upon loading the instruction or by using the data monitor. Refer to the PLC-3 Industrial Terminal User's Manual (publication 1770-6.5.15) for detailed information on these methods.

Example: Figure 8.8 shows a rung containing a file-move-with-mask instruction.

If the rung goes from false to true, the processor takes the bits from the words in the source (binary file 1) that corresponds to set bits in the words in the mask (binary file 2), and puts the results in the destination (binary file 3). In this file move instruction:

This parameter	Tells the processor
counter (C1)	what counter controls data-transfer operation
file position (POS=0)	to start at the first word (word 0) in binary file 1
file length (LEN=25)	to transfer 25 words
mode (ALL/SCAN)	to execute the entire file operation in one program scan

Figure 8.8
Example Rung for a File-move-with-mask Instruction



8.3 File-data-comparison Instructions

File-data-comparison instructions are output instructions that compare extensive numerical data in files. These instructions can operate on any of the data types stored in the data table. The upper and lower limits of the data being operated on depend on the section where the data are stored.

If a false-to-true rung transition occurs on a rung containing a file-data-comparison instruction, the processor executes the desired file comparison instruction on the data in the sources. In executing the instruction, the processor sets the file enable (bit 12) and compares the first word in source (A) to the first word in source (B). If the comparison condition is:

- true, the processor sets the file done (bit 5) and the file found (bit 10) in the counter control word. The position (POS) value indicates the word that met the condition. To continue comparing you must program a rung to unlatch the done bit. By unlatching the done bit, the processor also resets the found bit. It continues comparing until the next true comparison or the end of the file. The end of file is reached when the done bit is set without the found bit being set.

- false, the processor compares the second words in the sources (A and B) at a scan rate determined by the mode of operation. If the comparison condition is not met for any words in the file, the processor sets the file done (bit 15) when the position (POS) equals the length (LEN).

Upon completing the comparison between the files, the processor sets the done bit. The done bit remains set until the rung becomes false.

When file-data-comparison instructions compare data between data table section with different numeric formats, the processor automatically converts the data into its binary equivalent, then compares the data.

To program data-comparison instructions, you can use the following instructions:

- search equal
- search not equal to
- search less than
- search less than or equal
- search greater than
- search greater than or equal

8.3.1 Search Equal (SEQ)

Required Parameters: Sources (A and B) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a search-equal instruction goes from false to true, the processor executes an equal-to comparison operation between the file words specified as the sources. The number of words compared per program scan is determined by the selected mode of operation.

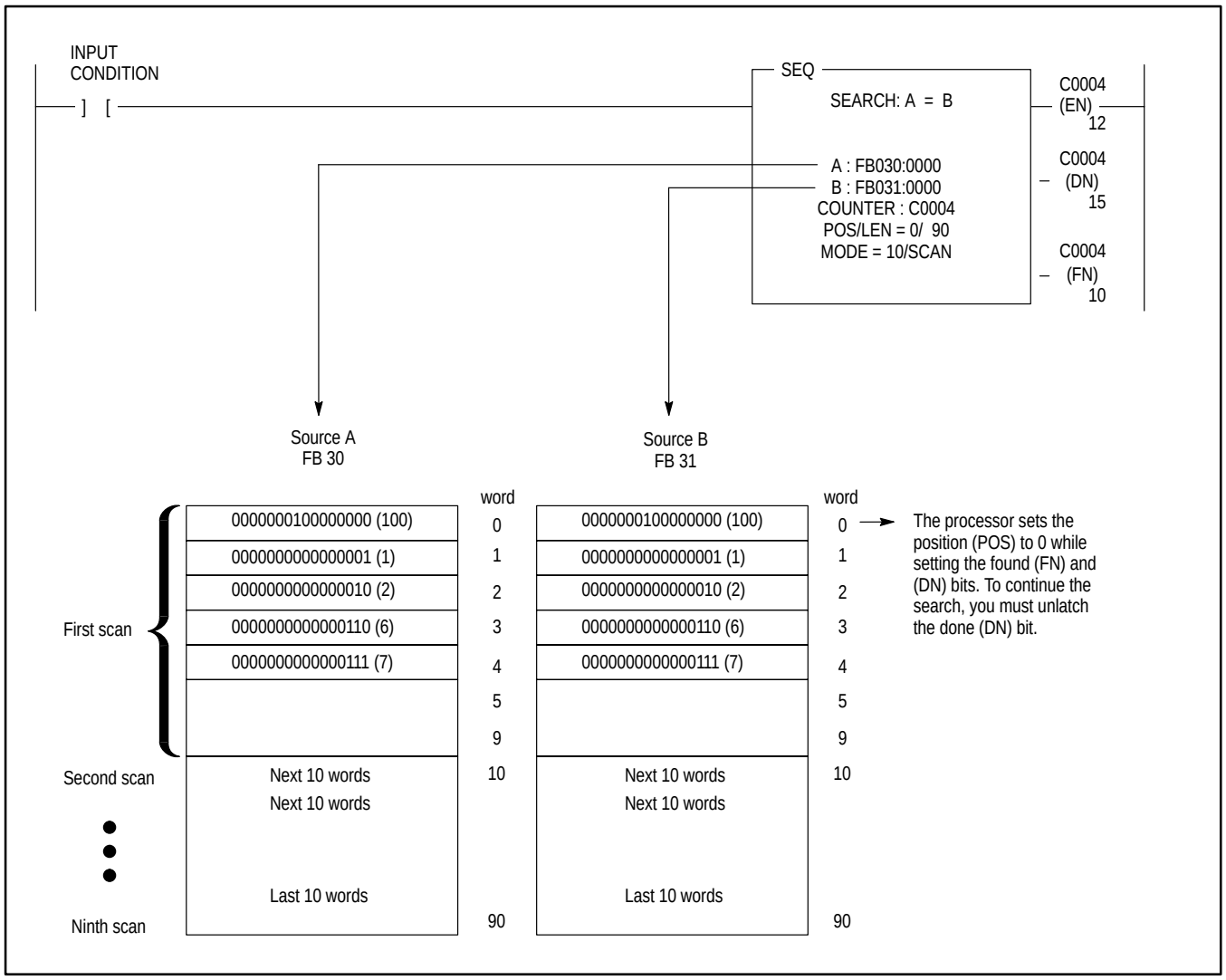
When the processor finds that a value within the source (A) file equals the corresponding value within the source (B) file, it sets the found and done bits. To continue the search equal comparison, you must unlatch the done bit.

Example: Figure 8.9 shows a rung containing a search-equal instruction.

If the rung goes from false to true, the processor executes an equal to operation between the source files (binary files 30 and 31). In this search-equal instruction:

This parameter	Tells the processor
counter (C4)	what counter controls data-transfer operation
file position (POS=0)	to start at the first word in the binary files
file length (LEN=90)	to compare 90 words (0-89)
mode (10/SCAN)	to execute the file operation on ten words per program scan

Figure 8.9
Example Rung for a Search-equal Instruction



8.3.2 Search Not Equal (SNE)

Required Parameters: Sources (A and B) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a search-not-equal instruction goes from false to true, the processor executes a not-equal-to comparison operation between the file words specified as the sources. The number of words compared per program scan is determined by the selected mode of operation.

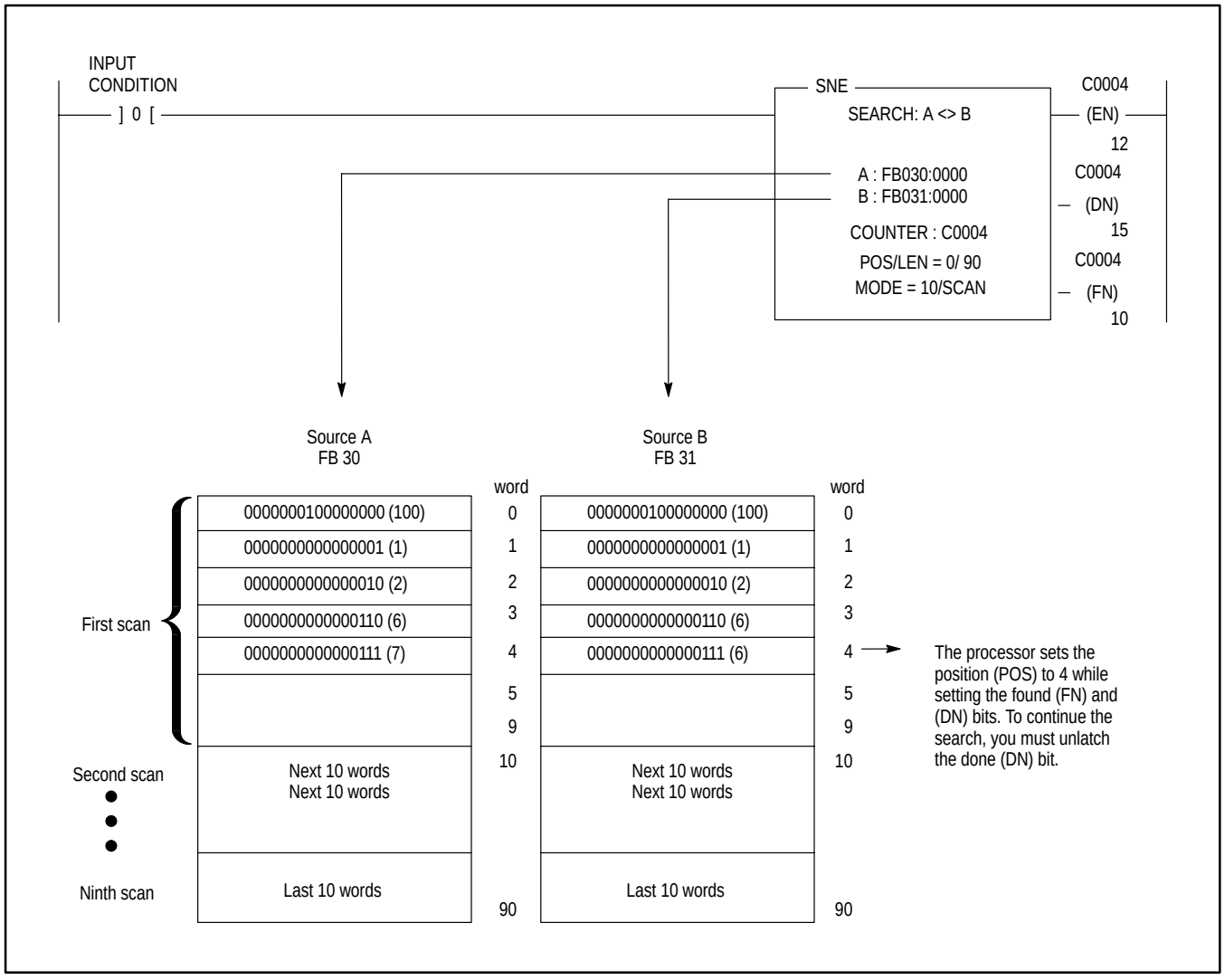
When the processor finds that a value within the source (A) file does not equal the corresponding value within the source (B) file, it sets the found and done bits. To continue the search-not-equal comparison, you must unlatch the done bit.

Example: Figure 8.10 shows a rung containing a search-not-equal instruction.

If the rung goes from false to true, the processor executes a not-equal-to operation between the source files (binary files 30 and 31). In this search-not-equal instruction:

This parameter	Tells the processor
counter (C4)	what counter controls data-transfer operation
file position (POS=0)	to start at the first word in the binary files
file length (LEN=90)	to compare 90 words
mode (10/SCAN)	to execute the file operation on 10 words per program scan

Figure 8.10
Example Rung for a Search-not-equal Instruction



8.3.3 Search Less Than (SLS)

Required Parameters: Sources (A and B) addresses, counter number, starting word (POS), length of file (LEN), mode of operation.

Description: When a rung containing a search-less-than instruction goes from false to true, the processor executes a less-than comparison operation between the file words specified as the sources. The number of words compared per program scan is determined by the selected mode of operation.

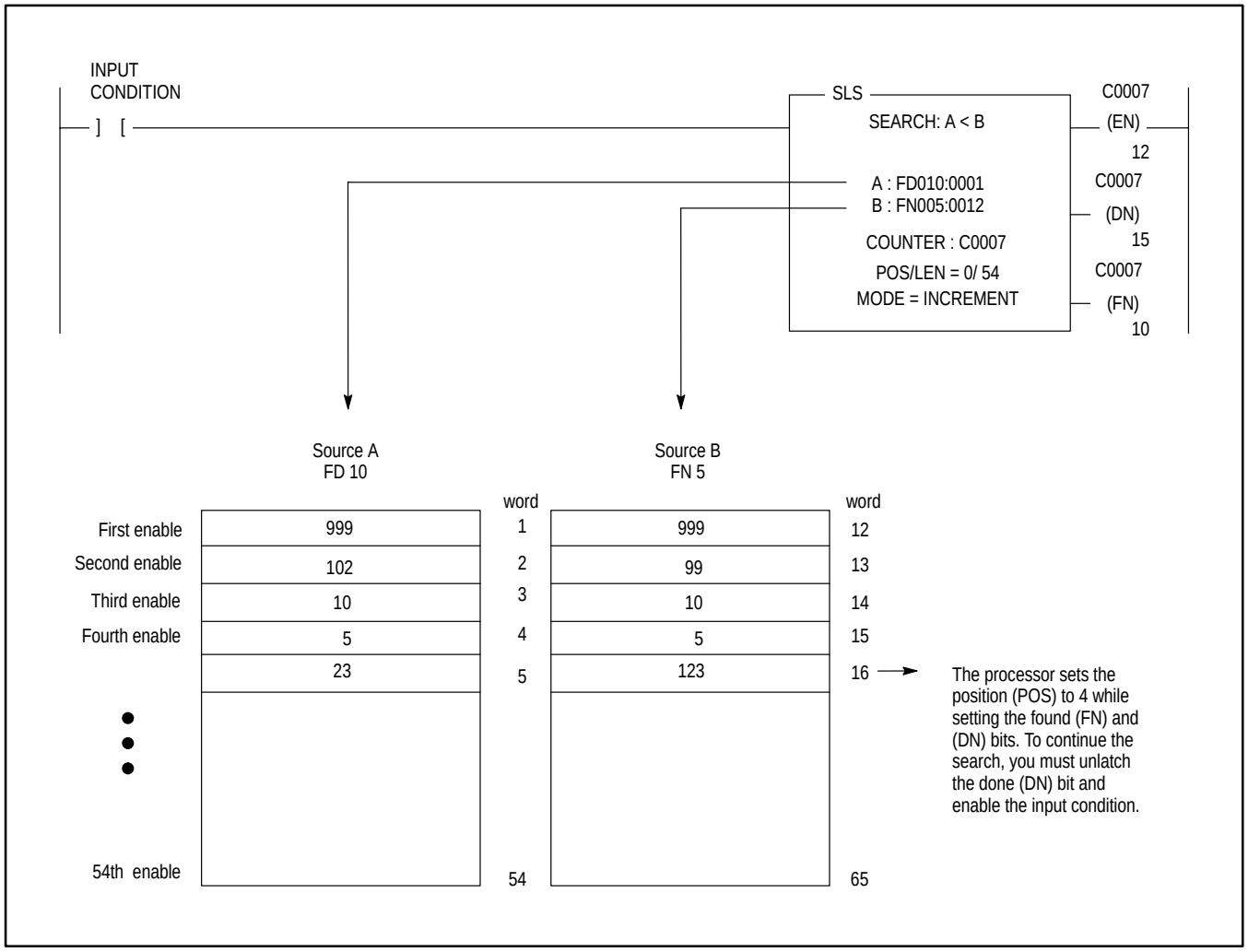
When the processor finds that a value within the source (A) file is less than the corresponding value within the source (B) file, it sets the found and done bits. To continue the search-less-than comparison, you must unlatch the done bit.

Example: Figure 8.11 shows a rung containing a search-less-than instruction.

If the rung goes from false to true, the processor executes a less-than operation between the source files (decimal file 10 starting a word 1 and integer file 5 starting at word 12). In this search-less-than instruction:

This parameter	Tells the processor
counter (C7)	what counter controls data-transfer operation
file position (POS=0)	to start at the first word in the files (word 1 in decimal file 10 and word 12 in integer file 5)
file length (LEN=54)	to compare 54 words
mode (INCREMENT)	to execute the file operation on one word per program scan each time that the rung goes from false to true

Figure 8.11
Example Rung for a Search-less-than Instruction



8.3.4 Search Less Than or Equal (SLE)

Required Parameters: Sources (A and B) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a search-less-than-or-equal instruction goes from false to true, the processor executes a less-than-or-equal-to comparison operation between the file words specified as the source. The number of words compared per program scan is determined by the selected mode of operation.

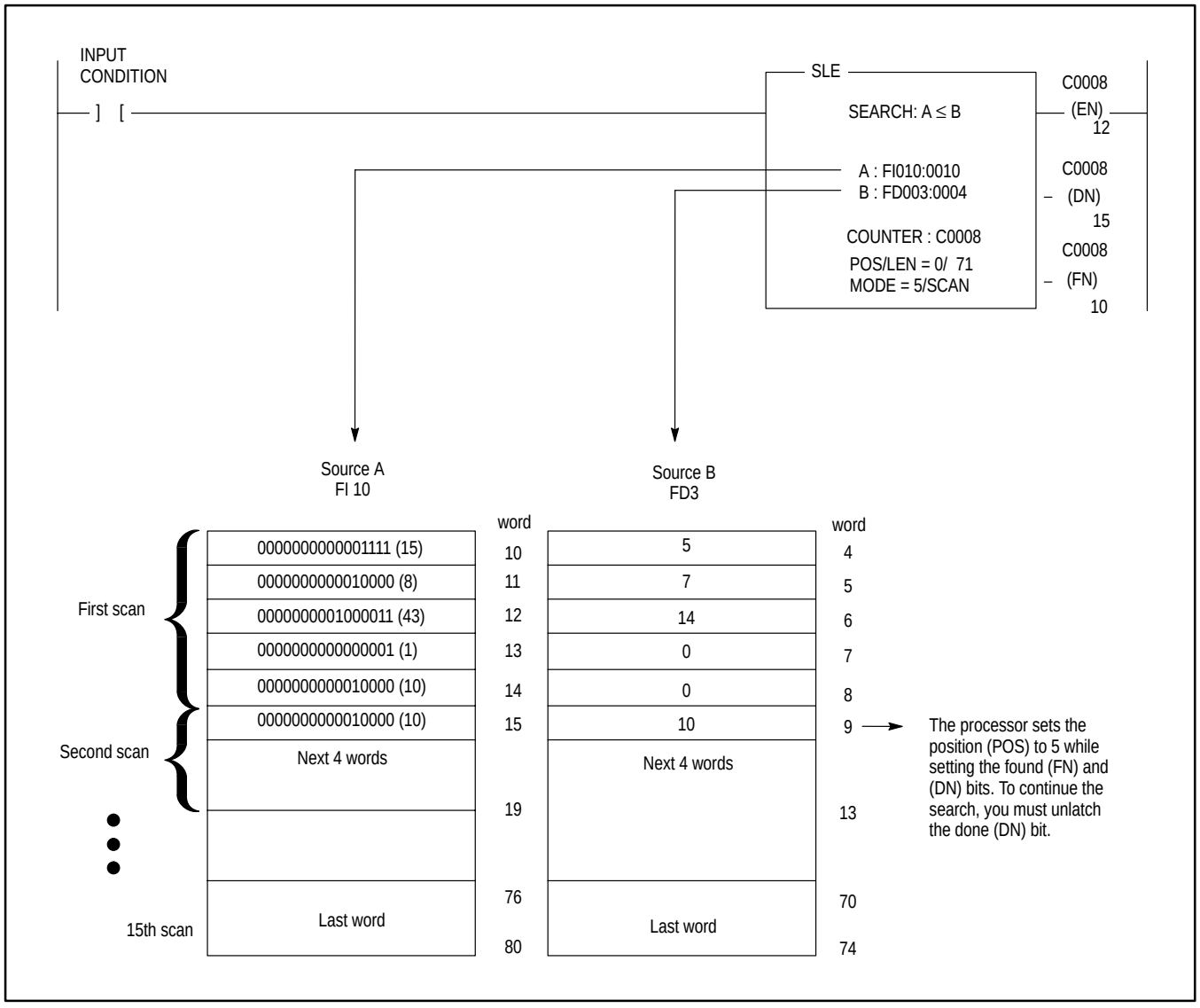
When the processor finds that a value within the source (A) file is less than or equal to the corresponding value within the source (B) file, it sets the found and done bits. To continue the search-less-than-or-equal-to comparison, you must unlatch the done bit.

Example: Figure 8.12 shows a rung containing a search-less-than-or-equal instruction.

If the rung goes from false to true, the processor executes a less-than–or-equal-to operation between the source files (input file 10 starting at word 10 and decimal file 3 starting at word 4). In this search less-than-or-equal instruction:

This parameter	Tells the processor
counter (C8)	what counter controls data-transfer operation
file position (POS=0)	to start at the first word in the files (word 10 in integer file 10 and word 4 in decimal file 3)
file length (LEN=71)	to compare 71 words
mode (5/SCAN)	to execute the file operation on five words per program scan

Figure 8.12
Example Rung for a Search-less-than-or-equal Instruction



8.3.5 Search Greater Than (SGR)

Required Parameters: Sources (A and B) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a search-greater-than instruction goes from false to true, the processor executes a greater-than comparison operation between the file words specified as the sources. The number of words compared per program scan is determined by the selected mode of operation.

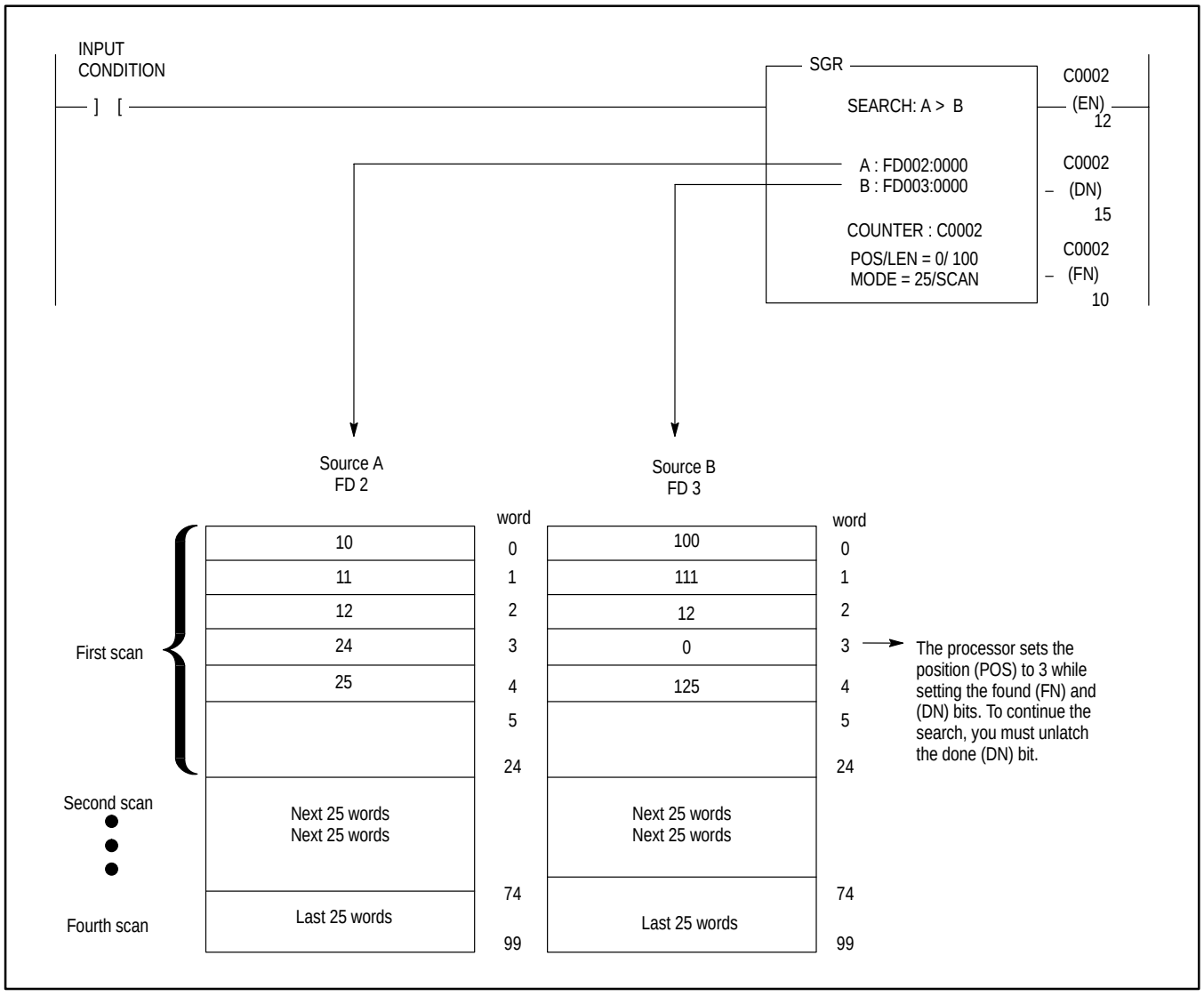
When the processor finds that a value within the source (A) file is greater than the corresponding value within the source (B) file, it sets the found and one bits. To continue the search-greater-than comparison, you must unlatch the done bit.

Example: Figure 8.13 shows a rung containing a search-greater-than instruction.

If the rung goes from false to true, the processor executes a greater-than operation between the source files (decimal files 2 and 3). In this search-greater-than instruction:

This parameter	Tells the processor
counter (C2)	what counter controls data-transfer operation
file position (POS=0)	to start at the first word in the decimal files
file length (LEN=100)	to compare 100 words
mode (25/SCAN)	to execute the file operation on 25 words per program scan

Figure 8.13
Example Rung for a Search-greater-than Instruction



8.3.6 Search Greater Than or Equal (SGE)

Required Parameters: Source (A and B) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a search-greater-than-or-equal instruction goes from false to true, the processor executes a greater-than-or-equal-to comparison operation between the file words specified as the sources. The number of words compared per program scan is determined by the selected file mode of operation.

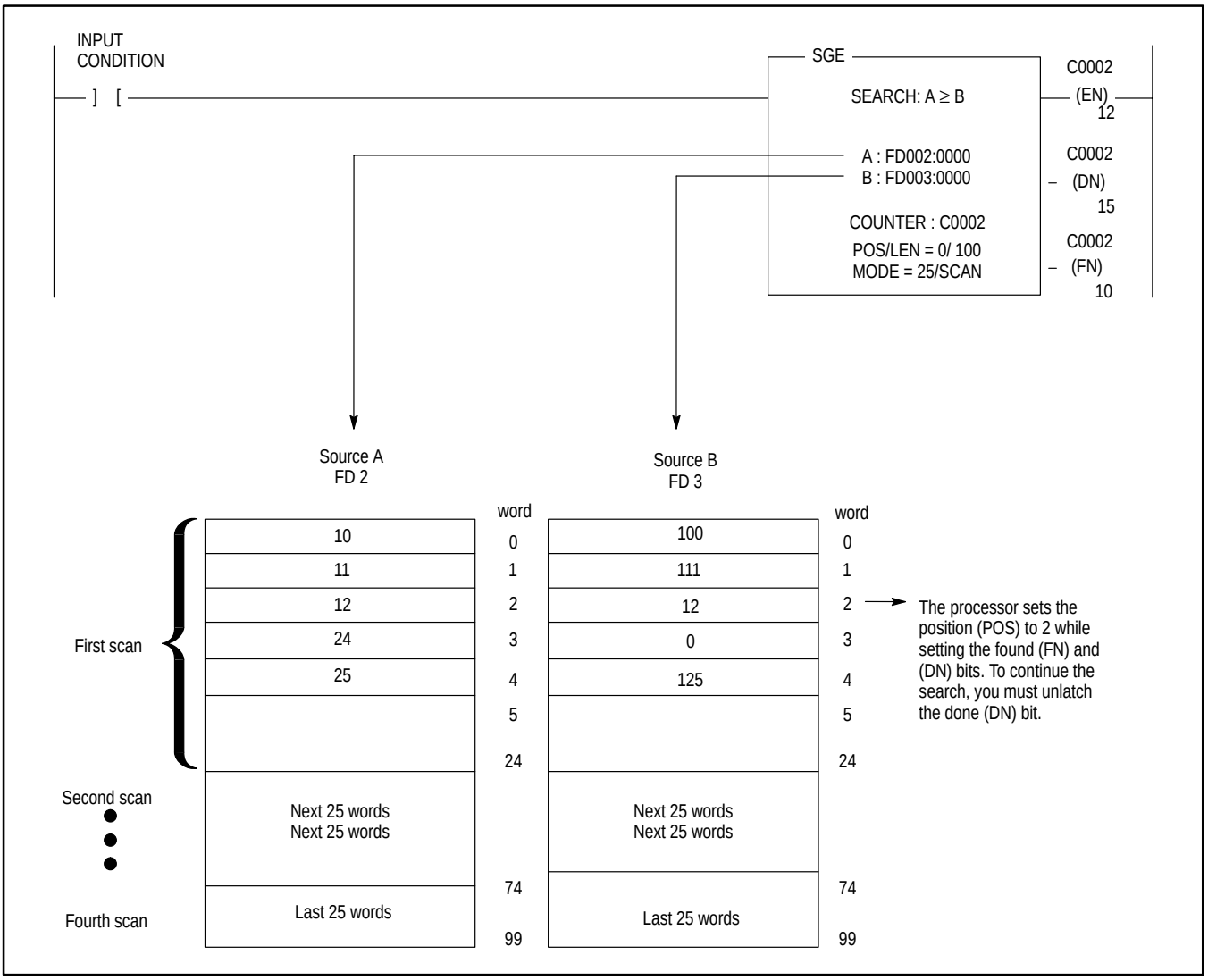
When the processor finds that a value within the source (A) file is greater-than or equal-to the corresponding value within the source (B) file, it sets the found and done bits. To continue the search-greater-than-or-equal-to comparison, you must unlatch the done bit.

Examples: Figure 8.14 shows a rung containing a search-greater-than-or-equal instruction.

If the rung goes from false to true, the processor executes a greater-than-or-equal-to operation between the source files (decimal files 2 and 3). In this search-greater-than or equal instruction:

This parameter	Tells the processor
counter (C2)	what counter controls data-transfer operation
file position (POS=0)	to start at the first word in the decimal files
file length (LEN=100)	to compare 100 words
mode (25/SCAN)	to execute the file operation on 25 words per program scan

Figure 8.14
Example Rung for a Search-greater-than-or-equal Instruction



8.4 File-arithmetic Instructions

File-arithmetic instructions are output instructions that operate on files or portions of files in the data table. The upper and lower limits of the data being operated on depend on the section where the data are stored.

Table 8.A lists the type of data stored and the range of values for the data table sections. If the result of an arithmetic instruction exceeds the limits of the data table section that stores the result, an error occurs. You can determine the specific error by monitoring the following bits in status file 0, word 0:

- arithmetic fault – bit 17
- arithmetic overflow – bit 13
- arithmetic underflow – bit 12

You may want to monitor these bits in your ladder program so that an arithmetic fault does not cause invalid data to be used. Refer to chapter 14 for detailed information on the status files.

When executing arithmetic instructions on integer values, if fractional values result in the final answer, the processor truncates or rounds the final result:

If the resultant remainder is	Then the processor
less than 0.5	truncates or drops the remainder
0.5 or greater	rounds the remainder to the next whole number

For example, the processor rounds 3.5 to 4 and truncates 3.2 to 3. If you do not want the processor to truncate or round the result, use values from the floating-point section of the data table.

You can use the following file arithmetic instructions:

- file add
- file subtract
- file multiply
- file divide
- file square root
- file negate

8.4.1 File Add (ADF)

Required parameters: Sources (A and B) and destination (R) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-add instruction goes from false to true, the processor adds the values in the file words specified as the sources and puts the results in the file words specified as the destination. If the rung is false, the processor does not execute the file-add instruction, and the destination file retains its last values.

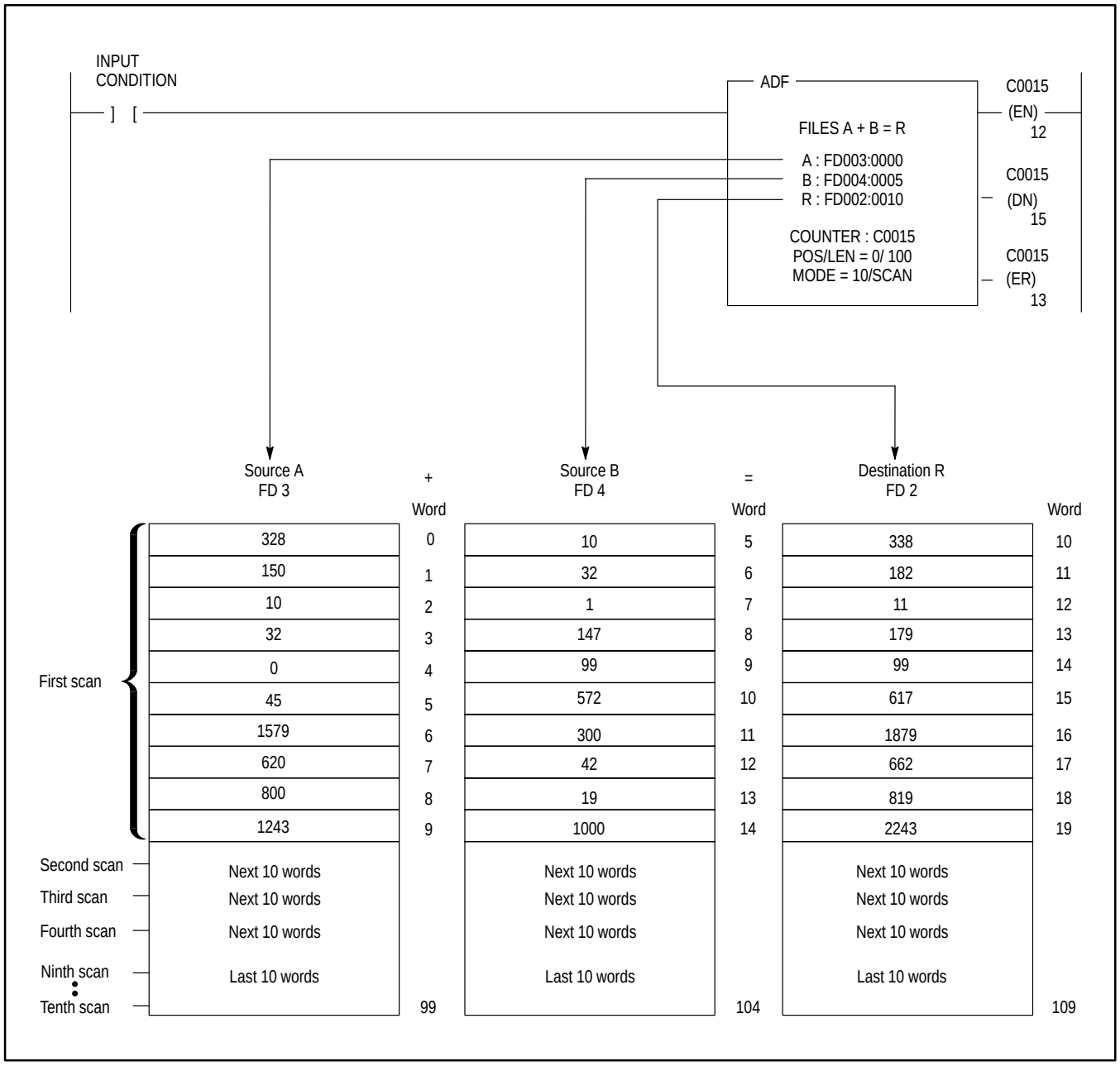
If the sources contain values with different numeric formats, the processor automatically converts the data. You can use the file-add instruction to add positive and negative values; however, if the result is negative, the destination address must be for a data table section that supports negative values. Otherwise an arithmetic fault occurs.

Example: Figure 8.15 shows a rung containing a file add instruction.

If the rung goes from false to true, the processor adds the word values in the sources (decimal files 3 and 4) and puts the results in the destination (decimal file 2). In this file-add instruction:

This parameter	Tells the processor
counter (C15)	what counter controls file-arithmetic operation
file position (POS=0)	to start at the first word (word 0 in decimal file 3 and word 5 in integer file 4)
file length (LEN=100)	to add 100 words
mode (10/SCAN)	to execute the file operation on 10 words per program scan

Figure 8.15
Example Rung for a File-add Instruction



8.4.2 File Subtract (SBF)

Required Parameters: Sources (A and B) and destination (R) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-subtract instruction goes from false to true, the processor subtracts the values in the file words specified as source (B) from the values in the file words specified as source (A) and puts the results in the file words specified as the destination. If the rung is false, the processor does not execute the file-subtract instruction, and the destination file retains its last values.

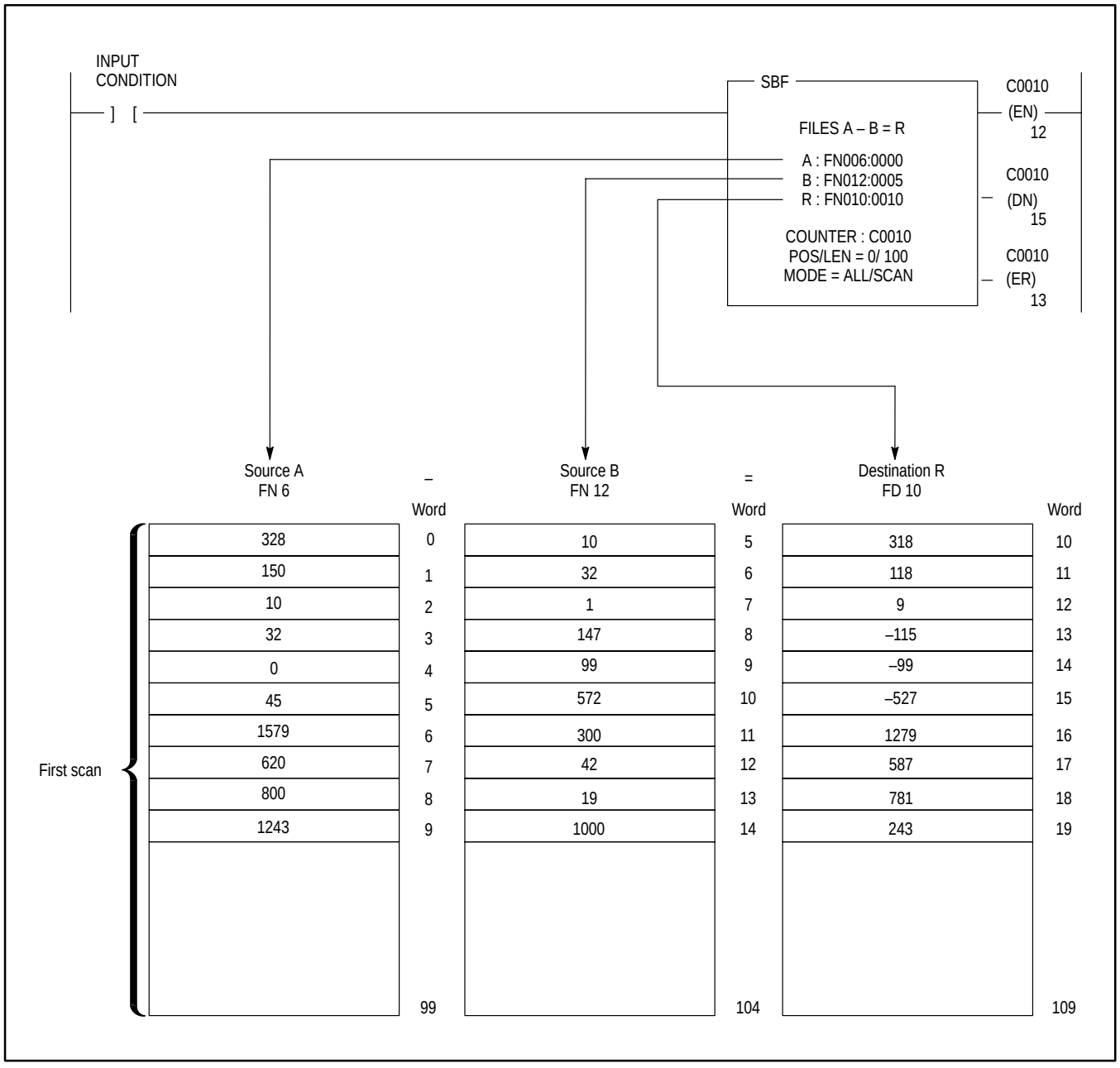
If the sources contain values with different numeric formats, the processor automatically converts the data. You can use the file-subtract instruction to subtract positive and negative values; however, if the result is negative, the destination address must be for a data table section that supports negative values. Otherwise an arithmetic fault occurs.

Example: Figure 8.16 shows a rung containing a file-subtract instruction.

If the rung goes from false to true, the processor subtracts the word values in source (B) (integer file 12 starting at word 5) from the word values in source (A) (integer file 6) and puts the results in the destination (integer file 10 starting at word 10). In this file-subtract instruction:

This parameter	Tells the processor
counter (C10)	what counter controls the file-arithmetic operation
file position (POS=0)	to start at the first word (word 0 in integer file 6 and word 5 in integer file 10)
file length (LEN=100)	to subtract 100 words
mode (ALL/SCAN)	to execute the entire file operation in one program scan

Figure 8.16
Example Rung for a File-subtract Instruction



8.4.3 File Multiply (MLF)

Required Parameters: Sources (A and B) and destination (R) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-multiply instruction goes from false to true, the processor multiplies the values in the file words specified as the sources and puts the results in the file words specified as the destination. If the rung is false, the processor does not execute the file-multiply instruction, and the destination file retains its last values.

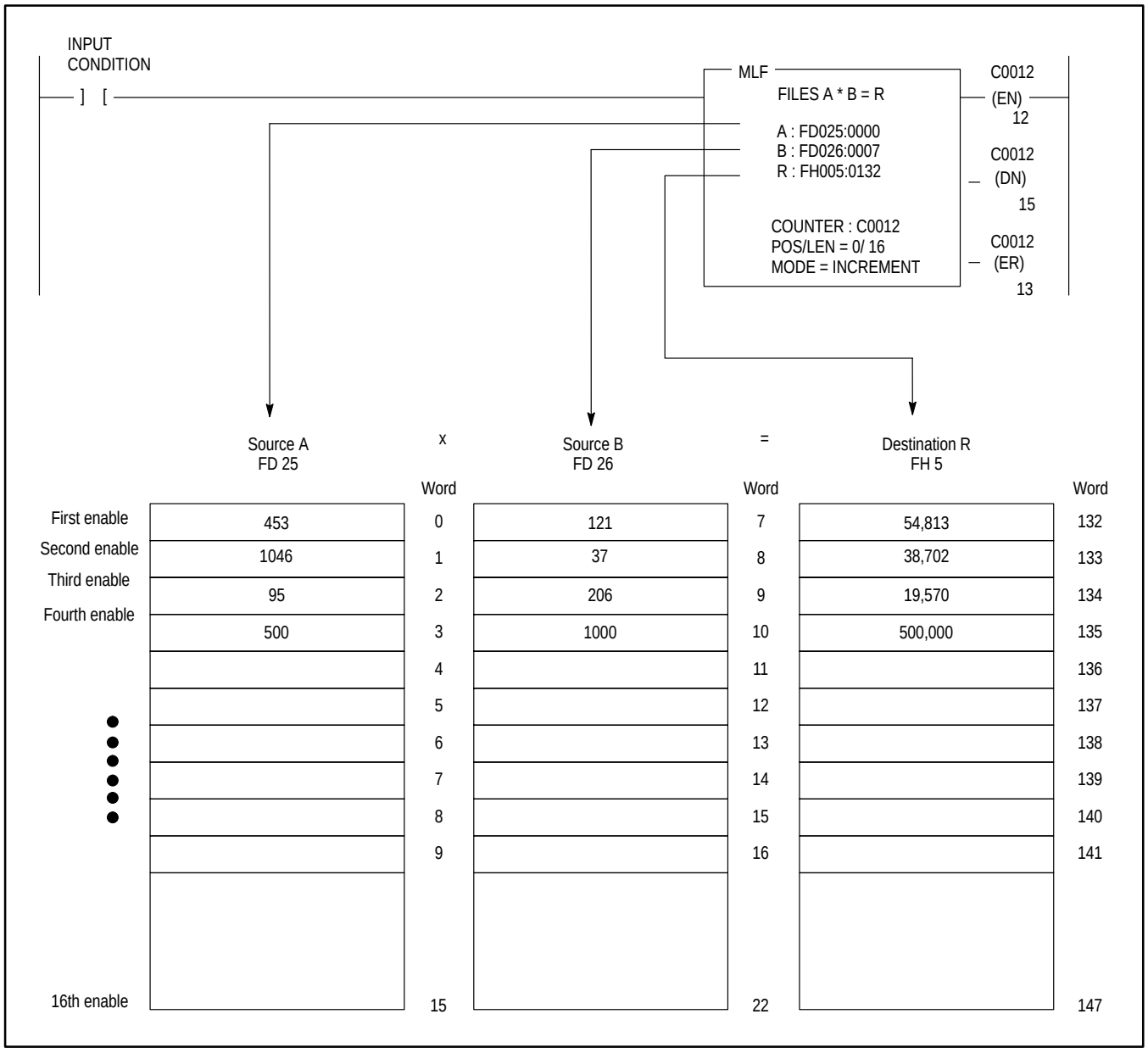
If the sources contain values with different numeric formats, the processor automatically converts the data. You can use the file-multiply instruction to multiply positive and negative values; however, if the result is negative, the destination address must be for a data table section that supports negative values. Otherwise an arithmetic fault occurs.

Example: Figure 8.17 shows a rung containing a file-multiply instruction.

If the rung goes from false to true, the processor multiplies the word values in the sources (decimal file 25 and decimal file 26 starting at word 7) and puts the results in the destination (high-order-integer file 5 starting at word 132). In this file multiply instruction:

This parameter	Tells the processor
counter (C12)	what counter controls file-arithmetic operation
file position (POS=0)	to start at the first word (word 0 in binary file 0 and word 7 in decmial file 26)
file length (LEN=16)	to multiply 16 words
mode (INCREMENT)	to execute the file operation on one word per program scan each time that the rung goes from false to true

Figure 8.17
Example Rung for a File-multiply Instruction



8.4.4 File Divide (DVF)

Required Parameters: Sources (A and B) and destination (R) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

description: When a rung containing a file-divide instruction goes from false to true, the processor divides the values in the file words specified as source (A) by the values in the file words specified by source (B) and puts the results in the file words specified as the destination. If the rung is false, the processor does not execute the file-divide instruction, and the destination file retains its last values.

If the sources contain values with different numeric formats, the processor automatically converts the data. You can use the file-divide instruction to divide positive and negative values; however, if the result is negative, the destination address must be for a data table section that supports negative values. Otherwise an arithmetic fault occurs.

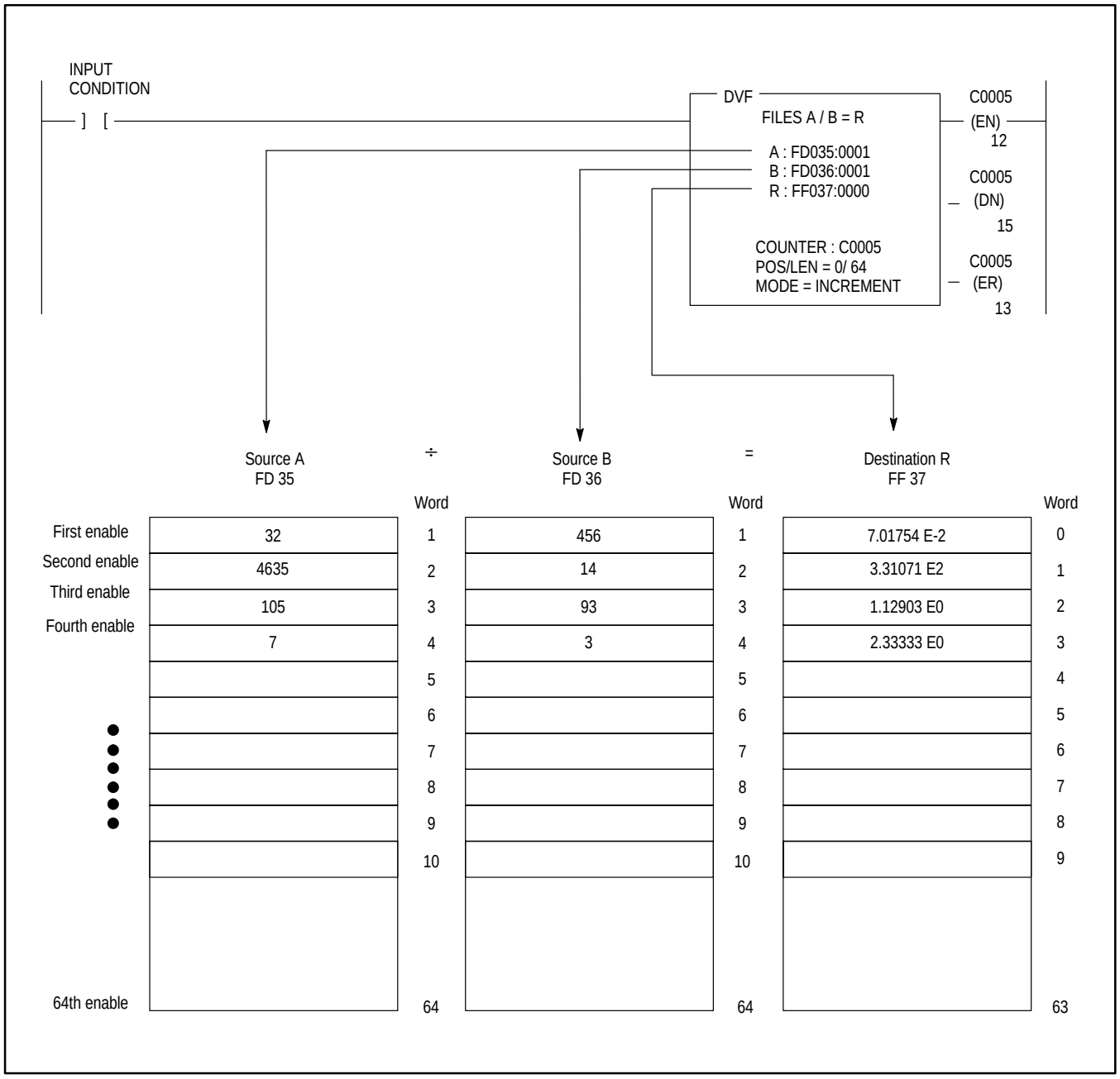
If source (B) contains 0, the processor declares an arithmetic fault and sets status bit 11 in status file 0, word 0 because you cannot divide a value by 0.

Example: Figure 8.18 shows a rung containing a file-divide instruction.

If the rung goes from false to true, the processor divides the word values in source (A) (decimal file 35 starting at word 1) by the word values in source (B) (decimal file 36 starting at word 1) and puts the results in the destination (floating point file 37 starting at word 0). In this file-divide instruction:

This parameter	Tells the processor
counter (C5)	what counter controls file-arithmetic operation
file position (POS=0)	to start at the first word (word 1) in decimal files 35 and 36
file length (LEN=64)	to divide 64 words
mode (INCREMENT)	to execute the file operation on one word per program scan each time that the rung goes from false to true

Figure 8.18
Example Rung for a File-divide Instruction



8.4.5 File Square Root (SQF)

Required Parameters: Source (A) and destination (R) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-square-root instruction goes from false to true, the processor takes the square root of the values in the file words specified as the source and puts the results in the file words specified as the destination. If the rung is false, the processor does not execute the file-square-root instruction, and the destination file retains its last values.

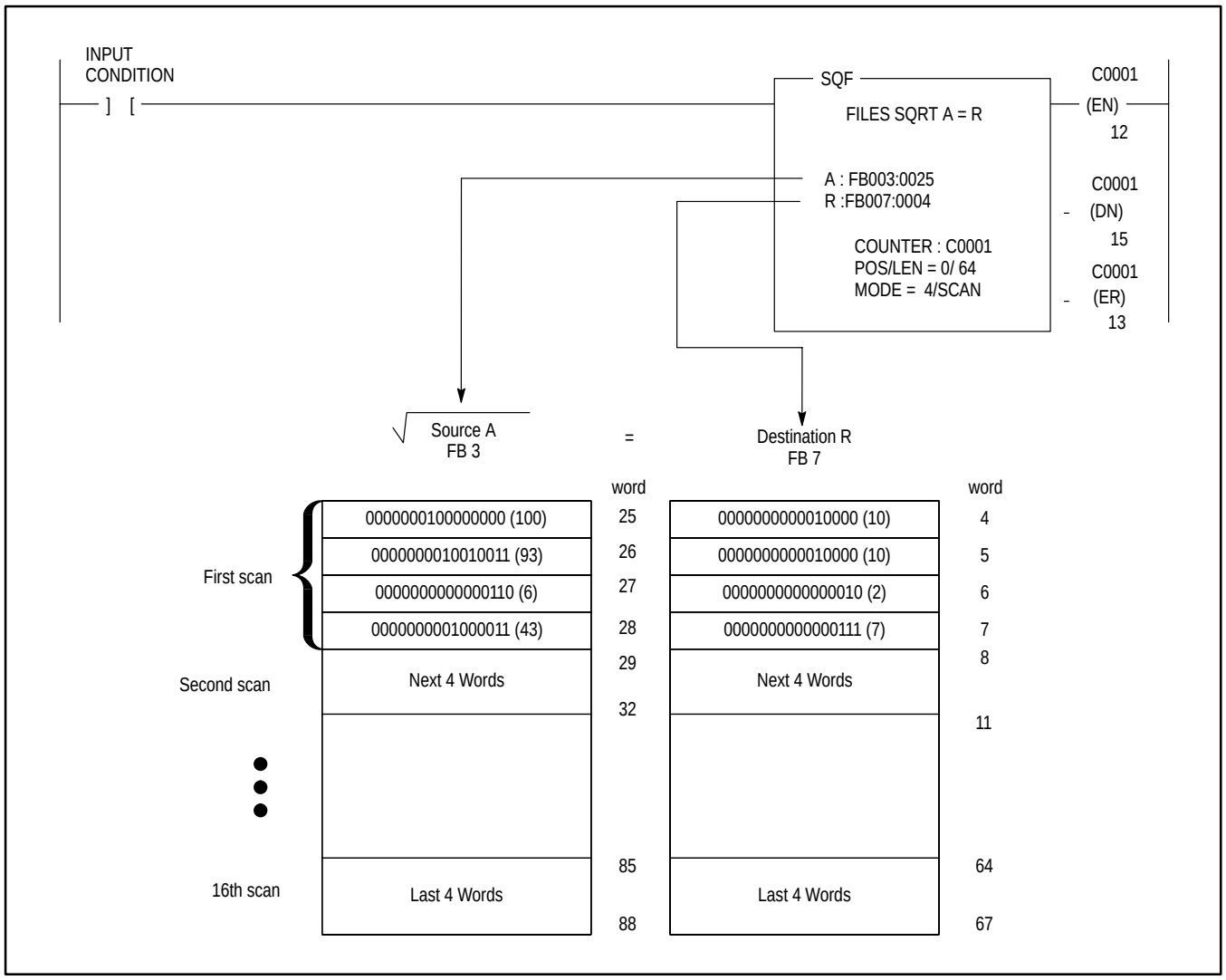
If source (A) contains 0 or a negative value, the processor declares an arithmetic fault and sets status bit 11 in status file 0, word 0 because you cannot take the square root of 0 or a negative value.

Example: Figure 8.19 shows a rung containing a file-square-root instruction.

If the rung goes from false to true, the processor takes the square root of the word values in the source (binary file 3 starting at word 25) and puts the results in the destination (binary file 7 starting at word 4). In this file-square-root instruction:

This parameter	Tells the processor
counter (C1)	what counter controls file-arithmetic operation
file position (POS=0)	to start at the first word (word 25) in the binary file
file length (LEN=64)	to take the square root of 64 words
mode (4/SCAN)	to execute the file operation on 4 words per program scan

Figure 8.19
Example Rung for a File-square-root Instruction



8.4.6 File Negate (NGF)

Required Parameters: Source (A) and destination (R) addresses, counter number, starting word numbers (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-negate instruction goes from false to true, the processor changes the sign of the values in the file words specified as the source and puts the results in the file words specified as the destination. If the rung is false, the processor does not execute the file-negate instruction, and the destination file retains its last values.

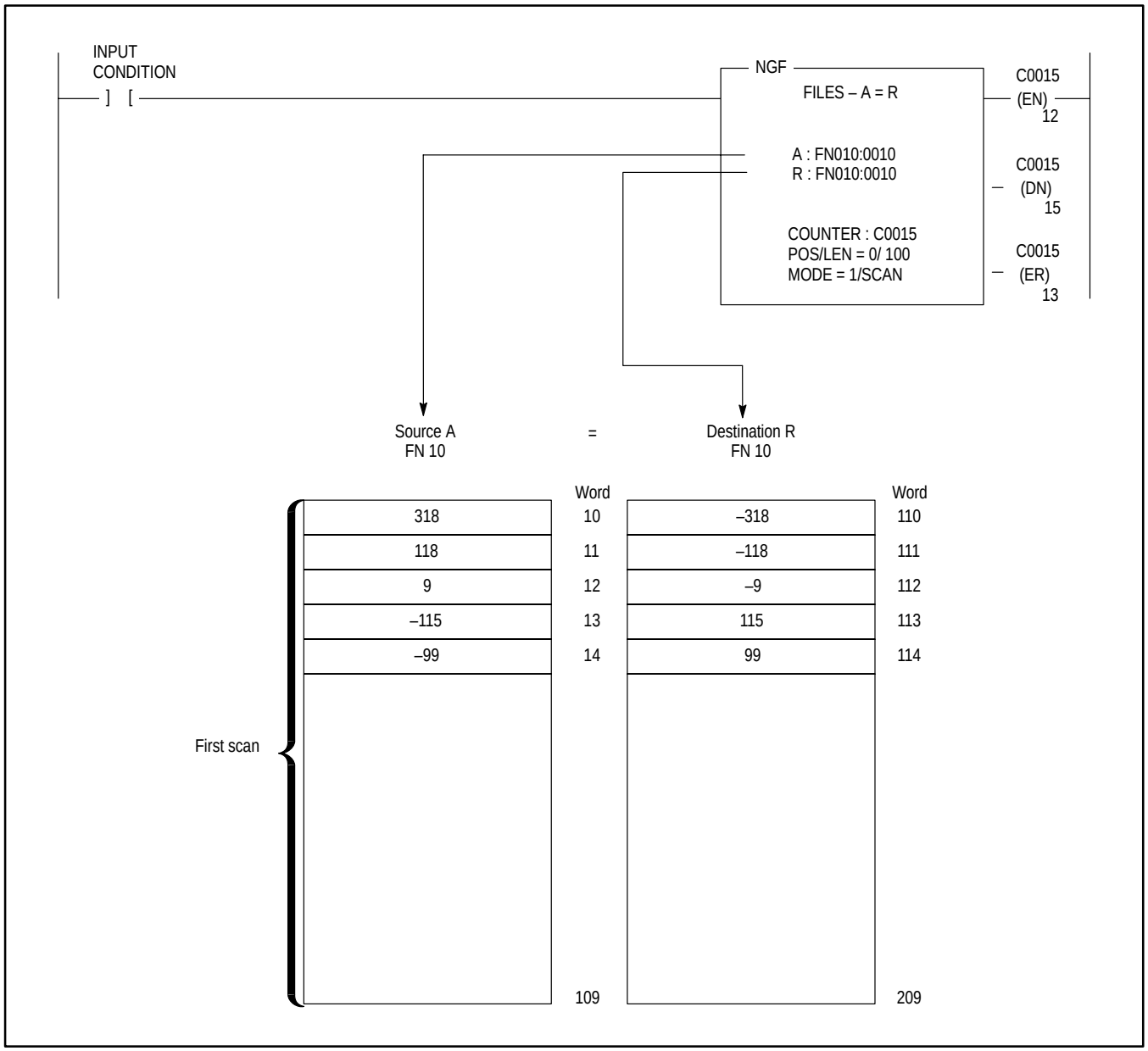
Make sure that the destination address is for a data table section that can store negative values. Otherwise the processor declares an arithmetic fault and sets bit 10 in status file 0, word 0.

Example: Figure 8.20 shows a rung containing a file-negate instruction.

If the rung goes from false to true, the processor changes the sign of the word values in the source (integer file 10 starting at word 10) and puts the results in the destination (integer file 10 starting at word 10). In this file-negate instruction:

This parameter	Tells the processor
counter (C15)	what counter controls file-negate operation
file position (POS=0)	to start at the first word (word 10) in the integer file
file length (LEN=100)	to negate 100 words
mode (ALL/SCAN)	to execute the entire file operation in one program scan

Figure 8.20
Example Rung for a File-negate Instruction



8.5 File-logic Instructions

File-logic instructions are output instructions that perform 16-bit word or one-word logic operations on files or portions of files in the data table. In file-logic operations, the processor looks at file addresses specified as sources and puts the result in the destination. It considers the data stored in the sources as binary and performs the operation bit by bit. For example:

```
    11001010
AND 10101010
    --------
    10001010
```

The addresses for the sources are usually from the binary section of the data table. However, you can specify other data table sections. If you do specify other data table sections, the processor does not convert to non-binary data types. It bases logic operations strictly on the bit pattern of each word in the files.

The values stored in the sources and destination must fall within the limits for the particular data table section used (Table 8.A).

Important: The floating-point and high-order-integer sections of the data table store data in 32-bit words. If you specify a 32-bit word as a source, the destination must also be a 32-bit word.

If you specify a 16-bit word data table section and a 32-bit data table section for a file-logic operation instruction, the processor executes a 32-bit-logic operation by adding 16 zeros to the upper byte of the 16-bit word. The original word is now stored in the lower 16 bits.

You can use the following logic-operation instructions:

- file AND
- file OR
- file XOR
- file NOT

8.5.1 File AND (ANF)

Required Parameters: Sources (A and B) and destination (R) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-AND instruction goes from false to true, the processor ANDs the values in the sources and puts the results in the destination. If the rung is false, the processor does not execute the file-AND instruction, and the destination file retains its last values. Table 8.B shows a truth table for a logical-AND operation.

Table 8.B
Truth Table for a Logical-AND Operation

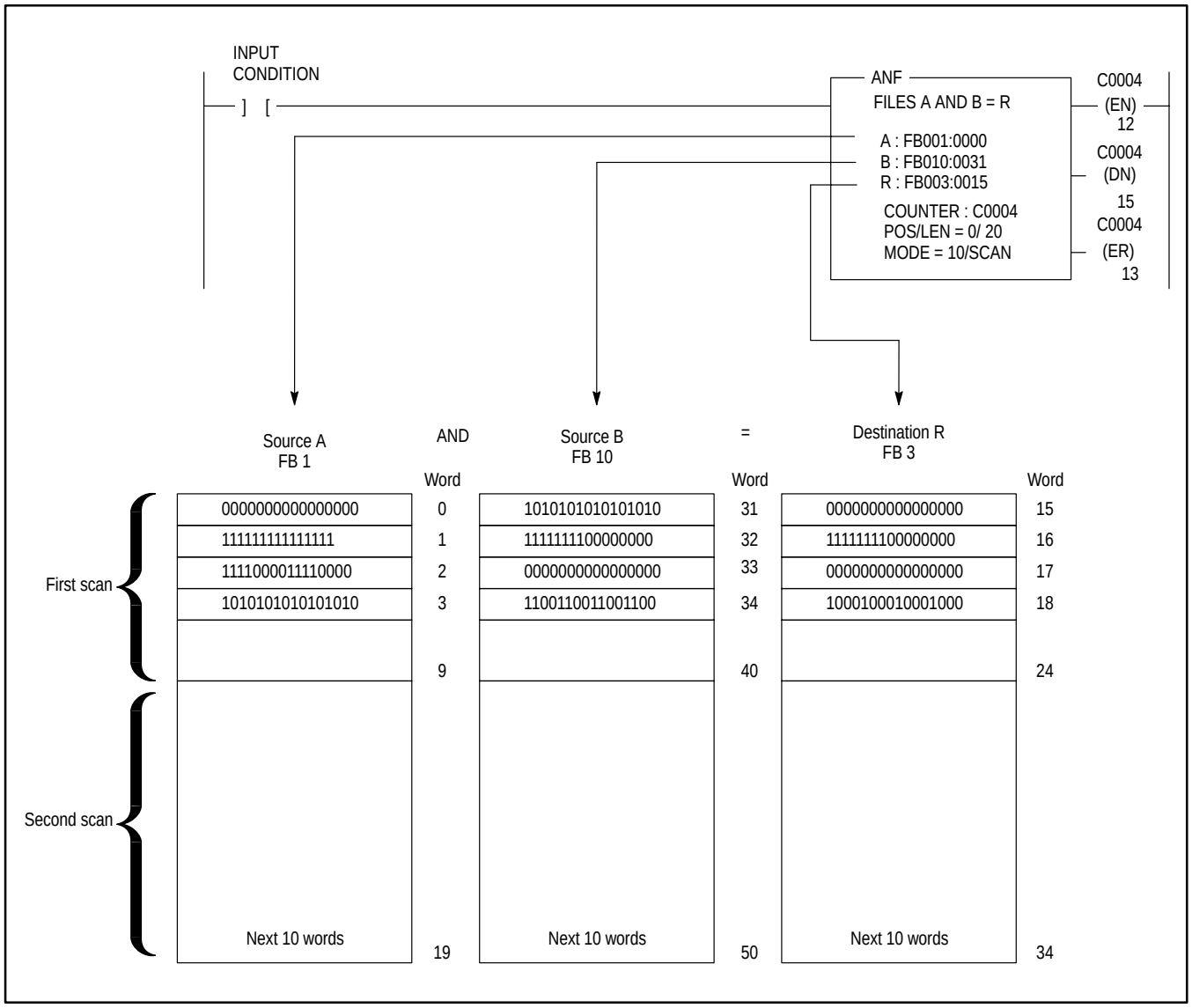
A	B	R
0	0	0
1	0	0
0	1	0
1	1	1

Example: Figure 8.21 shows a rung containing a file-AND instruction.

If the rung goes from false to true, the processor ANDs the file word values in the sources (binary file 1 starting at word 0 and binary file 10 starting at word 31) and puts the results in the destination (binary file 3 starting at word 15). In this file and instruction:

This parameter	Tells the processor
counter (C4)	what counter controls file-logic operation
file position (POS=0)	to start at the first word (word 0 in binary file 1 and word 31 in binary file 10)
file length (LEN=20)	to execute the file-AND instruction on 20 words
mode (10/SCAN)	to execute the file operation on 10 words per program scan

Figure 8.21
Example Rung for a File-AND Instruction



8.5.2 File OR (ORF)

Required Parameters: Sources (A and B) and destination (R) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-OR instruction goes from false to true, the processor ORs the values in the sources and puts the results in the destination. If the rung is false, the processor does not execute the file-OR instruction, and the destination file retains its last values. Table 8.C shows a truth table for a logical-OR operation.

Table 8.C
Truth Table for a Logical-OR Operation

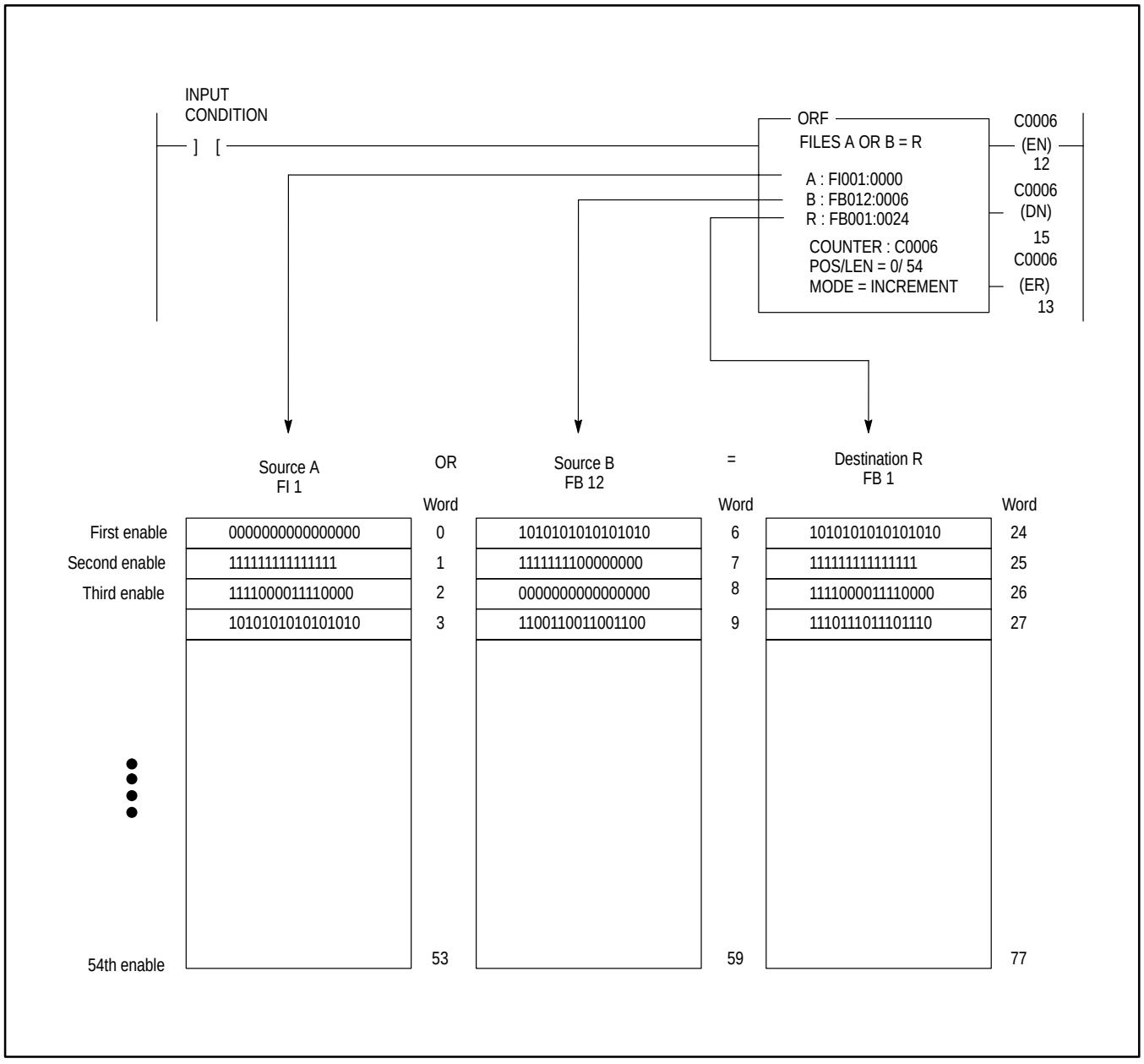
A	B	R
0	0	0
1	0	1
0	1	1
1	1	1

Example: Figure 8.22 shows a rung containing a file-OR instruction.

If the rung goes from false to true, the processor ORs the file word values in the sources (input image file 1 starting at word 0 and binary file 12 starting at word 6) and puts the result in the destination (binary file 1 starting at word 24) in this file-OR instruction:

This parameter	Tells the processor
counter (C6)	what counter controls file-logic operation
file position (POS=0)	to start at the first word (word 0 in input file 1 and word 6 in binary file 12)
file length (LEN=54)	to execute the file-OR instruction on 54 words
mode (INCREMENT)	to execute the file operation on one word per program scan each time that the rung goes from false to true

Figure 8.22
Example Rung for a File-AND Instruction



8.5.3 File XOR (XOF)

Required Parameters: Sources (A and B) and destination (R) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-XOR instruction goes from false to true, the processor XORs the values in the sources and puts the results in the destination. If the rung is false, the processor does not execute the file-XOR instruction, and the destination file retains its last values. Table 8.D shows a truth table for a logical-exclusive-or operation.

Table 8.D
Truth Table for a Logical-AND Operation

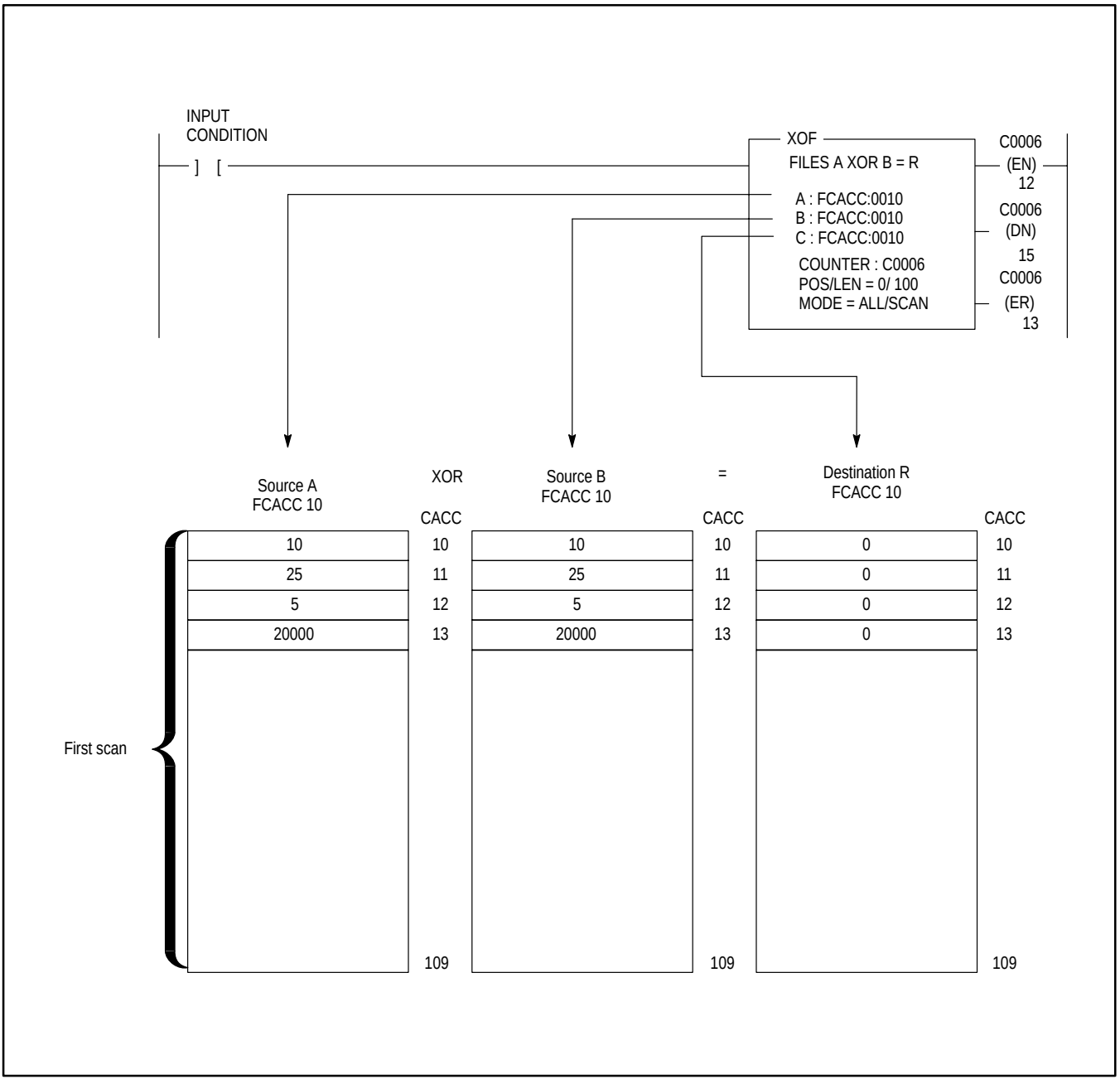
A	B	R
0	0	0
1	0	1
0	1	1
1	1	0

Example: Figure 8.23 shows a rung containing a file-XOR instruction.

If the rung goes from false to true, the processor XORs the counter-accumulated values defined by CACC:10. With the sources and destination addresses being the same, you can use this example to reset 100 counters. In this file-XOR instruction:

This parameter	Tells the processor
counter (C4)	what counter controls file-logic operation
file position (POS=0)	to start at the accumulated-value word for counter 10 in the counter section
file length (LEN=100)	to execute the file-XOR instruction on 100 words or 100 counters
mode (ALL/SCAN)	to execute the entire file operation in one program scan

Figure 8.23
Example Rung for a File-XOR Instruction



8.5.4 File NOT (NTF)

Required Parameters: Source (A) and destination (R) addresses, counter number, starting word number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-NOT instruction goes from false to true, the processor NOTs the values in the source and puts the results in the destination. If the rung is false, the processor does not execute the file-NOT instruction, and the destination file retains its last values. Table 8.E shows a truth table for a logical-NOT operation.

Table 8.E
Truth Table for a Logical-AND Operation

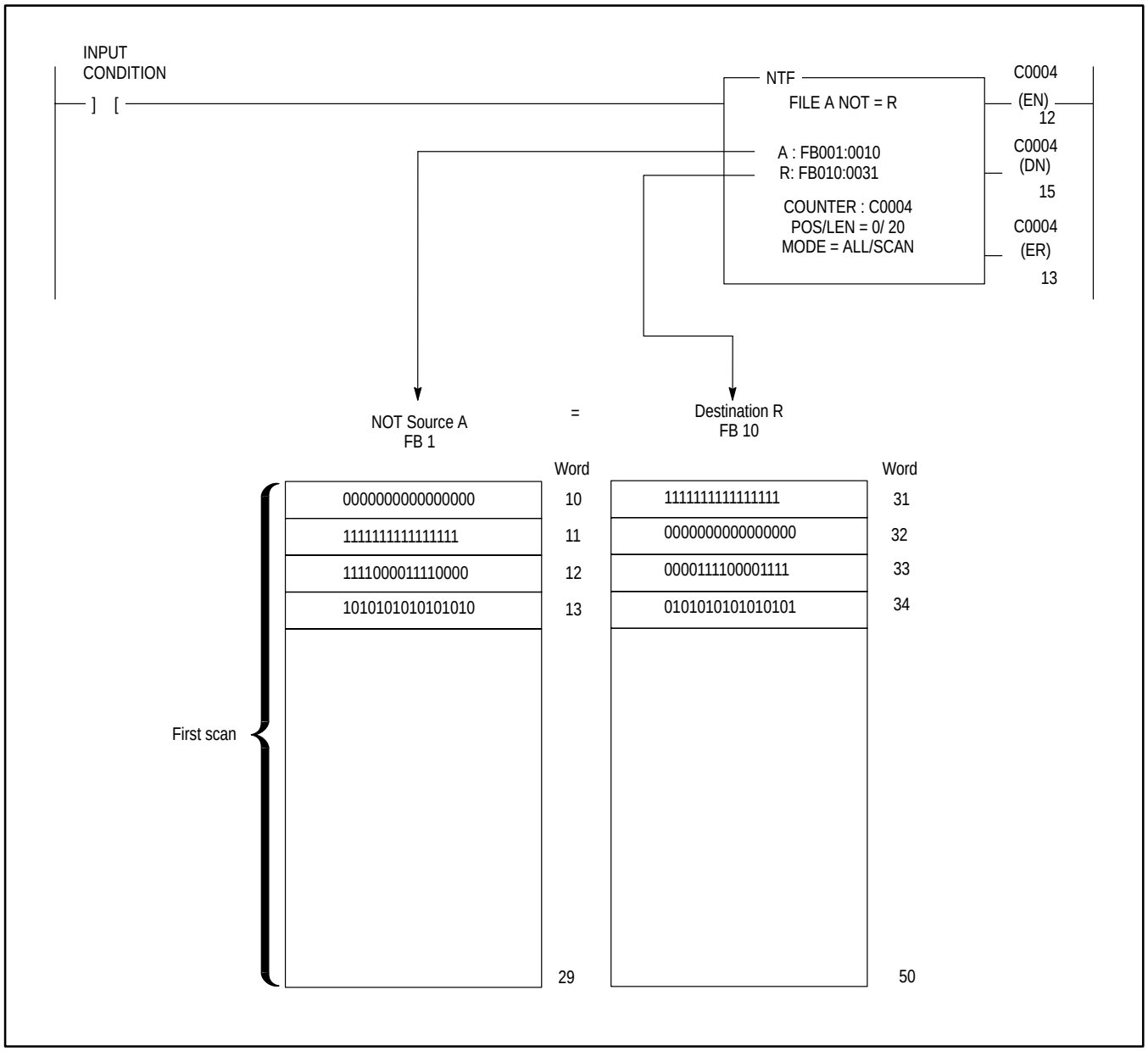
A	R
0	1
1	0

Example: Figure 8.24 shows a rung containing a file-NOT instruction.

If the rung goes from false to true, the processor NOTs the file word value in the source (binary file 1 starting at word 10) and puts the result in the destination (binary file 10 starting at word 31). In this file-Not instruction:

This parameter	Tells the processor
counter (C4)	what counter controls file-logic operation
file position (POS=0)	to start at the first word (word 10)
file length (LEN=20)	to execute the file-NOT instruction on 20 words
mode (ALL/SCAN)	to execute the file-NOT instruction in one program scan

Figure 8.24
Example Rung for a File-NOT Instruction



Chapter 9

Using Shift Registers

9.0 Chapter Objectives

In the preceding chapter, we described data manipulation instructions that you can use with files. In addition to these instructions, this chapter describes file instructions that you can use to program shift registers. After reading this chapter, you should understand:

- what shift registers are and how to apply them
- how to use bit shift and FIFO register instructions

9.1 Applying Shift Registers

A shift register is often used to simulate the movement or flow of parts and information on an assembly or transfer line.

If you use a shift register for	Data in the shift register could represent
simulating assembly or transfer lines	part types, quality, size, and/or status
inventory control	identification numbers or locations
system diagnostics	isolating a component that caused a shutdown

The processor supports two types of shift registers:

- Synchronous (bit shift registers) load bits into, shift data through, and unload bits from a file, one bit at a time.
- Asynchronous (first-in-first-out (FIFO) registers) operate similar to synchronous shift registers while providing a method of retrieving words in the order that they were stored.

We describe the instructions for using shift registers in the following sections.

9.2
Using Bit Shift Instructions

Bit shift instructions are output instructions that establish a synchronous shift register from a file (1 to 9999 bits in length). You can program the following bit shift instructions (Figures 9.1 and 9.2):

- bit shift left
- bit shift right

To program a bit shift instruction, you need to provide the processor with the following information:

- address containing the file of bits that you want to manipulate
- address of the bit that shifts into the file
- address of the bit that reflects the state of the last bit that shifts out of the file
- counter that the processor uses to locate data in the file

WARNING: Do not use a counter assigned to a bit shift instruction for any other purpose. Unexpected operation could result in damage to equipment and/or injury to personnel.

- number of bits that the processor shifts within the file

WARNING: If the number of bits in the file is not a multiple of 16, then the last word of the file contains bits that are not used in the bit-shift operation. Do not use these unused bits for storage. Unexpected operation could result in damage to equipment and/or injury to personnel.

Figure 9.1
Bit-shift-left Operation

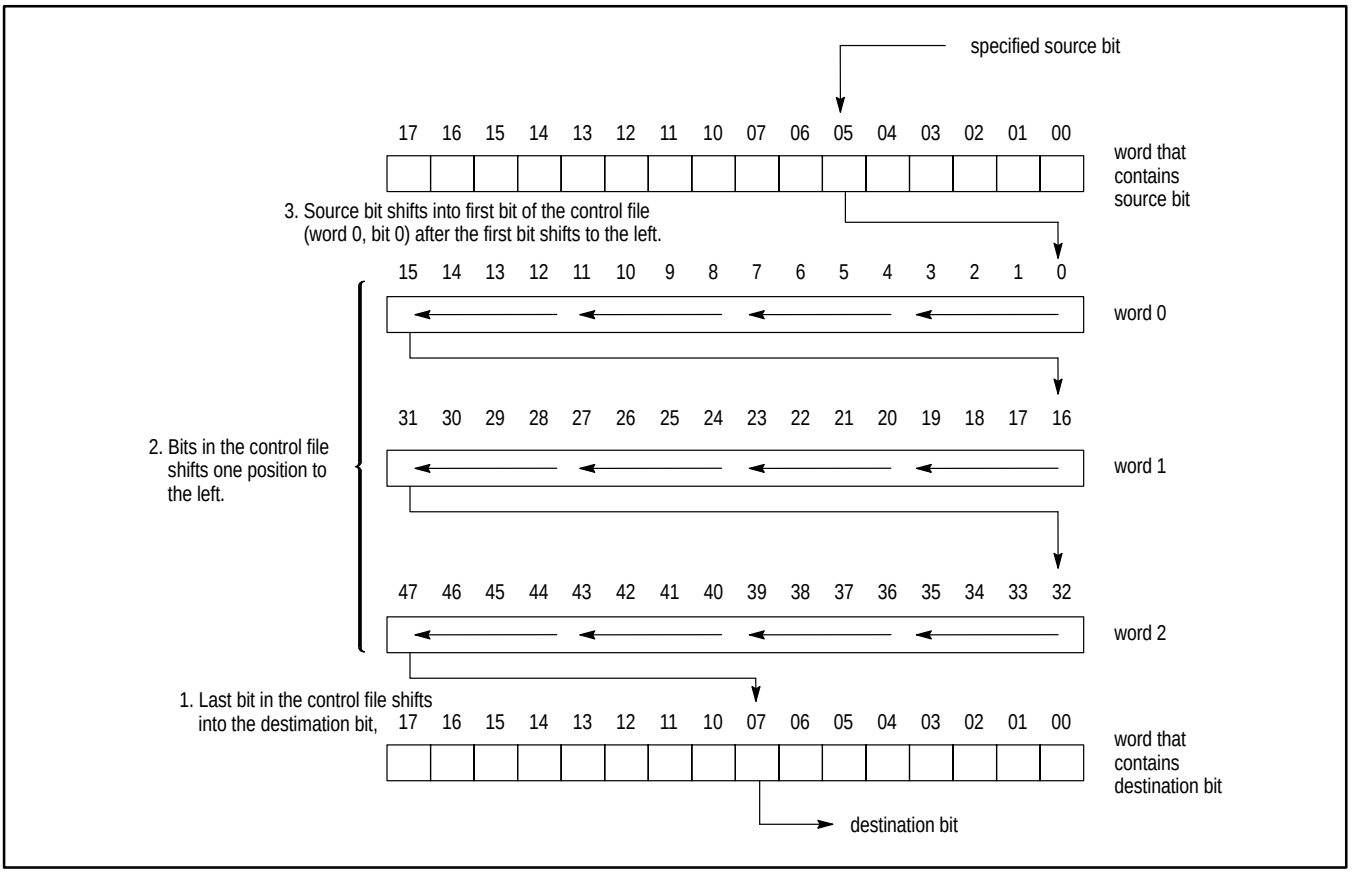
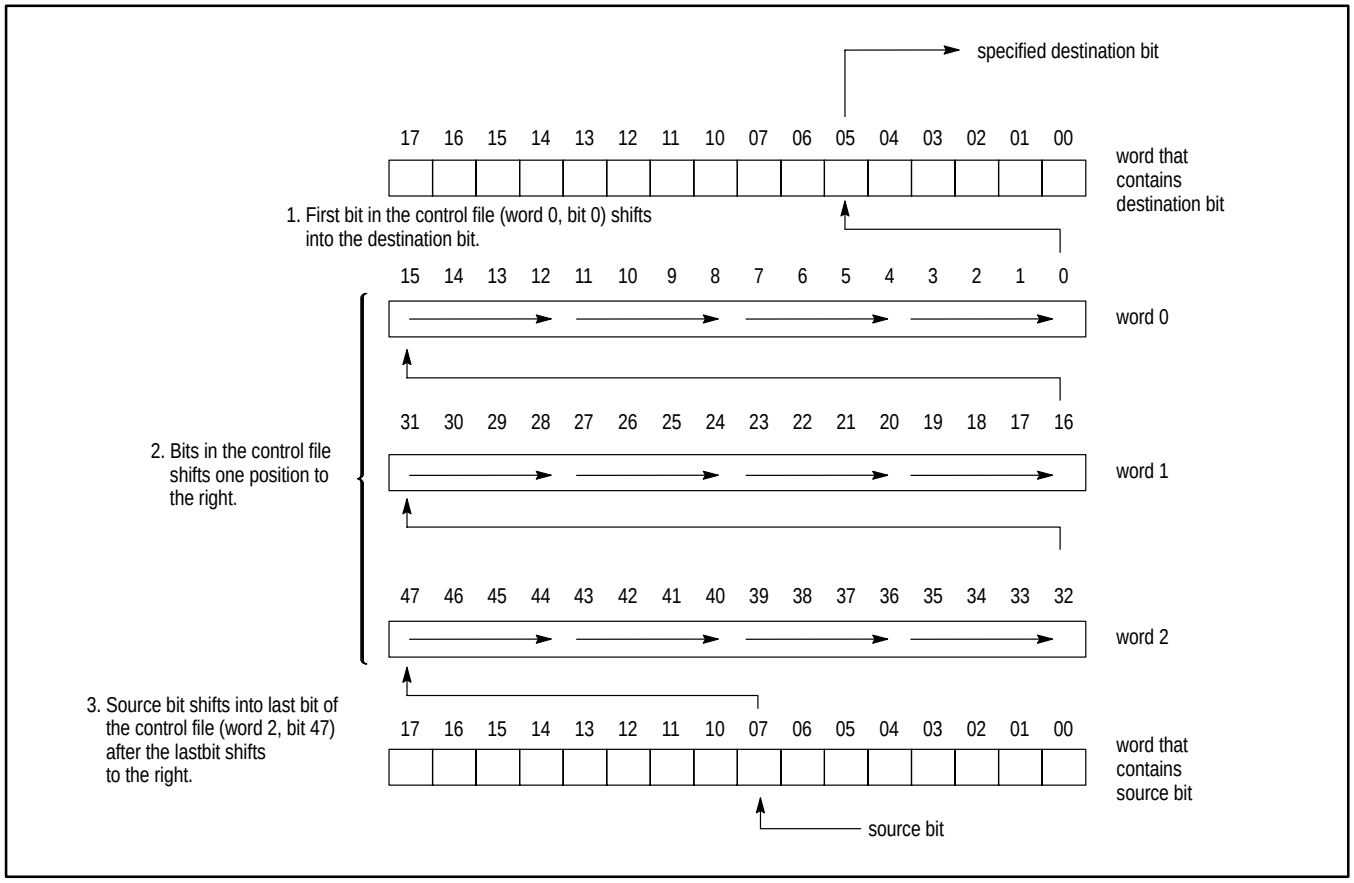


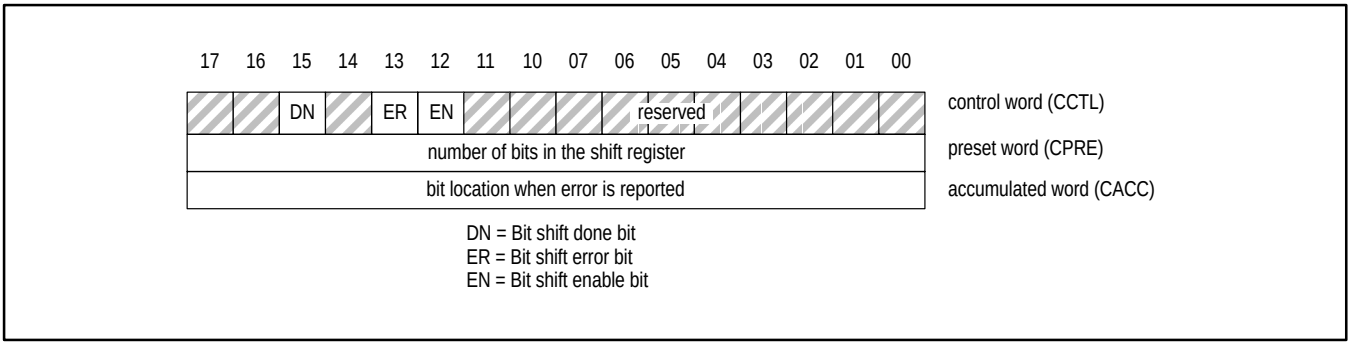
Figure 9.2
Bit-shift-right Operation



9.2.1 Counter Operation for Bit Shift Instructions

The counter for a bit shift instruction stores the following information (Figure 9.3):

Figure 9.3
Memory Storage for Bit Shift Instructions



Control word (CCTL) – stores the control bits that reflect the status of the bit shift instruction:

- Bit-shift Done – Bit 15 (DN) shows that a bit shift operation is complete.
- Bit-shift Error – Bit 13 (ER) shows that an error has occurred during the bit shift operation. You can find out the error by monitoring word 0 in status file 0 (refer to chapter 14). If an error occurs, the processor stops executing the bit shift instruction and stores the file bit number that caused the error in the counter accumulated value word. To restart the bit shift operation, you can reset the:
  - counter using the reset instruction to restart the entire bit shift operation
  - error bit to restart the bit shift operation from the point that the error occurred
- Bit-shift Enable – Bit 12 (EN) shows that a bit shift operation is in progress.

Preset value word (CPRE) – stores the number of bits that shift within the file.

Accumulated value word (CACC) – stores the bit position or the bit in the file that the processor is currently accessing.

9.2.2 Bit Shift Left (BSL)

Required Parameters: Control file (FILE), source bit (IN), destination bit (OUT) addresses, counter number, number of bits, mode of operation.

Description: When a rung containing a bit-shift-left instruction goes from false to true, the processor sets the enable bit and shifts the bits in the control file to the left (higher bit number) one bit position. The last bit shifts out of the control file into the specified destination bit. The specified source bit shifts into the first bit position. The processor executes the entire bit shift operation for one bit in one program scan.

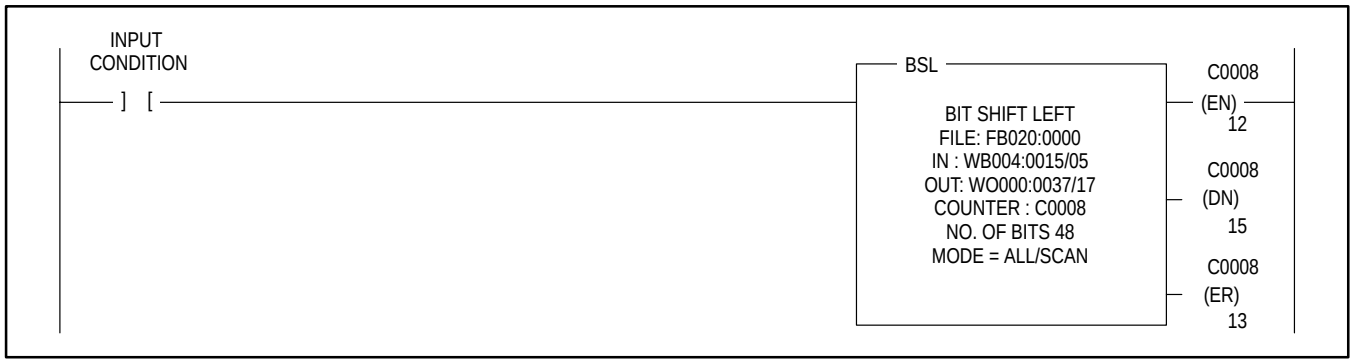
Example: Figure 9.4 shows a rung containing a bit-shift-left instruction.

If the rung goes from false to true, the processor shifts the bits in the control file (binary file 20) to the left one bit position. The source bit (binary bit WB4:15/05) shifts into the first bit position and the last bit shifts into the destination bit (output bit WO0:37/17).

In this bit-shift-left instruction:

This parameter	Tells the processor
file (FB020:0000)	the location of the control file
source (WB4:15/05)	to shift in bit 5 form binary file 4, word 15
destination (WO0:37/17)	to shift the last bit out to bit 17 in output file 0, word 37
counter (C8)	what counter controls the bit shift instruction operation
no. of bits (48)	to shift 48 bits in the control file

Figure 9.4
Example Rung for a Bit-shift-left Instruction



9.2.3 Bit Shift Right (BSR)

Required Parameters: Control file (FILE), source bit (IN), destination bit (OUT) addresses, counter number, number of bits, mode of operation.

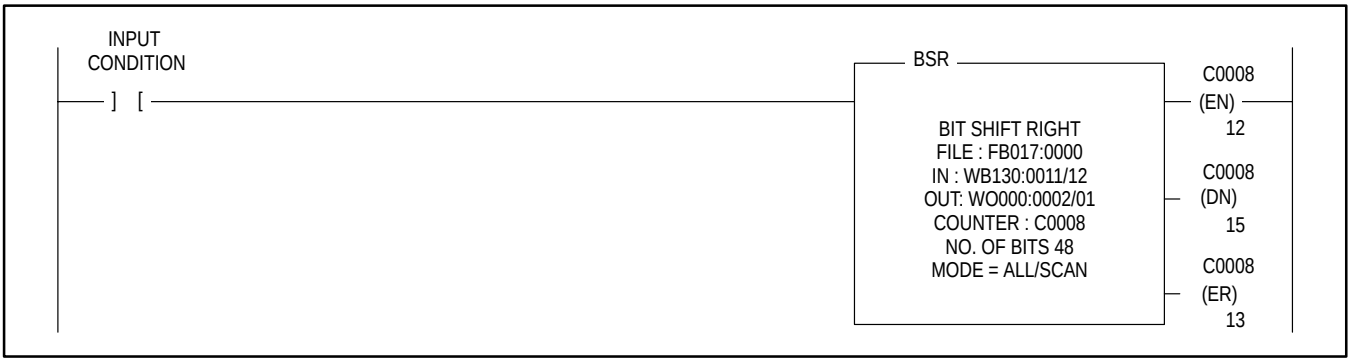
Description: When a rung containing a bit-shift-right instruction goes from false to true, the processor sets the enable bit and shifts the bits in the control file to the right (to a lower bit number) one bit position. The first bit shifts out of the control file into the specified destination bit. The specified source bit shifts into the last bit position. The processor executes the entire bit shift operation for one bit in one program scan.

Example: Figure 9.5 shows a rung containing a bit-shift-right instruction.

If the rung goes from false to true, the processor shifts the bits in the control file (binary file 17) to the right one bit position. The source bit (binary bit WB130:11/12) shifts into the last bit position and the first bit shifts into the destination bit (output bit WO0:2/01). in this bit-shift-right instruction:

This parameter	Tells the processor
file (FB017:0000)	the location of the control file
source (WB130:11/12)	to shift in bit 12 form binary file 130, word 11
destination (WO0:2/01)	to shift the first bit out to bit 01 in output file 0, word 2
counter (C8)	what counter controls the bit shift operation
no. of bits (48)	to shift 48 bits in the control file

Figure 9.5
Example Rung for a Bit-shift-right Instruction



9.3 Using FIFO Instructions

FIFO instructions are output instructions that establish an asynchronous shift register from a file (1 to 9999 words in length) (Figure 9.6). You can program the following fifo instructions:

- FIFO load
- FIFO unload

When programming a FIFO register, pair these instructions. Use the same file and counter address for both instructions.

To program a FIFO instruction, you need to provide the processor with the following information:

- address containing the file words that you want to manipulate
- counter that the processor uses to locate data in the file

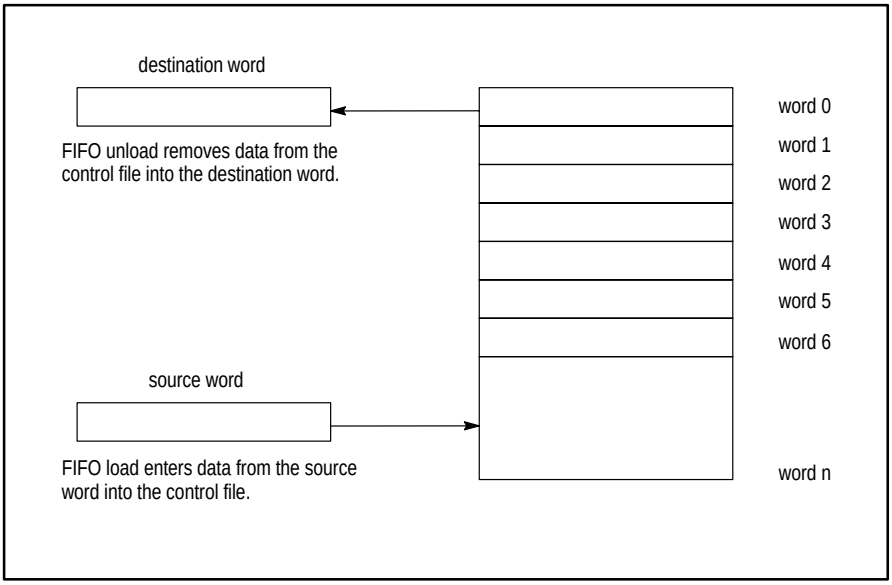
WARNING: Do not use a counter assigned to a FIFO instruction for any other purpose. Unexpected operation could result in damage to equipment and/or injury to personnel.

- number of words in the file
- address of the word that shifts into the file for a FIFO load operation
- address of the word that receives data shifted out of the file

The file used by the FIFO instructions stores the data used in the FIFO register. Since the order that the processor stores data in and removes data from this file is not apparent, do not use the data monitor to determine the register contents.

WARNING: Do not alter or use the contents of a file or counter assigned to a FIFO instruction for any other purpose. Unexpected operation could result in damage to equipment and/or injury to personnel.

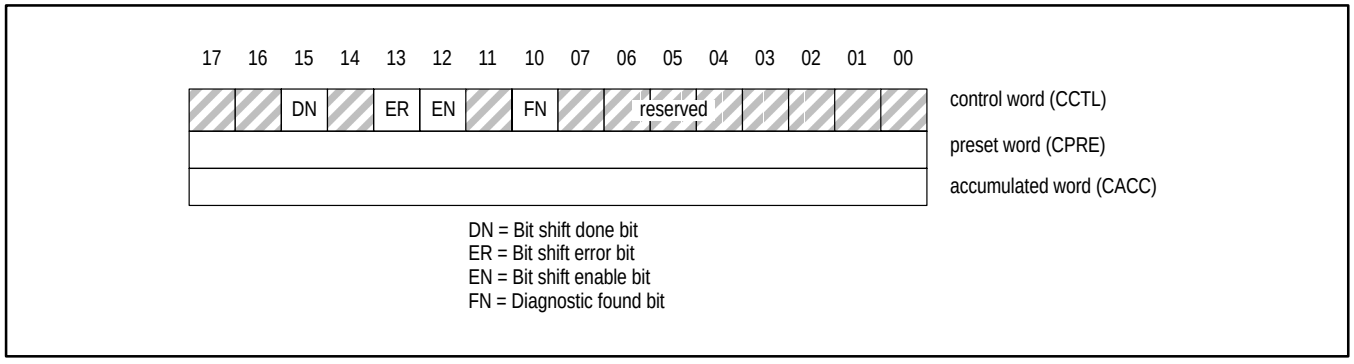
Figure 9.6
FIFO Operation



9.3.1 Counter Operation for FIFO Instructions

The counter for a FIFO instruction stores the following information (Figure 9.7):

Figure 9.7
Memory Storage for FIFO Instructions



Control word (CCTL) – stores the control bits that reflect the status of the FIFO instruction:

- FIFO Load – Bit 17 (LD) shows that a FIFO load operation is in progress.
- FIFO Unload – Bit 16 (UN) shows that a FIFO unload operation is in progress.

- FIFO Full – Bit 14 (FL) shows that the FIFO file is full.
- FIFO Empty – Bit 13 (EM) shows that the FIFO file is empty.

Preset value word (CPRE) – points to the file address where the next word will be stored.

Accumulated value word (CACC) – points to the file address where the next word will be taken from.

9.3.2 FIFO Load (FFL)

Required Parameters: Control file (FILE) address, counter number, number of words in file (FIFO SIZE), source address (INPUT).

Description: When a rung containing a FIFO-load instruction goes from false to true, the processor sets the load bit and loads the source word into the next available word in the control file as pointed to by the counter preset word. The processor loads a word each time the rung goes from false to true until it fills the control file. When the control file becomes full, the processor sets the full bit.

Your program should detect that the control file if full and inhibit the FIFO-load instruction from loading information stored at the source.

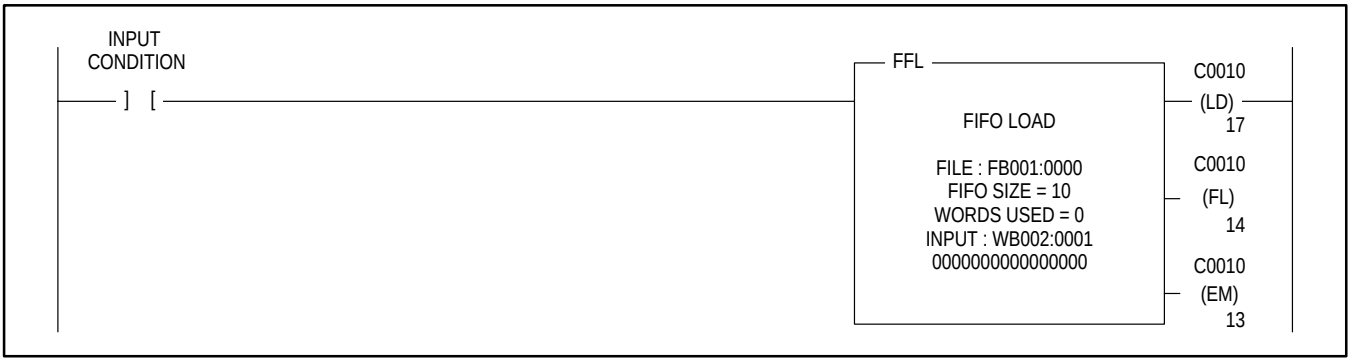
You may want to load the file in advance, or enable the FIFO-load instruction while inhibiting the FIFO-unload instruction until the control file contains the desired data.

Example: Figure 9.8 shows a rung containing a FIFO-load instruction.

If the rung goes from false to true, the processor loads the source word (binary word WB2:01) into the first available word in the control file (binary file one, word 0). In this FIFO-load instruction:

This parameter	Tells the processor
file (FB001:000)	the location of the control file
counter (C10)	what counter controls the FIFO operation
file size (FIFO SIZE=10)	the number of words in the control file
words used (0)	how many words of the register have been loaded minus the words that have been unloaded. This value changes as the FIFO-load instruction executes.
source (INPUT=WB2:01)	to load in word 1 from binary file 2

Figure 9.8
Example Rung for a FIFO-load Instruction



9.3.3 FIFO Unload (FFU)

Required parameters: Control file (FILE) address, counter number, number of words in file (FIFO SIZE), destination address (OUTPUT).

Description: When a rung containing a FIFO-unload instruction goes from false to true, the processor sets the unload bit and unloads data from the first word stored in the control file into the destination word. The processor unloads a word each time the rung goes from false to true until it empties the control file. When the control file becomes empty, the processor sets the empty bit. Thereafter, the processor transfers a zero value for each false-to-true rung transition until the FIFO-load instruction loads new values.

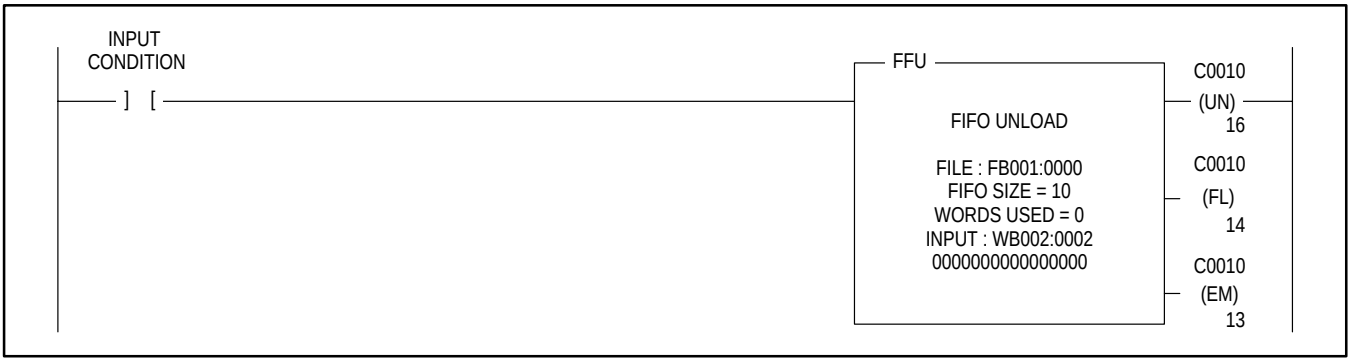
Your program should detect that the control file is empty and inhibit other instructions from using old information stored at the destination.

Example: Figure 9.9 shows a rung containing a FIFO-unload instruction.

If the rung goes from false to true, the processor unloads the words specified in the control file (binary file 1) into the destination word (binary word WB2:02). In this FIFO-unload instruction:

This parameter	Tells the processor
file (FB001:000)	the location of the control file
counter (C10)	what counter controls the FIFO operation
file size (FIFO SIZE=10)	the number of words in the control file
words used (0)	how many words of the register have been loaded minus the words the have been unloaded. This value changes as the FIFO-unload instruction executes.
source (OUTPUT=WB2:02)	to load in word 2 from binary file 2

Figure 9.9
Example Rung for a FIFO-unload Instruction



9.4
Example Program

Figure 9.10 shows a series of rungs that monitor the ladder program for data highway execution errors. The processor stores the corresponding error code for each error that occurs in a FIFO file. Then you can access the error codes in the order that they occurred. For detailed information on executing message procedures on the data highway, refer to the PLC-3 Communication Adapter Module (cat. no. 1775-KA) User's Manual (Publication 1775-6.5.1).

Figure 9.10
Example Program Using FIFO Instructions

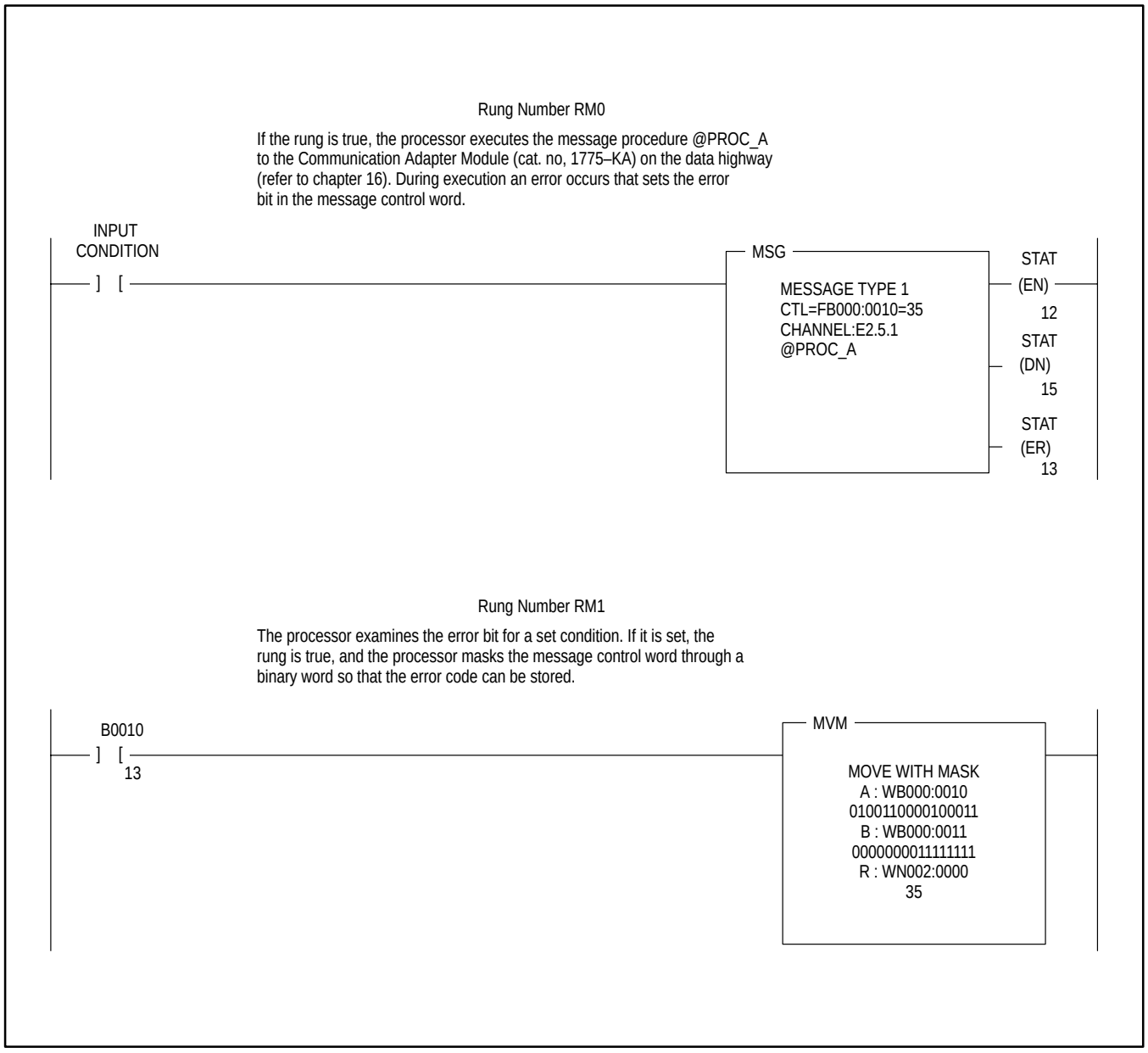
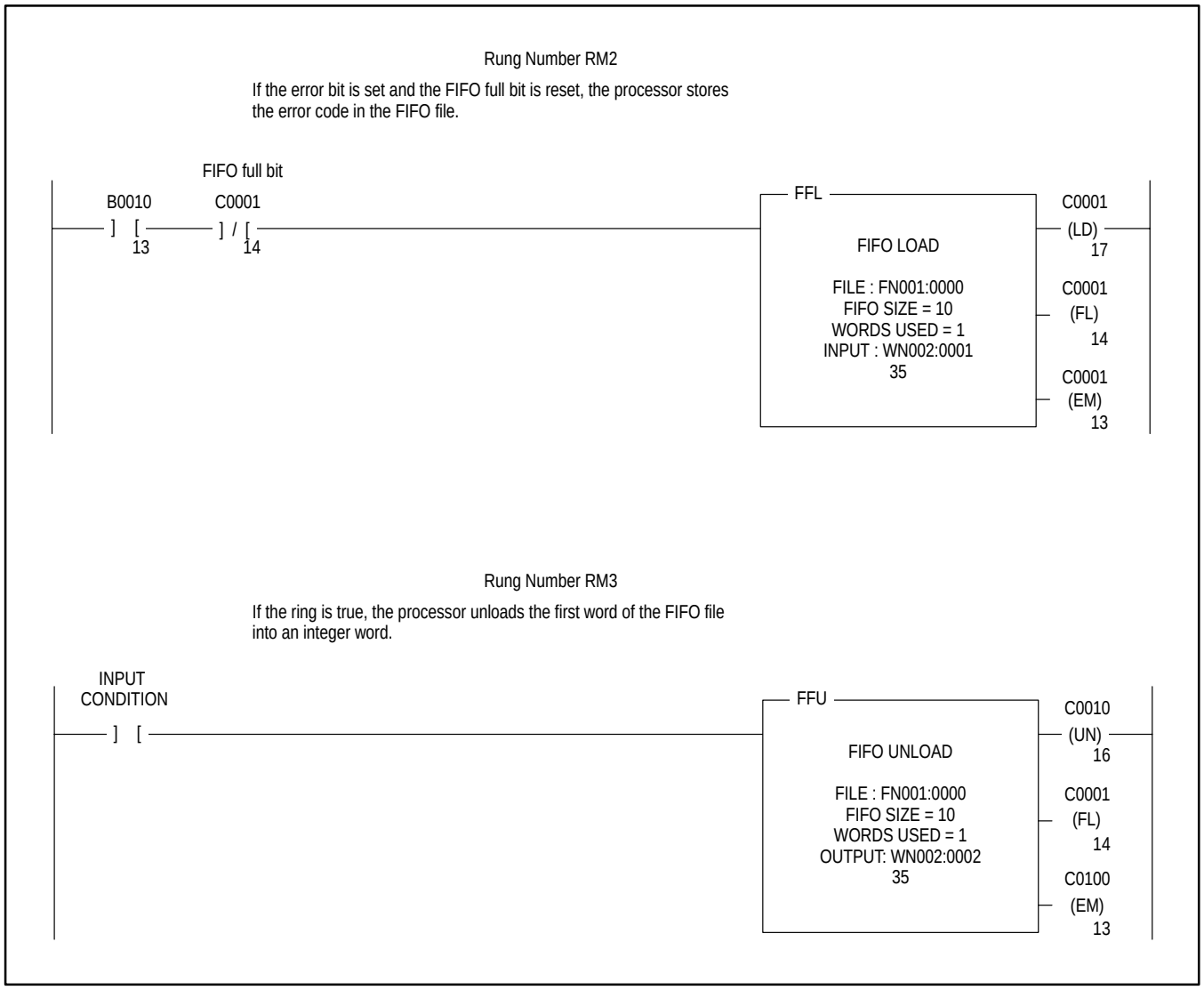


Figure 9.10
Example Program Using FIFO Instructions (continued)



Chapter 10

Indexing Bits within Files

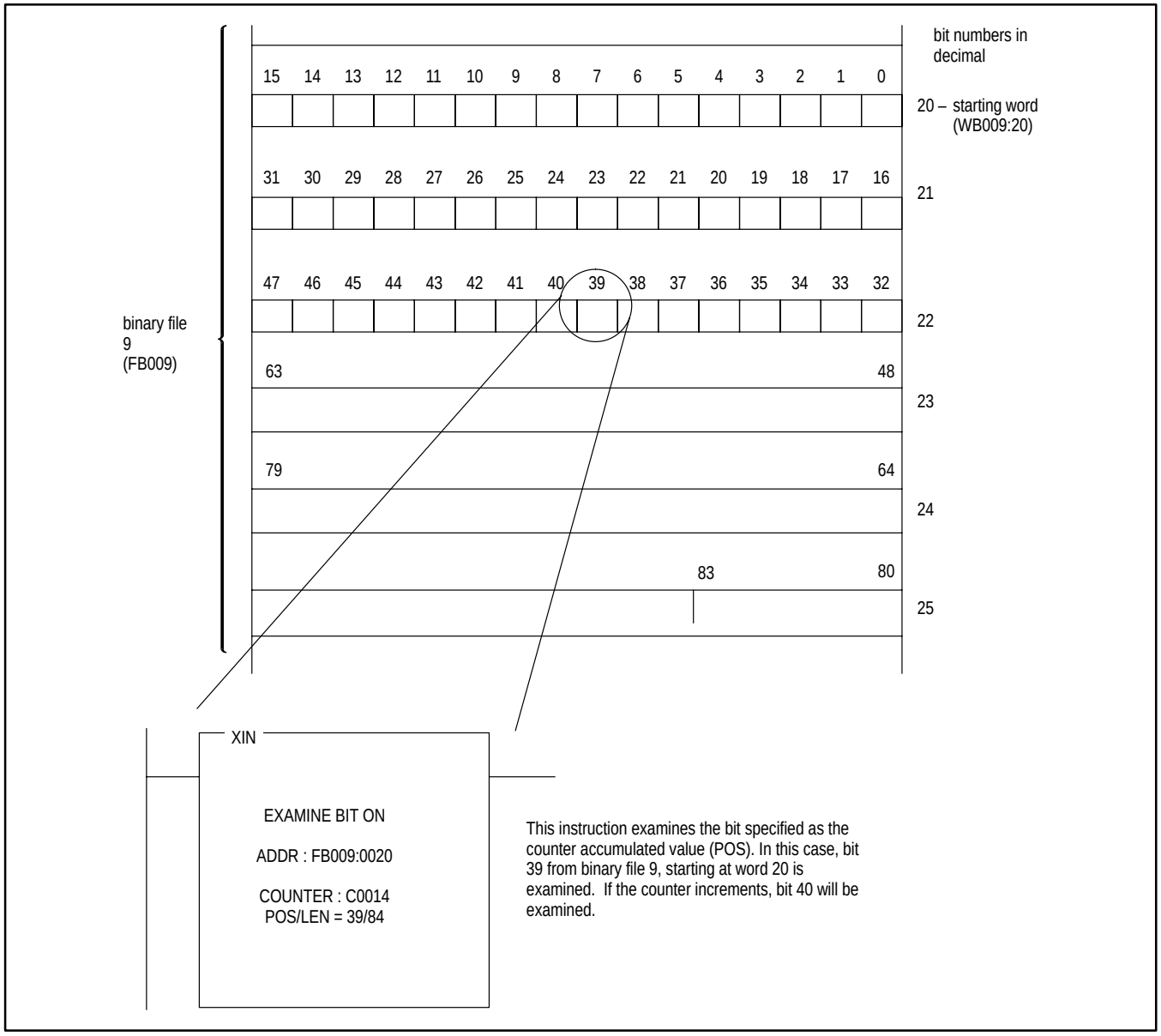
10.0 Chapter Objectives

After reading this chapter, you should understand how to use indexed-logic instructions with decimal bit addressing to index bits within files.

10.1 Using Indexed-logic Instructions

In chapter 7, we described how you can use decimal bit addressing to address bits within files. By addressing files using this method and using indexed-logic instructions, other instructions in the ladder program can tell the processor to (Figure 10.1):

Figure 10.1
Indexed-logic Instruction Operation



- examine a bit within a file for a set or reset condition
- set a bit within a file
- latch a bit within a file to the set or reset condition

Indexed-logic instructions are similar to relay logic instructions. The difference is that indexed-logic instructions operate on files and use a counter that increments or decrements based on other instructions that you have programmed. The counter accumulated value tells the processor what bit to examine, set, or reset.

To program indexed-logic instructions, you can use the following instructions:

- examine indexed bit on
- examine indexed bit off
- indexed bit on
- indexed bit set
- indexed bit reset

For each of these instructions, you need to provide the processor with the following information:

- address containing the file bits that you want to manipulate (specify file and starting word)
- counter that the processor uses to locate data in the file. The indexed-logic instruction does not increment or decrement the associated counter. This is done by other instructions in the ladder program.

WARNING: Do not use a counter assigned to a indexed-logic instruction for any other purpose. Unexpected operation could result in damage to equipment and/or injury to personnel.

However, you can assign more than one indexed-logic instruction to the same counter.

- number of bits within the file. Remember that bit numbers are one less than the bit position. The first bit within a file is number 0. If you set the number of bits within the file to 40, then the last bit number is 39.

CAUTION: If you attempt to access a bit number that is not within the file, the processor declares a bad address major fault and shuts down.

- current bit position within the file

10.1.1 Examine Indexed Bit On (XIN)

Required Parameters: Source file address (ADDR), counter number, starting bit number (POS), length of file in bits (LEN).

Description: The examine-indexed-bit-on instruction is an input instruction that tells the processor to monitor the file bit specified by the counter accumulated value:

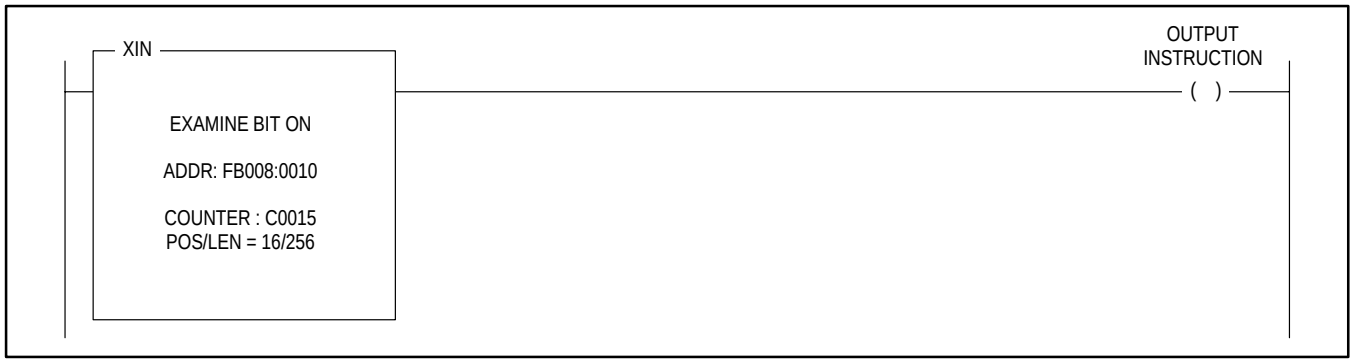
If the bit is	Then the instruction is
set	true
reset	false

Example: Figure 10.2 shows a rung containing an examine-indexed-bit-on instruction.

If the bit corresponding to the counter accumulated value (POS) is set, the processor executes the output instruction. In this examine-indexed-bit-on instruction:

This parameter	Tells the processor
address (FB008:0010)	the location of the source file
counter (C15)	what counter controls the indexed-logic operation
file position (POS=16)	to monitor bit 16 within the source file
file length (LEN=256)	the length of the source file. The last bit number in the file is one less than the length value.

Figure 10.2
Example Rung for an Examine-indexed-bit-on Instruction



10.1.2 Examine Indexed Bit Off (XIF)

Required Parameters: Source file address (ADDR), counter number, starting bit number (POS), length of file in bits (LEN).

Description: The examine-indexed-bit-off instruction is an input instruction that tells the processor to monitor the file bit specified by the counter accumulated value:

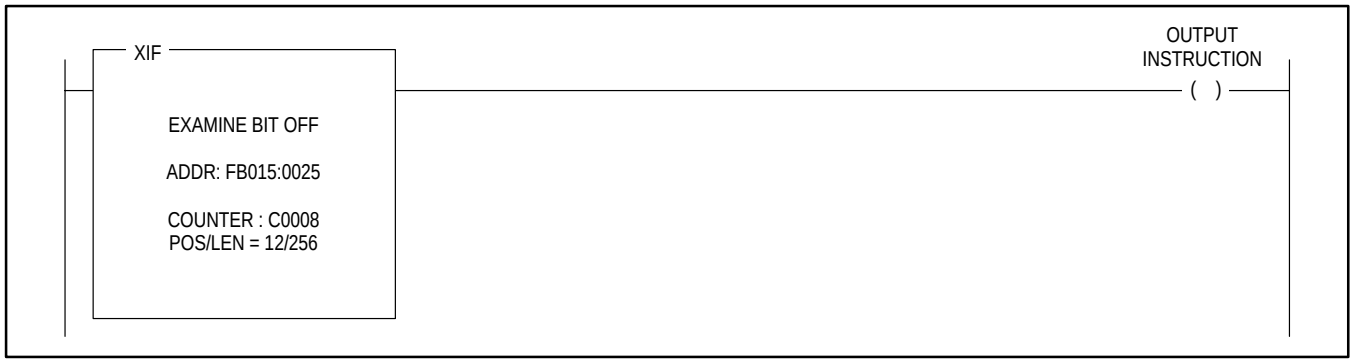
If the bit is	Then the instruction is
reset	true
set	false

Example: Figure 10.3 shows a rung containing an examine-indexed-bit-off instruction.

If the bit corresponding to the counter accumulated value (POS) is reset, the processor executes the output instruction. In this examine-indexed-bit-off instruction:

This parameter	Tells the processor
address (FB015:0025)	the location of the source file
counter (C8)	what counter controls the indexed-logic operation
file position (POS=12)	to monitor bit 12 within the source file
file length (LEN=256)	the length of the source file. The last bit number in the file is one less than the length value.

Figure 10.3
Example Rung for an Examine-indexed-bit-off Instruction



10.1.3 Indexed Bit On (BIN)

Required Parameters: Source file address (ADDR), counter number, starting bit number (POS), length of file in bits (LEN).

Description: The indexed-bit-on instruction is an output instruction that tells the processor to turn on the file bit specified in the counter accumulated value based on the rung conditions:

If the rung is	Then the processor turns the file bit
true	on
false	off

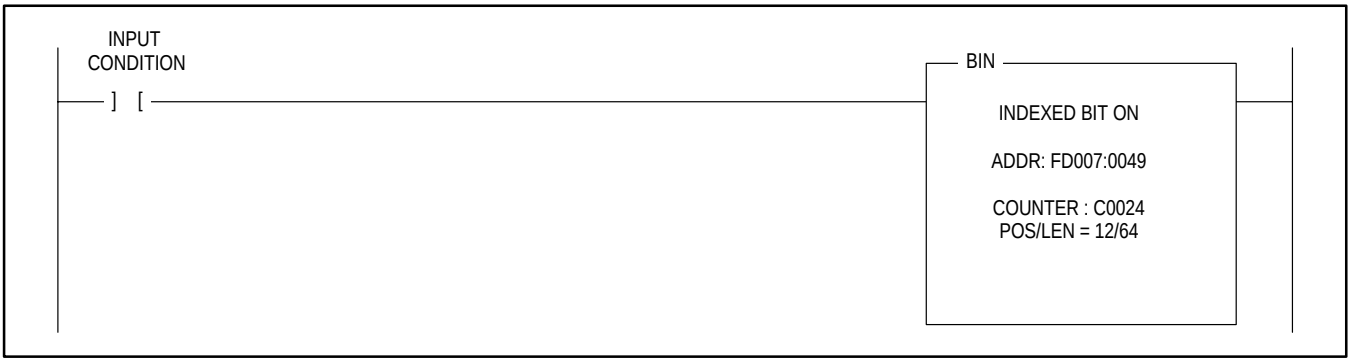
Example: Figure 10.4 shows a rung containing an indexed-bit-on instruction.

If the rung is true, the processor sets the bit corresponding to the counter accumulated value (POS).

In this indexed-bit-on instruction:

This parameter	Tells the processor
address (FD007:0049)	the location of the source file
counter (C24)	what counter controls the indexed-logic operation
file position (POS=12)	to monitor bit 12 within the source file
file length (LEN=64)	the length of the source file. The last bit number in the file is one less than the length value.

Figure 10.4
Example Rung for an Indexed-bit-on Instruction



10.1.4 Using Retentive Indexed-logic Instructions (BIS, BIR)

Required Parameters: Source file address (ADDR), counter number, starting bit number (POS), length of file in bits (LEN).

Description: The indexed-bit-set and indexed-bit-reset instructions are retentive output instructions meaning that they retain their last state in memory.

The indexed-bit-set instruction tells the processor to hold the bit specified by the counter accumulated value set regardless of the rung conditions:

If the rung is	Then the processor
true	sets and latches the file bit
false	leaves the bit in its previous state

If the processor sets the file bit, it remains set even after the rung conditions go false. To reset the bit, you can use the indexed-bit-reset instruction which performs the opposite operation:

If the rung is	Then the processor
true	resets the file bit
false	leaves the bit in its previuos state

Example: Figure 10.5 shows a rung containing an indexed-bit-set instruction.

If the rung is true, the processor latches the bit corresponding to the counter accumulated value (POS). In this indexed-bit-set instruction:

This parameter	Tells the processor
address (FN001:0050)	the location of the source file
counter (C10)	what counter controls the indexed-logic operation
file position (POS=63)	to latch bit 63, the last bit in the source file
file length (LEN=64)	the length of the source file. The last bit number in the file is one less than the length value.

Figure 10.5
Example Rung for an Indexed-bit-set Instruction

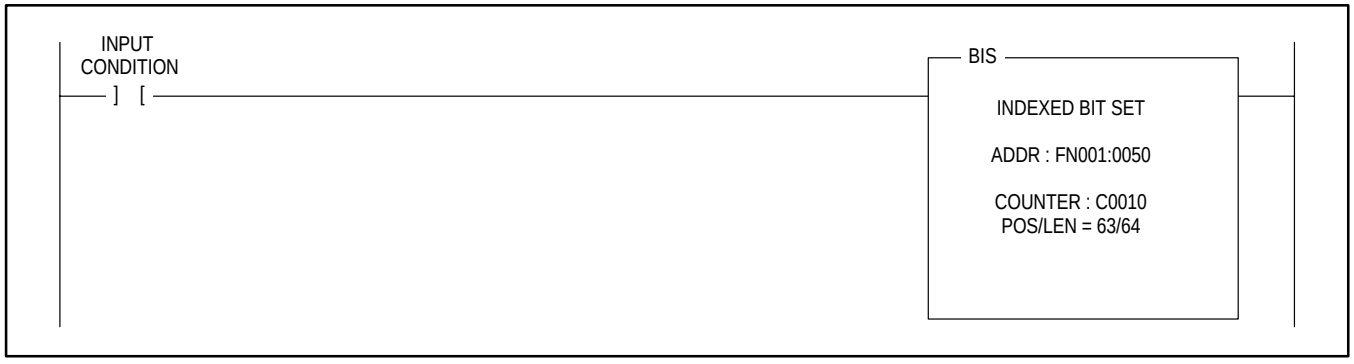
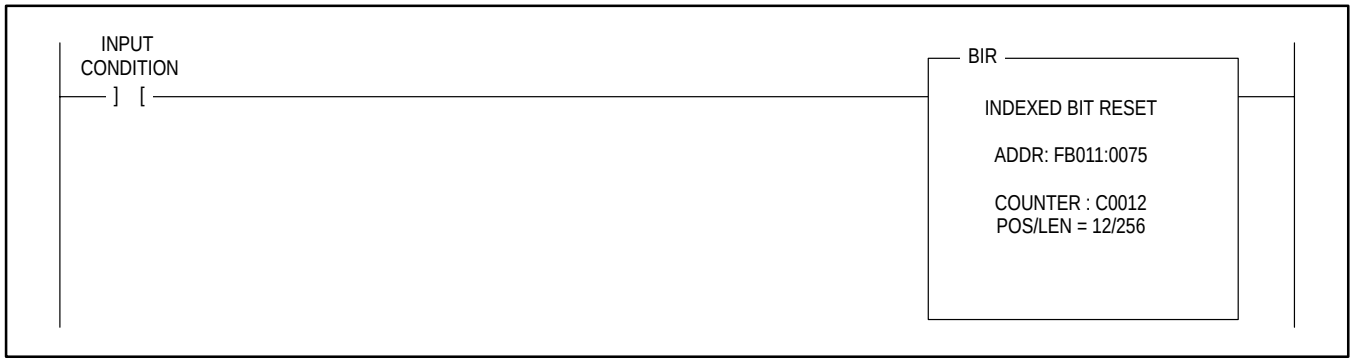


Figure 10.6 shows a rung containing an indexed-bit-reset instruction.

If the rung is true, the processor unlatches the bit corresponding to the counter accumulated value (POS). In this unlatch-indexed-bit instruction:

This parameter	Tells the processor
address (FB011:0075)	the location of the source file
counter (C12)	what counter controls the indexed-logic operation
file position (POS=12)	to unlatch bit 12 within the source file
file length (LEN=256)	the length of the source file. The last bit number in the file is one less than the length value.

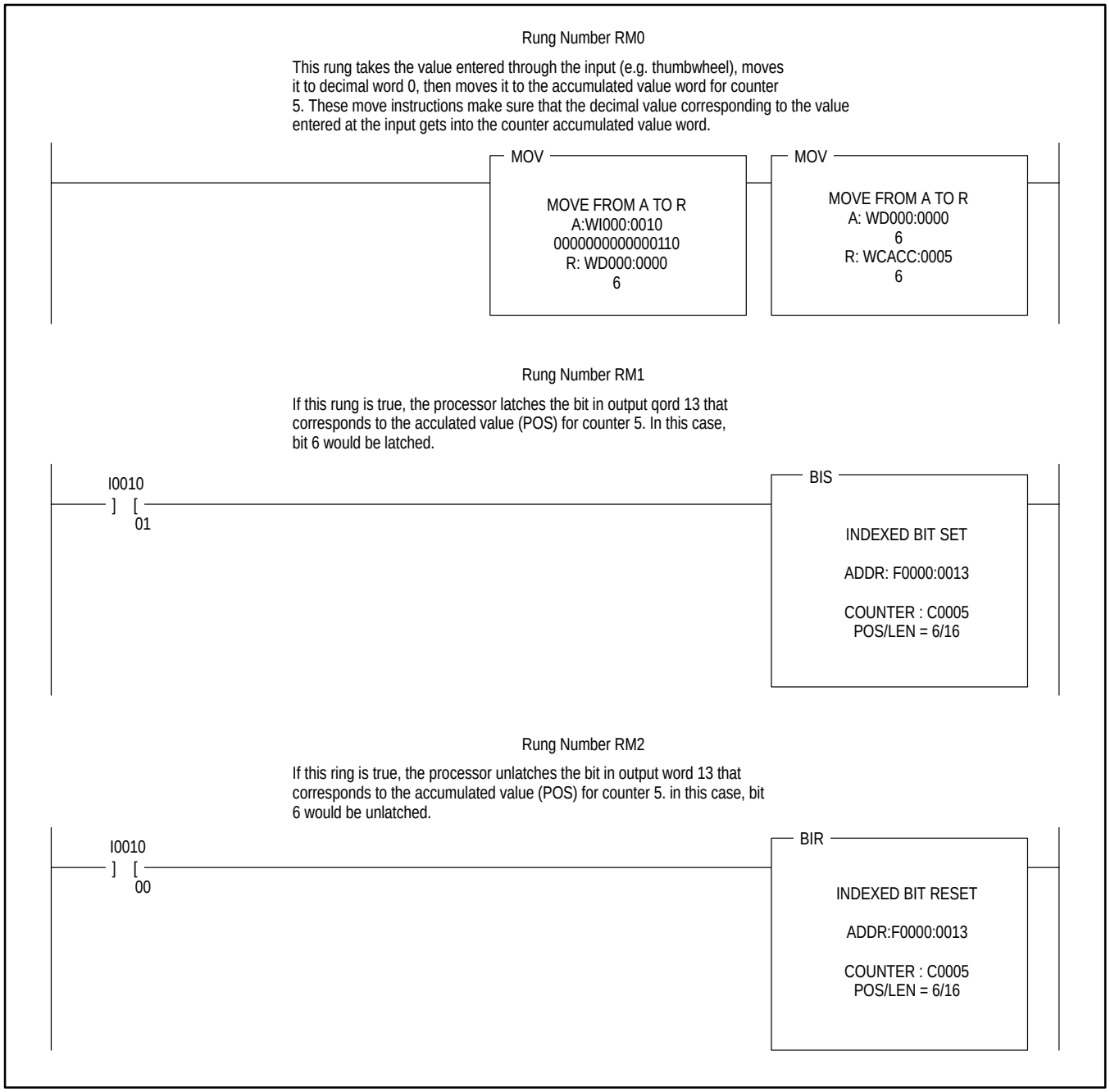
Figure 10.6
Example Rung for an Indexed-bit-reset Instruction



10.2 Example Program

Figure 10.7 shows a series of rungs that read in values from an input (e.g. thumbwheel switch) into the accumulated value word for a counter. Then by using indexed-logic instructions with this counter, you can latch a series of outputs on or off.

Figure 10.7
Example Program Using Indexed-logic Instructions



Chapter 11

Using Pointers for Indirect Addressing

11.0 Chapter Objectives

In this chapter, we describe the pointer mechanism that you can use to indirectly address data table locations. After reading this chapter, you should:

- understand what a pointer is and how to apply it
- understand how pointers operate
- know what the advantages of pointers are

11.1 Applying Pointers

As we discussed in chapter 3, the pointer section of the data table, unlike all other data table sections, stores an address instead of data. This address points to a word in the data table. By using special program structures called pointers, you can program rung instructions in the ladder program to indirectly access the address stored in the pointer section of the data table.

As a programming tool, you can use pointer to:

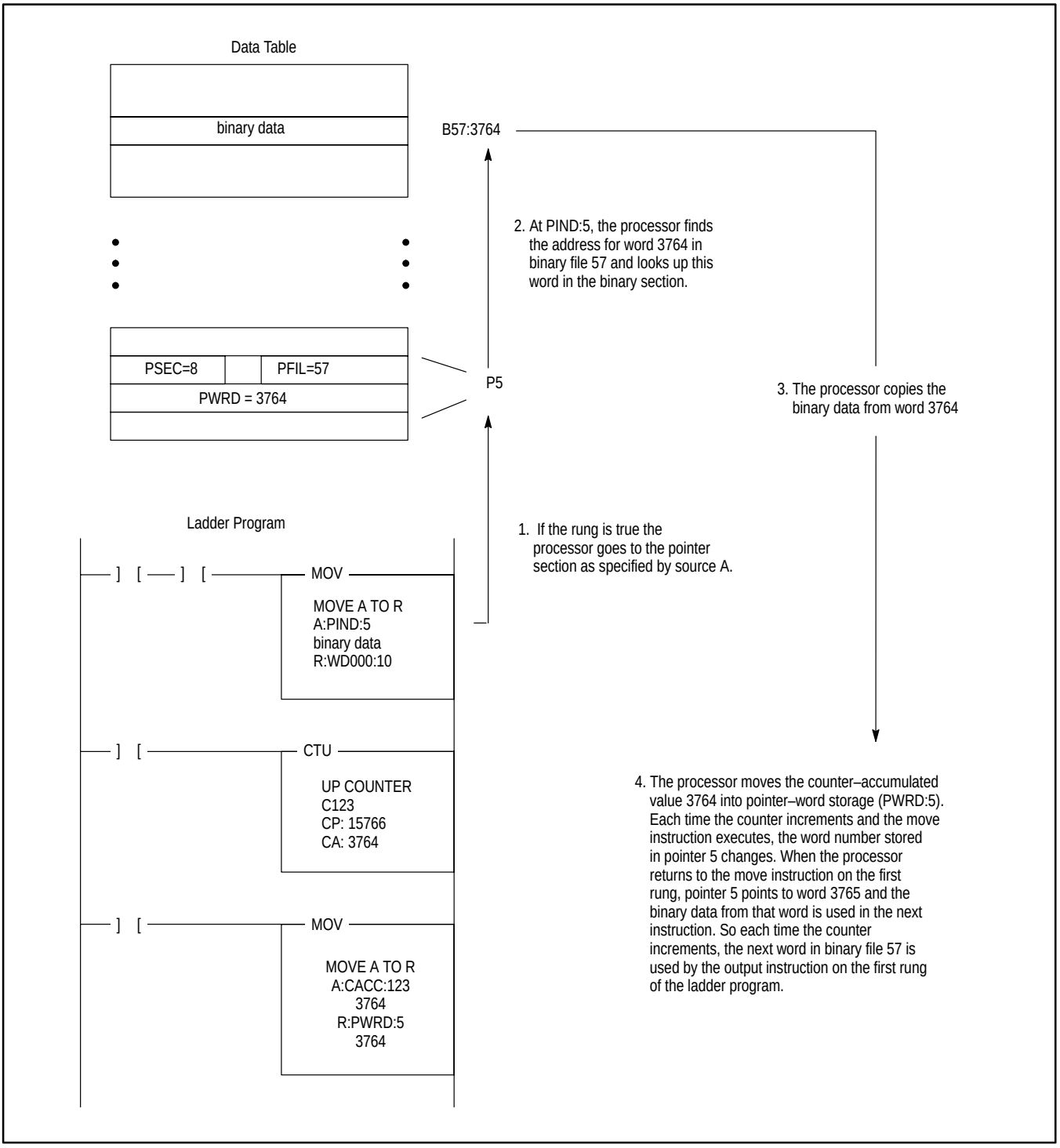
- reduce the program length for repetitive tasks
- allow one instruction to operate on large quantities of data

Do not confuse ladder-program pointers with system pointers. System pointers are used by the controller to define physical addresses for the first word of each implemented area of memory.

11.2 Pointer Operation

Figure 11.1 shows you a simple example on how pointers operate between the ladder program and the data table.

Figure 11.1
Example Program Showing Pointers Used in a Program Index Counter



To program pointers, you need to provide the processor with the following information:

Data table section number – tells the processor the location of the address in the data table:

Section	Number
Output image	1
Input image	2
Timer	3
Counter	4
Integer	5
Floating point	6
Decimal	7
Binary	8
ASCII	9
High-order integer	10
Status	13

Important: You cannot specify section numbers 0, 11, 14, or 15.

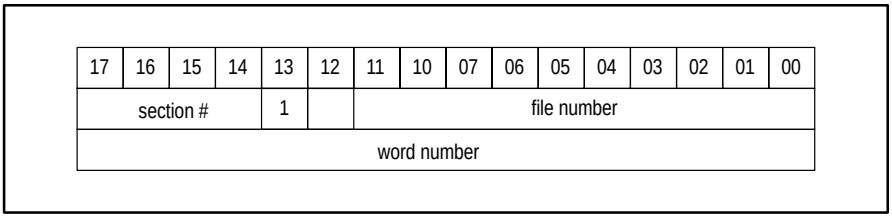
See the sections 11.2.3 and 11.2.4 for detailed information on using pointers to address timers, counters, and pointers.

File number – tells the processor the file location for the address in the data table.

Word number – tells the processor the word location for the address in the data table.

Within the pointer section of the data table, each pointer uses two words to store the parameters (Figure 11.2).

Figure 11.2
Memory Storage for Pointers



By manipulating the parameters within these words, you can read or change the location that the pointer addresses. To access the pointer information in an instruction, use the following abbreviations along with the pointer number:

- **PSEC** specifies the section number of the data table section that the pointer addresses.
- **PFIL** specifies the file number that the pointer addresses.
- **PWRD** specifies the word number that the pointer addresses.
- **PIND** accesses the data in the word specified by the pointer.

Important: You can change these values with instructions in the ladder program or through the programming device. By changing the PSEC, PFIL, or PWRD values, you are changing the **address** the pointer looks at; not the **data** itself.

In using pointers, you can address the data table to locate a:

- word inside of a file
- file starting at a certain word address

We give you simple examples of these addressing methods in the following sections.

11.2.1 Locating a Word Inside of a File

If pointer 5 stores the address binary word 3764 in file 57, then you can use the following designation in an instruction to access the data stores at this address:

PIND:5 or WPIND:5

You could then use the following designations to make changes on one or more of the three parts that make up this pointer:

- PSEC:5 accesses the section number which is 8 for the binary section.
- PFIL:5 accesses the file number which is 57.
- PWRD:5 accesses the word number which is 3764.

Figure 11.3 shows you example rungs that access these parameters.

Figure 11.3
Example Rungs for Using Pointers to Locate a Word Inside of a File

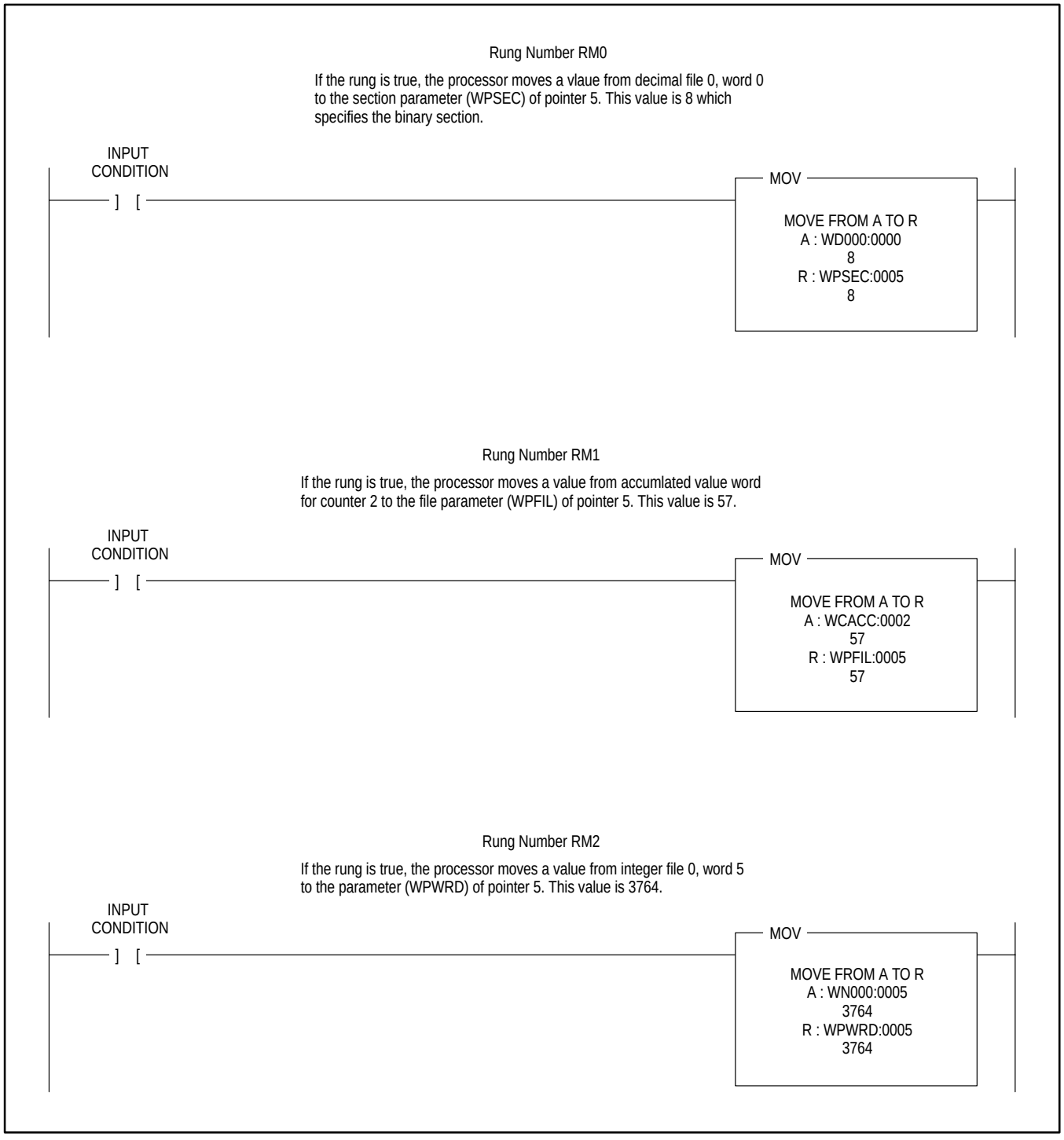
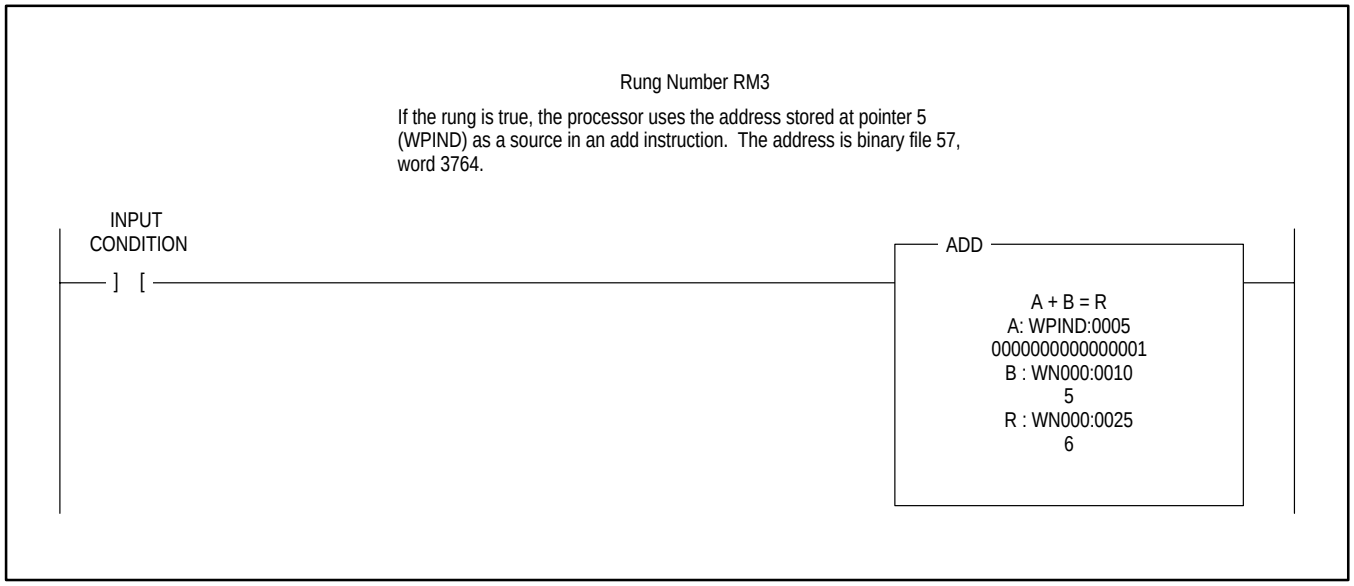


Figure 11.3
Example Rungs for Using Pointers to Locate a Word Inside of a File (continued)



11.2.2 Locating a File Starting at a Certain Word Address

If pointer 17 stores decimal file 3 starting at word 21, then you can use the following designation in an instruction to access the data stored at this address:

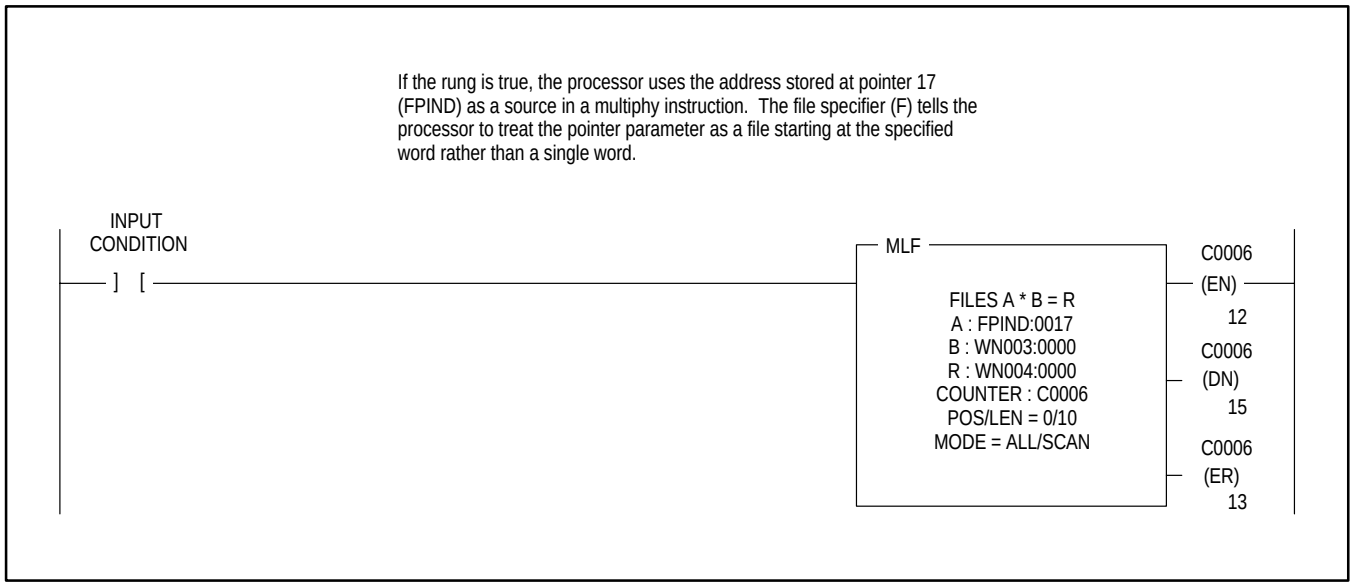
FPIND:17

The FPIND tells the processor that the values inside the pointer address a part of the file rather than one specific word. You could then use the following designations to make changes on one or more of the three parts that make up this pointer:

- PSEC:17 accesses the section number which is 7 for the decimal section.
- PFIL:17 accesses the file number which is 3.
- PWRD:17 accesses the starting word number which is 21.

Figure 11.4 shows you an example rung that accesses these parameters.

Figure 11.4
Example Rungs for Locating a File Starting at a Word Address



11.2.3 Pointer Operation for Timers and Counters

To program pointers to address a timer or counter, you provide the processor with the following information:

Data table section number – uses 3 for a timer and 4 for a counter.

Word number – tells the processor which word to access for the timer or counter (control, accumulated value, or preset value word).

Timer or counter number – tells the processor the number of the timer or counter.

Figure 11.5 shows an example data monitor for a pointer that addresses a timer or counter.

Figure 11.5
Example Data Monitor for Pointers that Address Timers or Counters

START = WPSEC:0001

WORD #	SECT	FILE	WORD	ADDRESS
00001	3 = T	512	1	TACC:0001
00002	4 = C	256	1	CPRE:0001

In programming the pointer, you use the PSEC, PFIL, and PWRD abbreviations:

- **PSEC** – specifies the data table section number:
  - 3 for timer
  - 4 for counter
- **PFIL** specifies the word number:
  - 0 for the control word
  - 256 for the preset-value word
  - 512 for the accumulated-value word
- **PWRD** specifies the timer or counter number.

Important: You cannot program the PIND abbreviation to indirectly address a timer or counter.

11.2.4 Nested Pointer Operation

You can also use a pointer to address another pointer. However, this programming technique can cause confusion in the ladder program. If you choose to nest pointers, we recommend that you document all pointers to that you understand how they function in the ladder program.

To program pointers to address another pointer, you provide the processor with the following information:

Data table section number – uses 12 for pointers.

Pointer parameter – tells the processor which part to access for the pointer (section (PSEC), file (PFIL), word (PWRD), or indirect (PIND)).

Pointer number – tells the processor the number of the pointer.

Figure 11.6 shows an example data monitor for a pointer that addresses another pointer.

Figure 11.6
Example Data Monitor for Pointers that Address Other Pointers

START = WPSEC:0001

WORD #	SECT	FILE	WORD	ADDRESS
00001	12 = P	768	1	WPIND:0001
00002	12 = P	512	1	WPWRD:0001
00003	12 = P	256	1	WPFIL:0001
00004	12 = P	0	1	WPSEC:0001

In programming the pointer you use the PSEC, PFIL, and PWRD abbreviations:

- **PSEC** specifies the data table section number (11 for pointers).
- **PFIL** specifies the pointer parameter:
  - 0 for the section number (PSEC)
  - 256 for the file number (PFIL)
  - 512 for the word number (PWRD)
  - 768 for the indirect (PIND)
- **PWRD** specifies the pointer number.

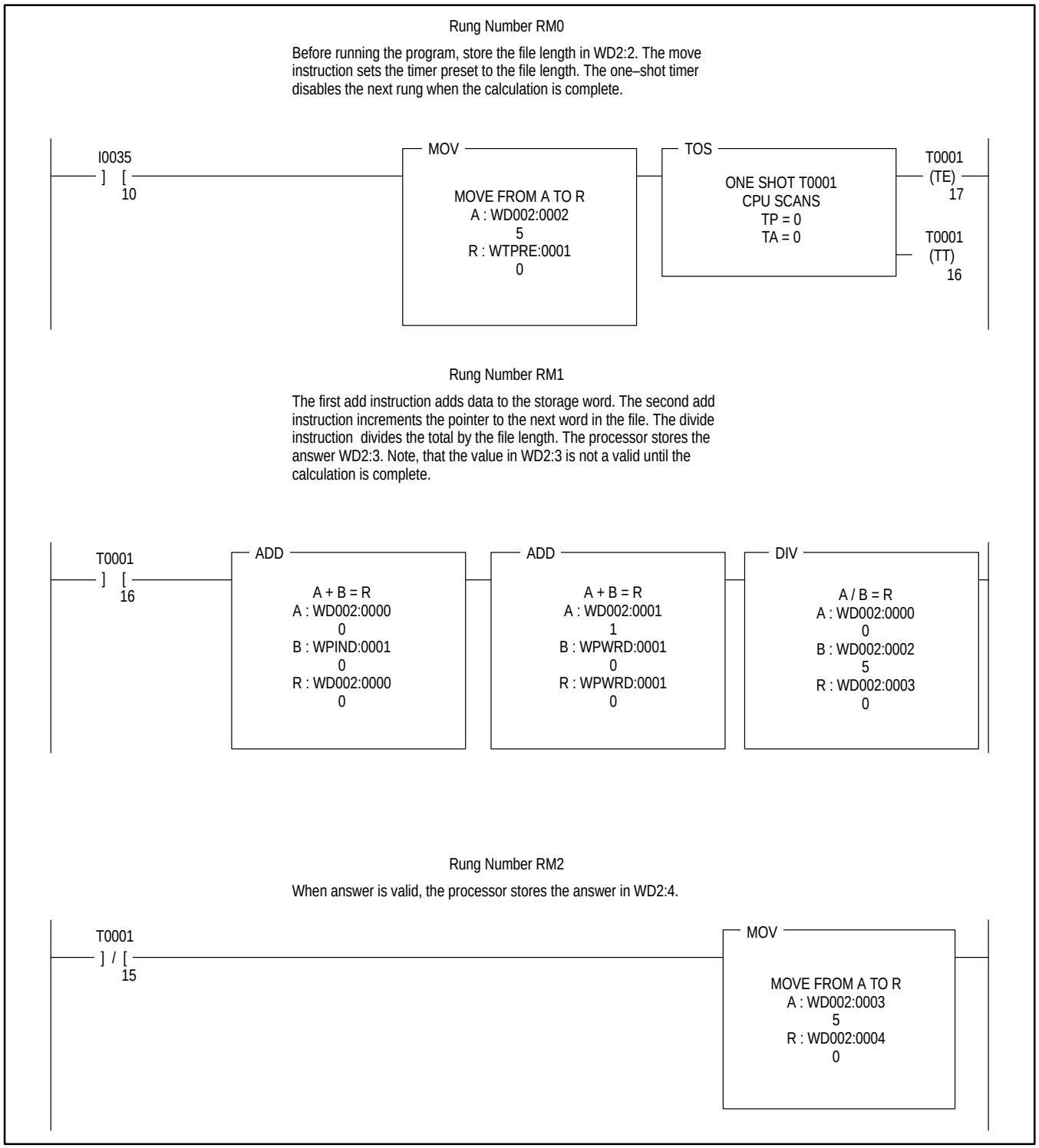
11.3 Example Pointer Using Pointers

The following sample program demonstrates how:

- to use pointers
- pointers can shorten program length

The rungs shown in Figure 11.7 calculate the average value of data in a file. We use a pointer to look at each word in the file.

Figure 11.7
Example Program Showing File Averaging Before Program Execution



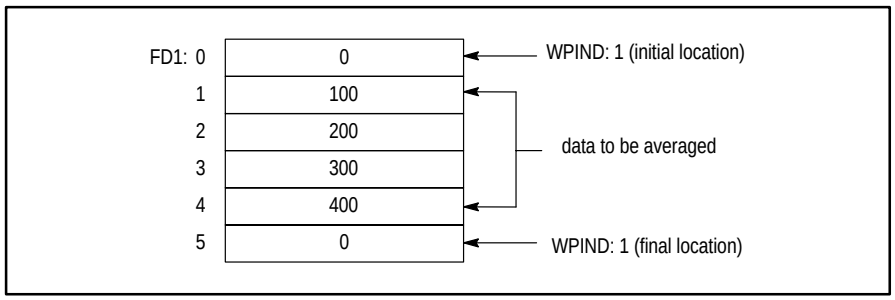
In the program, the processor:

1. increments the PWRD designation by 1 to look at each word in the file.

2. accesses the data located by the pointer with the PIND designation.

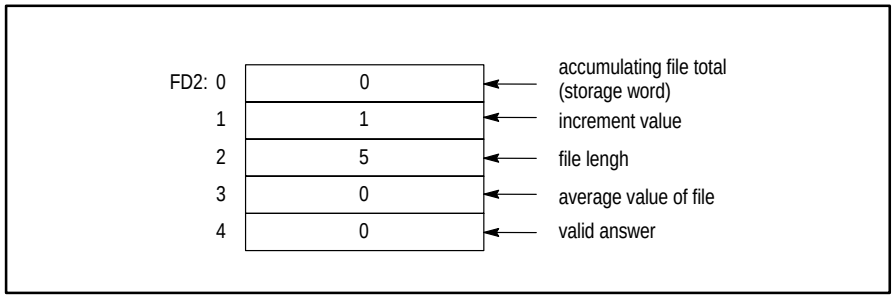
3. stores the data that is to be averaged in decimal file 1, words 0 through 4 (Figure 11.8).

Figure 11.8
The Processor Stores the Data to be Averaged in Decimal File 1



4. totals the file, divides by the file length, and stores the average value in decimal file 2, word 3 (Figure 11.9).

Figure 11.9
The Processor Stores the Values for the Calculations in Decimal File 2

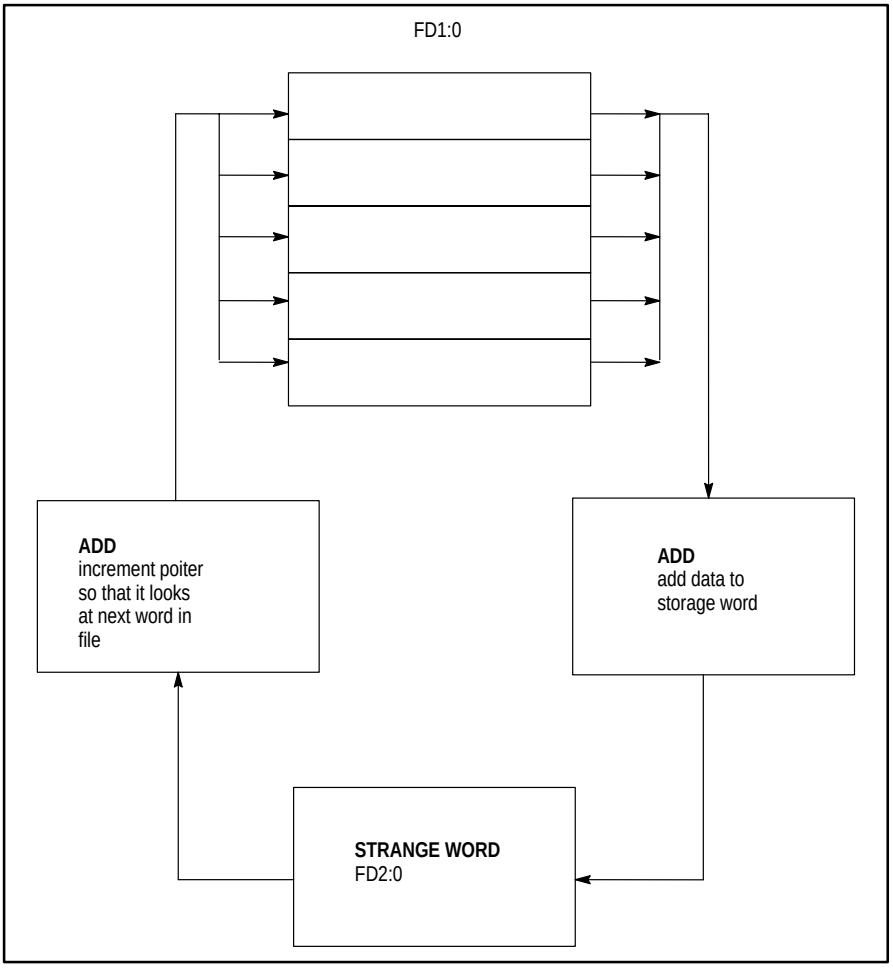


Rung 0 sets up a timer one shot so that the program only executes once. When pushbutton I035/10 is pushed, the processor moves file length into the timer preset. The timer timing bit enables rung 1 for 5 scans.

Rung 1 is enabled for 5 scans (file length = 5). The first add instruction adds the indirectly addressed data to storage word FD2:0 (Figure 11.9).

The storage word is set up to accumulate the total of all the words in the file. The second add instruction increments the pointer word so that it now points to the next word in the file (Figure 11.10).

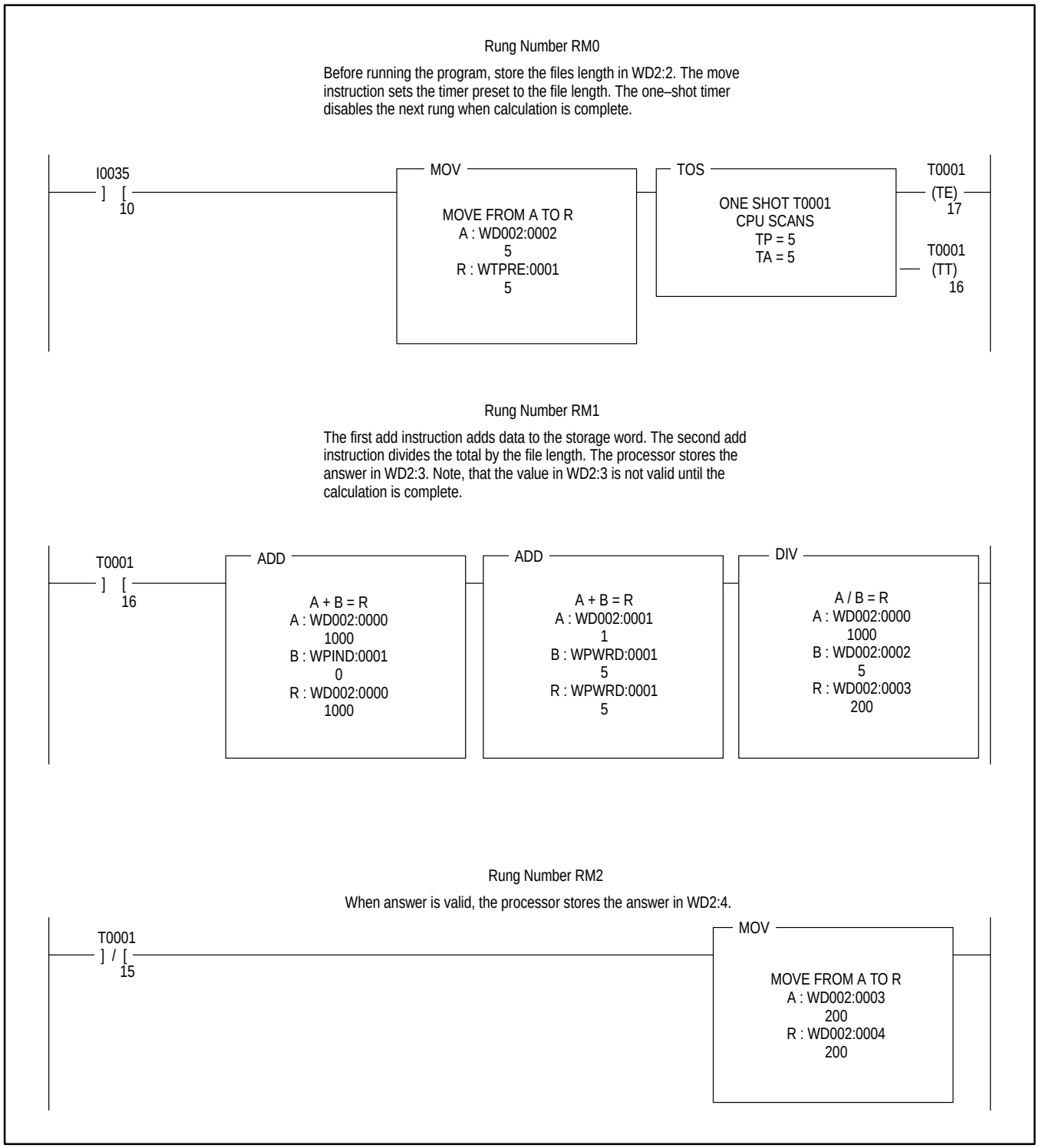
Figure 11.10
The Processor Summing the File



Important: The divide instruction divides the file total by the file length. The file average is stored in FD2:3 and is not valid until the program finishes executing. When the average calculation is complete, rung 2 moves the valid answer to WD2:4.

Figure 11.11 shows the program after execution. Word 4 in decimal file 2 contains the average value of the file.

Figure 11.11
Example Program Showing File Averaging After Program Execution



Important: Before executing this program, you must:

1. create pointer 1 (P1) in memory by using the create command.

2. enter the section (7), file (1), and word (0) values for the pointer using the data monitor (Figure 11.12).

Figure 11.12
Data Monitor for Pointer 1

START = WPSEC:0001

WORD #	SECT	FILE	WORD	ADDRESS
00001	7 = D	1	0	WD001:0000
00002				

3. set FD2:0 and FD2:3 to zero using the data monitor.

After the program executes, before executing it again, you must reset the:

- pointer to the beginning of the file
- timer
- file total storage word (FD2:0)

11.3.1 The Advantage of Using Pointers

By using pointers in this program, two important advantages are shown:

- You can easily modify this program for various file lengths by changing the value of FD2:2 (Figure 11.9). If you do not use pointers and you want to average a file that is 100 words long, you would have to program 100 add instructions.
- Your program can be more flexible. Since only three rungs are used for this program, you could easily write this program as a subroutine. Then you could average files of various lengths throughout the ladder program.

11.4 Programming Considerations for Pointers

When using pointers in your ladder program, note the following:

- You can program up to 10,000 pointers in a ladder program.
- You can program a pointer within a pointer.
- You can use pointers to indirectly address all data table sections except sections 0, 11, 14, and 15.
- You cannot program pointers down to the bit level.
- When creating a pointer, the processor allocates memory for all pointers up to the specified pointer. For example, if you create pointer 100 (P100), the processor allocates memory for pointers 0 through 100. To conserve memory, you should use pointers in numeric order starting from 0.

Chapter 12

Using Diagnostic Instructions

12.0 Chapter Objectives

In this chapter, we describe diagnostic instructions that you can use to monitor process operations. After reading this chapter, you should understand how to:

- apply diagnostics to your application
- use diagnostic instructions
- use a change-of-state diagnostic routine for your application

12.1 Applying Diagnostics

Diagnostic instructions are output instruction used for diagnosing machine or process operation by comparing an input file with a reference file for deviations. You can program the following diagnostic instructions:

- file bit compare
- diagnostic detect

Both instructions compare bits in a file of a real-time machine or process values for a mismatch with bits in a reference file. The instruction records the position and state of each mismatch in a a result word.

The difference between the file bit compare and diagnostic detect instructions is that the diagnostic detect instruction automatically changes the mismatched reference bit to the same state as the real-time bit.

To program a diagnostic instruction, you need to provide the processor with the following information:

- address of the source or input file
- address of the reference file
- address of the result word that stores the state and position of the mismatch
- counter that the processor uses to locate data in the files

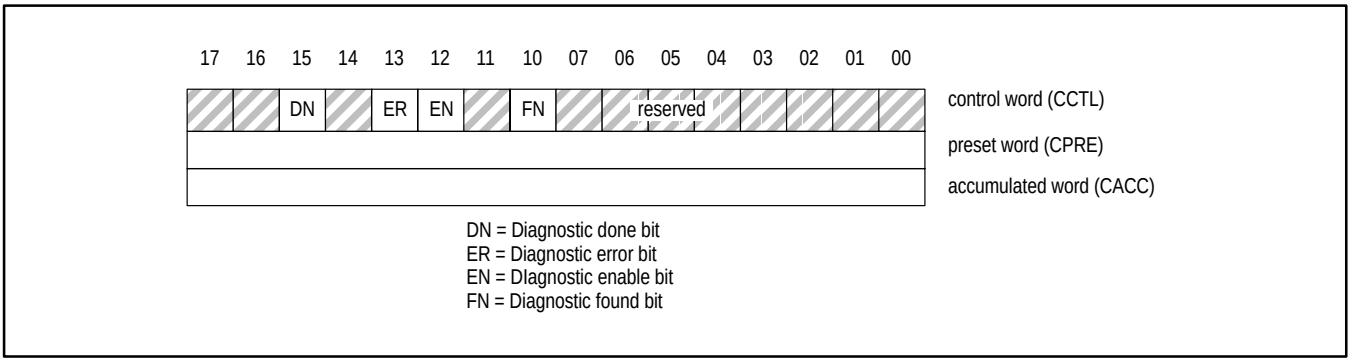
WARNING: Do not use a counter assigned to a diagnostic instruction for any other purpose. Unexpected operation could result in damage to equipment and/or injury to personnel.

- file length or the number of bits that the processor compares within the files
- file position or the bit location of the file that the processor is currently accessing. You generally enter a zero to start at the beginning of the file.
- file mode of operation for the diagnostic operation. For diagnostic instructions, you can enter:
  - ALL/SCAN for the all mode to execute the entire diagnostic operation in one program scan, or
  - #/SCAN for the numeric mode to execute the diagnostic operation on the specified number of bits each scan.

12.1.1 Counter Operation for Diagnostic Instructions

The counter for a diagnostic instruction stores the following information (Figure 12.1):

Figure 12.1
Memory Storage for Diagnostic Instructions



Control word (CCTL) – stores the control bits that reflect the status of the diagnostic instruction:

- Diagnostic Done – Bit 15 (DN) shows that a diagnostic operation is complete.
- Diagnostic Error – Bit 13 (ER) shows that an error has occurred during the diagnostic operation. You can find out the error by monitoring word 0 in status section 0, file 0 (refer to chapter 14). If an error occurs, the processor stops executing the diagnostic instruction and stores the file bit number that caused the error in the counter accumulated value word. To restart the diagnostic operation, you can reset the:

  - counter using the reset instruction to restart the entire diagnostic operation
  - error bit to restart the diagnostic operation from the point that the error occurred

- Diagnostic Enable – Bit 12 (EN) shows that a diagnostic operation is in progress.
- Diagnostic Found – bit 10 (FN) shows that a mismatch has been found.

Preset value word (CPRE) – stores the number of bits that are compared within the files.

Accumulated value word (CACC) – stores the bit position or the bit in the file that the processor is currently accessing.

12.1.2 File Bit Compare (FBC)

Required Parameters: Source file (A), reference file (B), result word (C) addresses, counter number, starting bit number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a file-bit-compare instruction goes from false to true, the processor sets the enable bit and begins comparing the bits in the source file to the bits in the reference file. The selected mode of operation determines the number of bits compared per program scan.

When the processor finds that a bit within the source file does not match the corresponding value within the reference file, it sets the found and done bits, and records the following information in the result word:

- location (bit number) in binary in lower 15 bits
- state of the source bit in the sixteenth bit

To continue the file-bit-compare operation, you must reset the done bit.

You can use the file-bit-compare instruction to flag or record any deviation from a reference file.

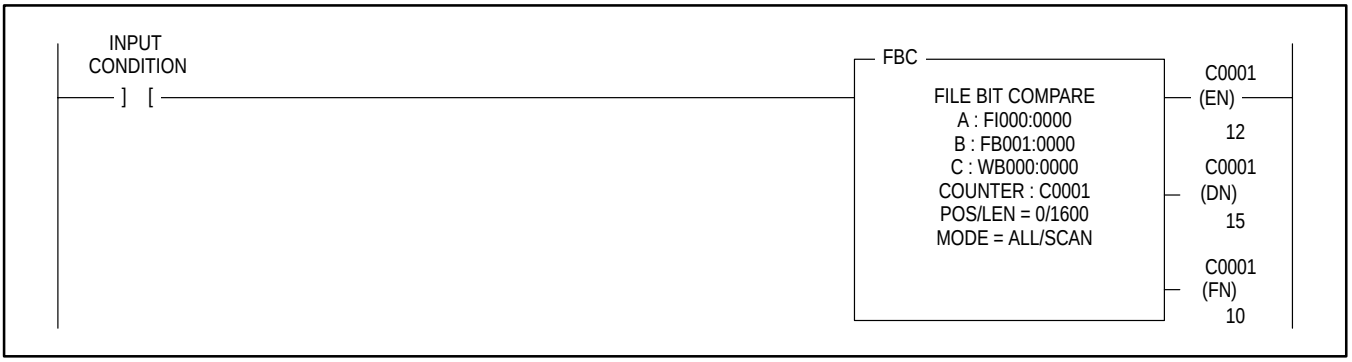
CAUTION: Do not use the floating-point or high-order-integer sections with the file-bit-compare instruction. This causes the processor to declare a bad-address-major fault by setting bit 14 in status file 0, word 1 and shut down.

Example: Figure 12.2 shows a rung containing a file-bit-compare instruction.

If the rung goes from false to true, the processor compares the bits in the source file (input file 0) to the bits in the reference file (binary file 1). If a mismatch is found, the processor stores the location and state of the source bit in the result word (binary file 0, word 0). In this file-bit-compare instruction:

This parameter	Tells the processor
counter (C1)	what counter controls the file-bit-compare operation
file position (POS=0)	to start at the first bit in the file (bit 0)
file length (LEN=1600)	to execute the file-bit-compare instruction on 1600 bits
mode (ALL/SCAN)	to execute the entire file-bit-compare instruction in one program scan

Figure 12.2
Example Rung for a File-Bit-Compare Instruction



12.1.3 Diagnostic Detect (DDT)

Required Parameters: Source file (A), reference file (B), result word (C) addresses, counter number, starting bit number (POS), length of file (LEN), mode of operation.

Description: When a rung containing a diagnostic-detect instruction goes from false to true, the processor sets the enable bit and begins comparing the bits in the source file to the bits in the reference file. The selected mode of operation determines the number of bits compared per program scan.

When the processor finds that a bit within the source file does not match the corresponding value within the reference file, it sets the found and done bits, and records the following information in the result word:

- location (bit number) in binary in lower 15 bits
- state of the source bit in the sixteenth bit

In addition, the processor changes the state of the reference bit so that it matches the corresponding source bit. To continue the diagnostic-detect operation, you must reset the done bit.

You can use the diagnostic-detect instruction in change-of-state diagnostics. We describe this diagnostic technique in the following section.

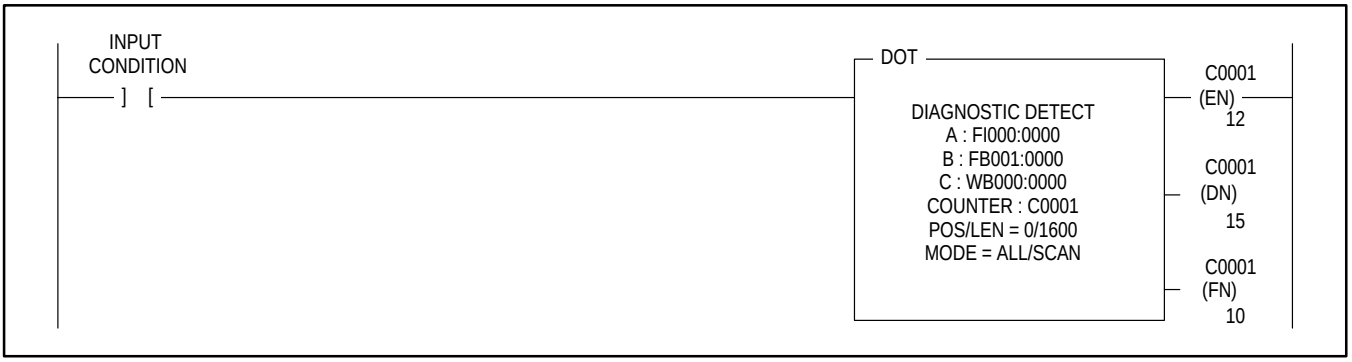
CAUTION: Do not use the floating-point or high-order-integer sections with the diagnostic-detect instruction. This causes the processor to declare a bad-address-major fault by setting bit 14 in status file 0, word 1 and shut down.

Example: Figure 12.3 shows a rung containing a diagnostic-detect instruction.

If the rung goes from false to true, the processor compares the bits in the source file (input file 0) to the bits in the reference file (binary file 1). If a mismatch is found, the processor stores the location and state of the source bit in the result word (binary file 0, word 0). it also changes the state of the reference bit so that it matches the corresponding source bit. In this diagnostic-detect instruction:

This parameter	Tells the processor
counter (C1)	what counter controls the diagnostic-detect operation
file position (POS=0)	to start at the first bit in the file (bit 0)
file length (LEN=1600)	to execute the diagnostic-detect instruction on 1600 bits
mode (ALL/SCAN)	to execute the entire diagnostic-detect instruction in one program scan

Figure 12.3
Example Rung for a Diagnostic-Detect Instruction



12.2 PLC-3 Event Driven/Change of State Diagnostic Routine

You can use the ladder program described in this section to detect input faults on both sequential and asynchronous machines. The change-of-state diagnostic routine sequentially records any inputs that change state during a repetitive machine cycle. Recorded along with each change is the rack, group, bit, and the direction of the change. Should the machine malfunction, the processor compares the recorded input changes to a reference profile of learned changes. The comparison detects the faulted inputs that are not in the proper state.

The change-of-state diagnostic program described in this section can monitor up to 2,048 inputs (16 full racks). The racks must be sequential and can start with any assigned rack number one or higher. Depending on your application, you can modify the program to accommodate fewer inputs.

Figure 12.4 shows a logic flow chart for the program and Figure 12.5 shows the ladder program. Tables 12.A and 12.B give you the memory usage and a data table map for the program. We describe the program rung by rung in the following sections.

Figure 12.4
Flow Charts for the Change-of-State Diagnostic Routine

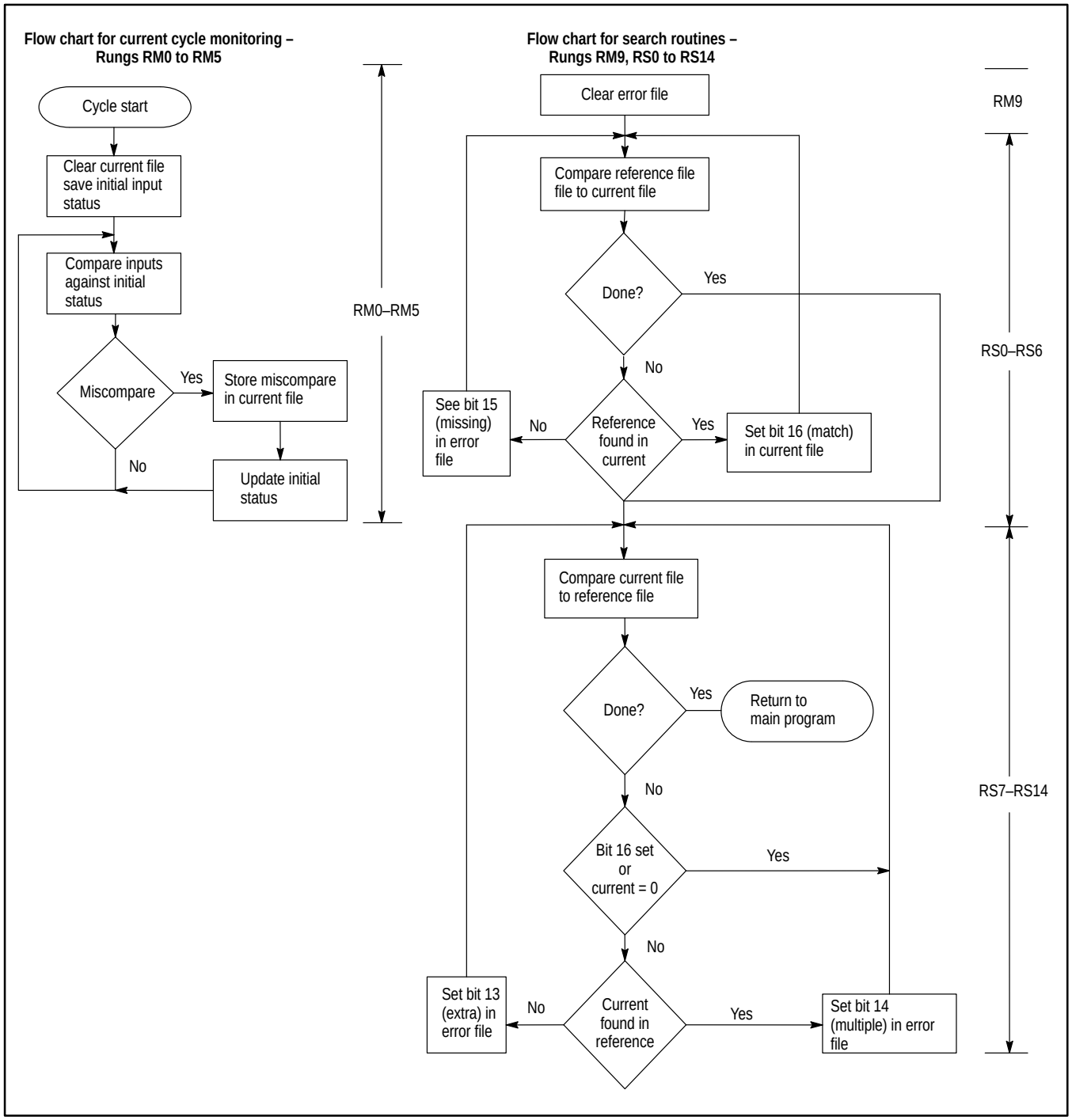


Table 12.A
Memory Usage for the Change-of-state Ladder Program

Area	Number of words used
Ladder program (E4)	450
Data table (E3)	(62+n)+((2+y)*x)) n = # of input words to be monitored x = # of transitions per sequence y = # of teach sequences
Message and system symbol (E5 and E6) for optional message procedure	200

Table 12.B
Data Table Map for the Change-of-state Ladder Program

Address	Corresponds to
I 0:0/00 I 0:0/01 I 0:0/02 I 0:0/03 I 0:0/04	cycle start switch search switch teach switch continue switch stop switch
O 0:0/00	cycle start latch output
T0001-T0004	timers for program
C0001-C0013	counters for program
PSEC:0 PFIL:0 PWRD:0 PIND:0	optional pointer used for multiple machine sequences
FB1 FB2	current file (length = # of transitions) initial status file (length = # of input words)
WB3:0 WB4:0 WB4:1 WB4:2 WB4:3 WB4:4 WB4:5 WB4:6 WB4:7 WB4:10-WB4:20	storage word storage word storage word constant 0 that you must enter constant 1 that you must enter storage word Mask = 87FF (hex) that you must enter storage word optional pointer file number (teach file #) optional message instruction control file
FB5 FB6 FB7 FB(6+n)	error file (length = # of transitions) teach file 1 (length = # of transitions) optional teach file 2 (length = # of transitions) teach file n (length = # of transitions) n = number of different machine sequences

Figure 12.5
Change-of-State Diagnostic Ladder Program

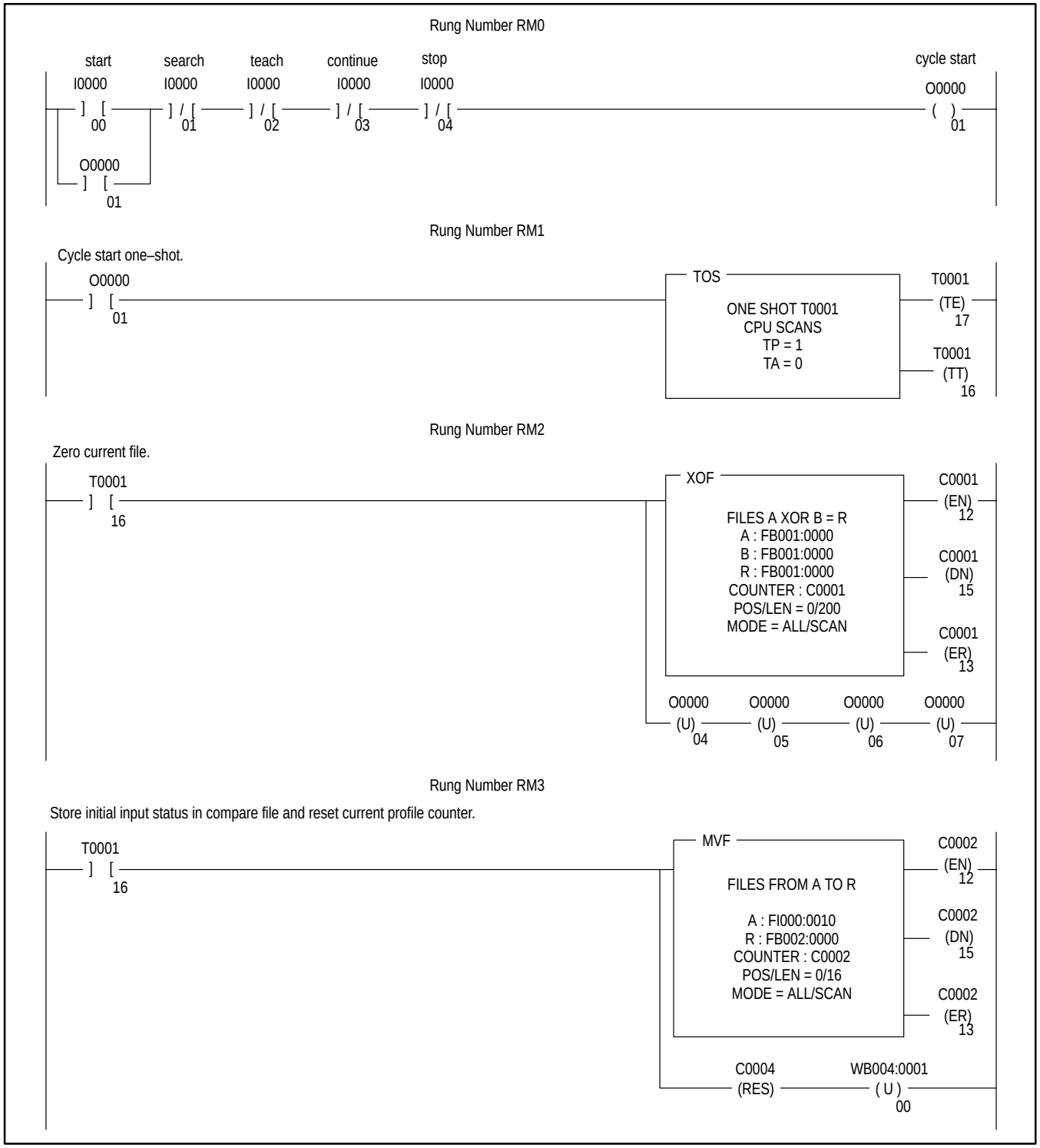


Figure 12.5
Change-of-State Diagnostic Ladder Program (continued)

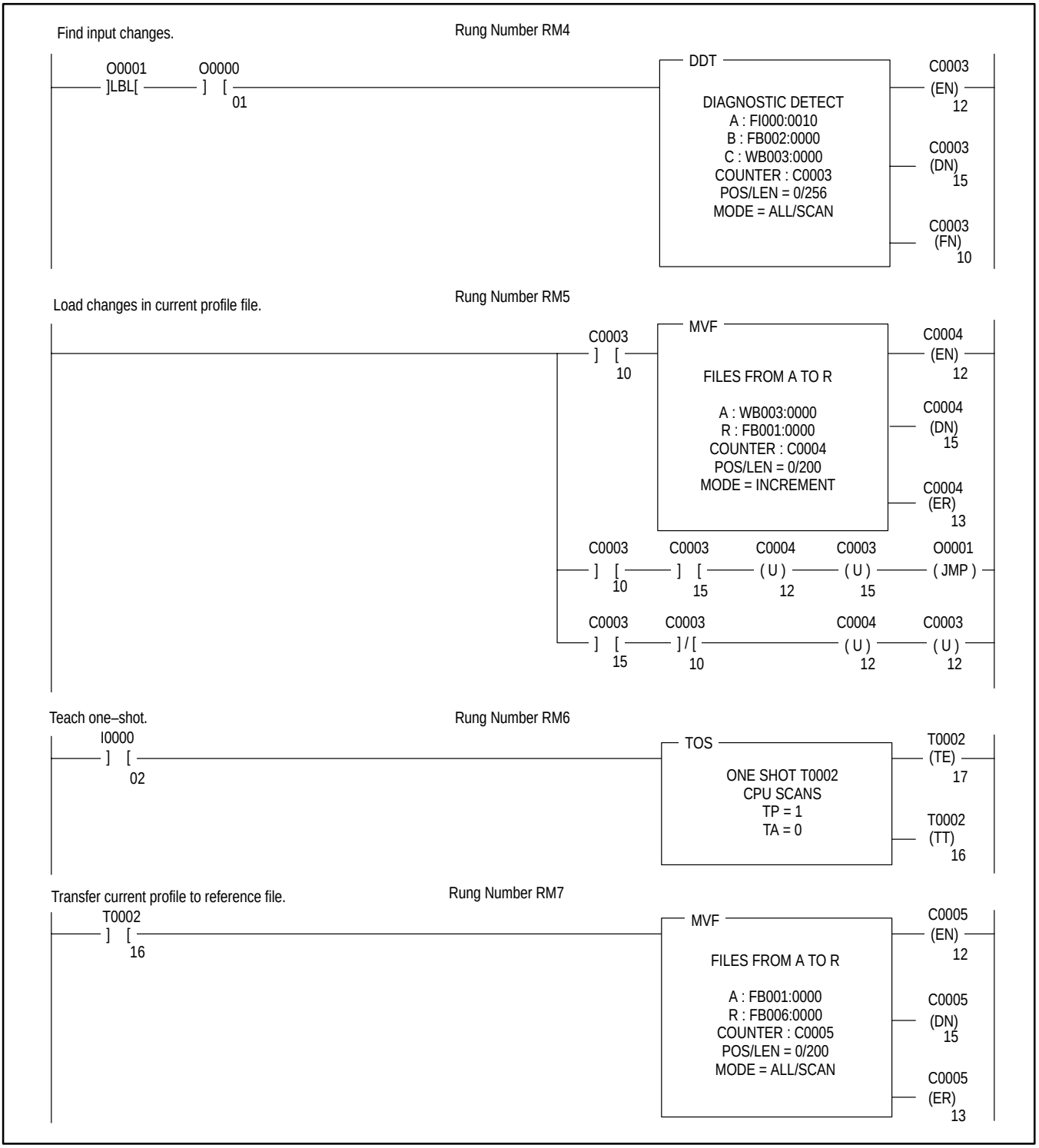


Figure 12.5
Change-of-State Diagnostic Ladder Program (continued)

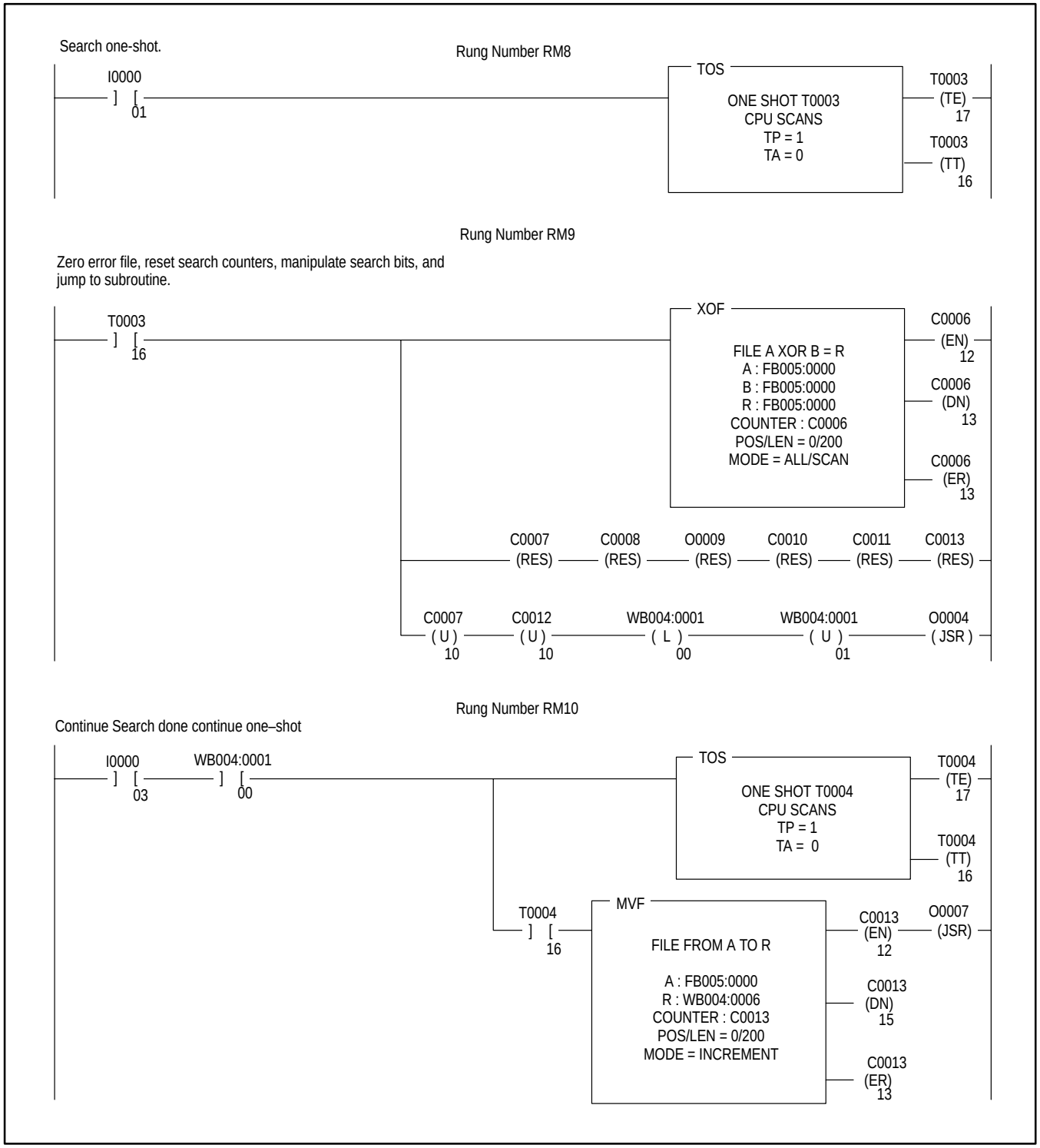


Figure 12.5
Change-of-State Diagnostic Ladder Program (continued)

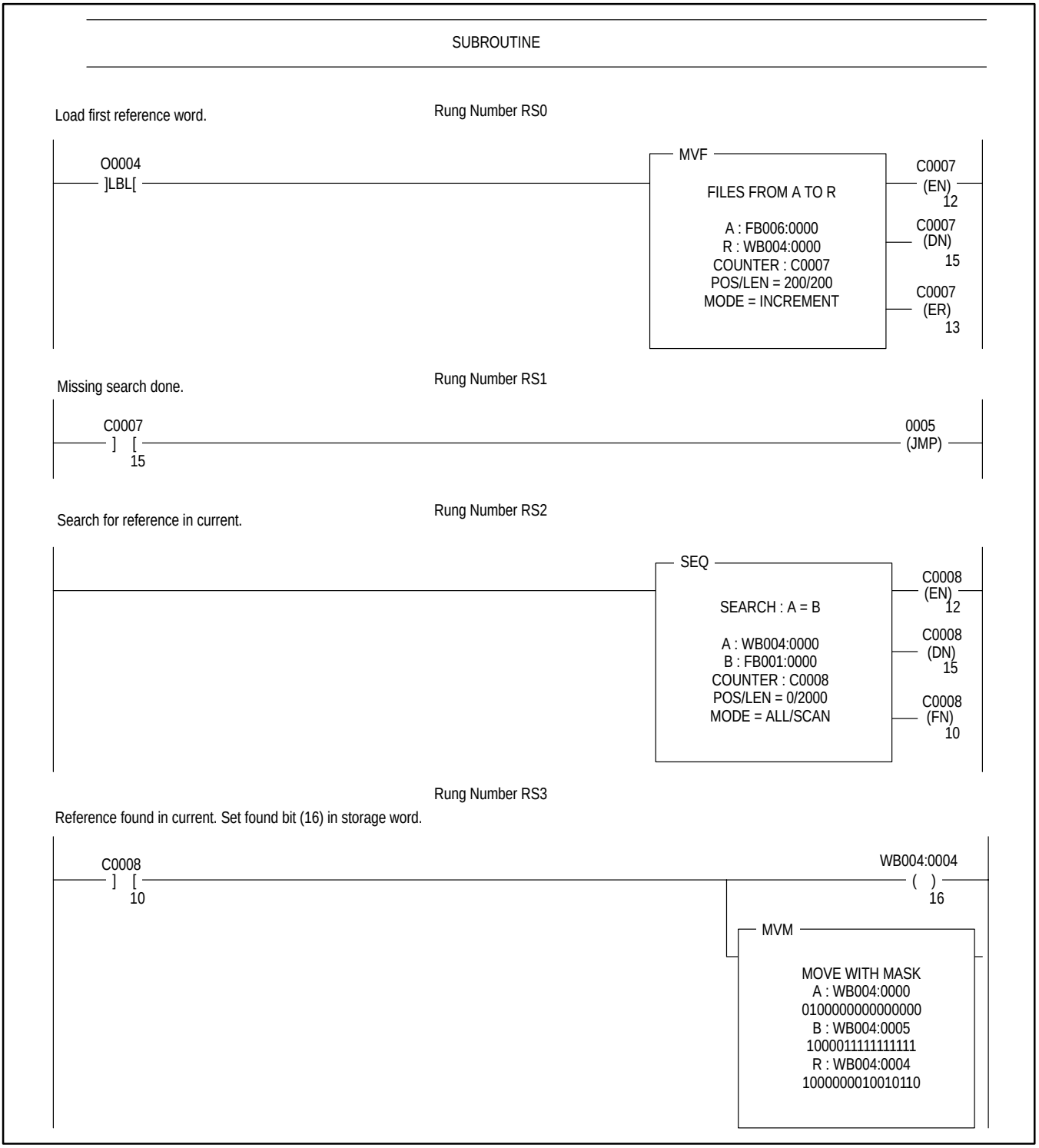


Figure 12.5
Change-of-State Diagnostic Ladder Program (continued)

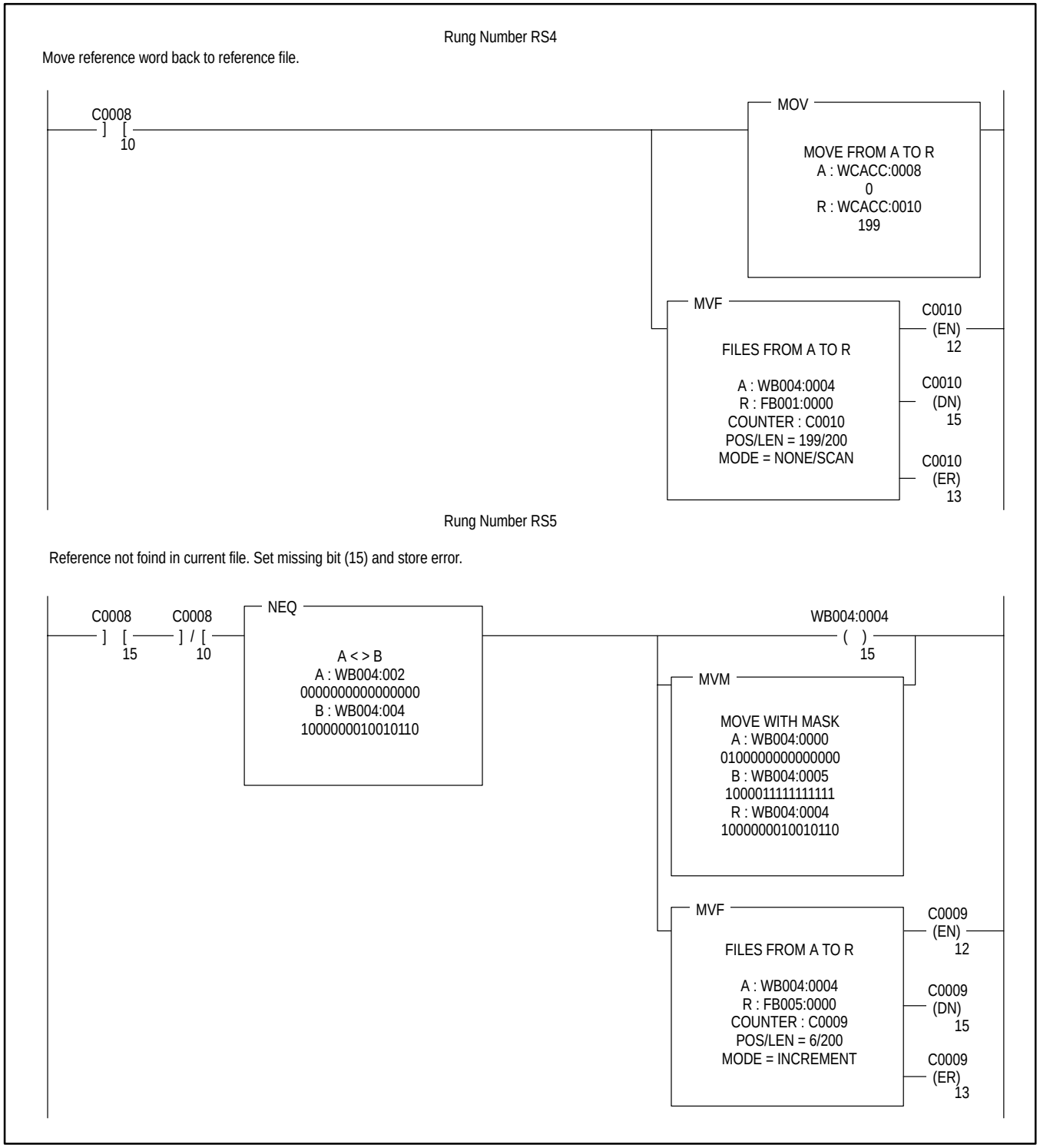


Figure 12.5
Change-of-State Diagnostic Ladder Program (continued)

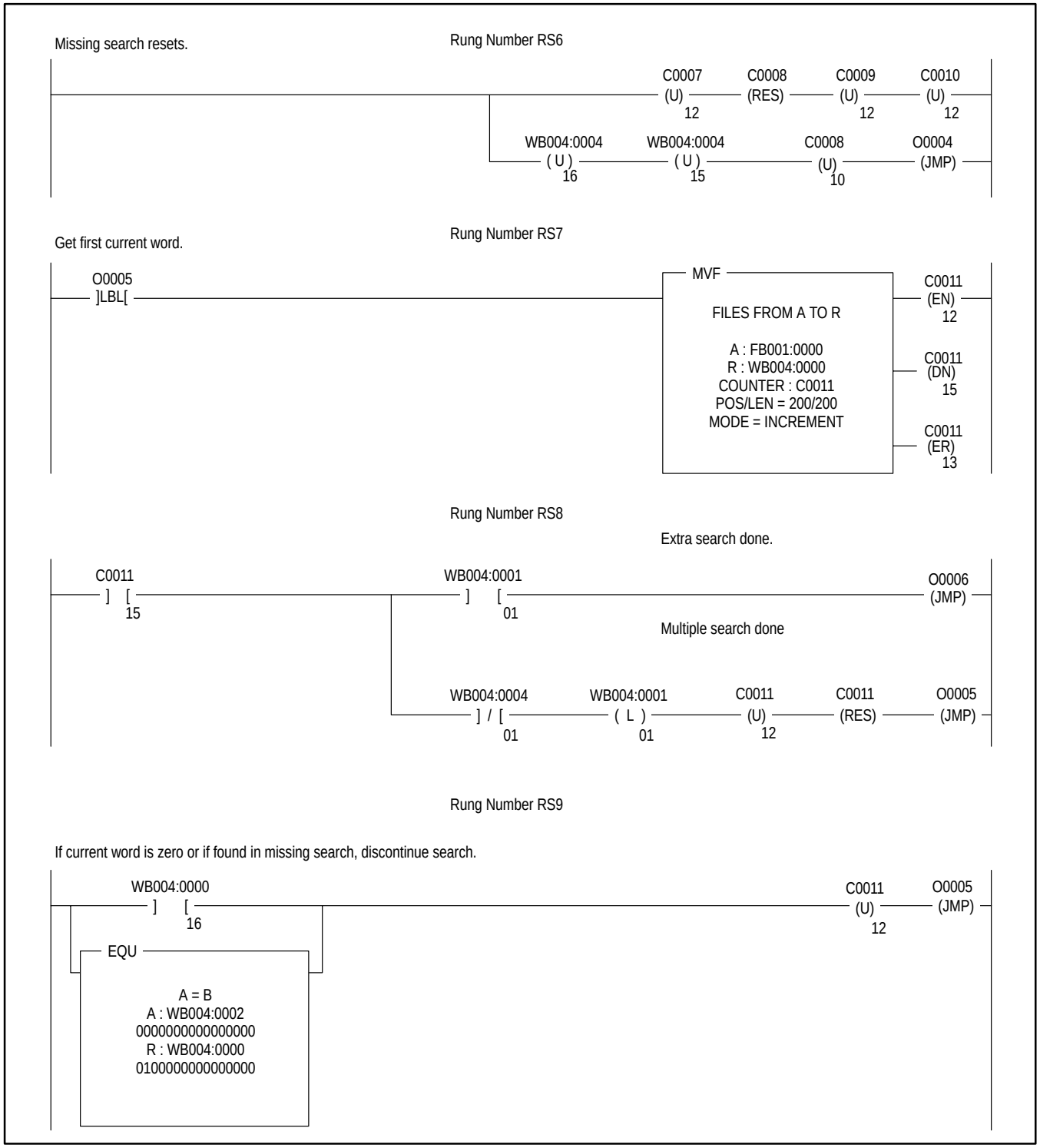


Figure 12.5
Change-of-State Diagnostic Ladder Program (continued)

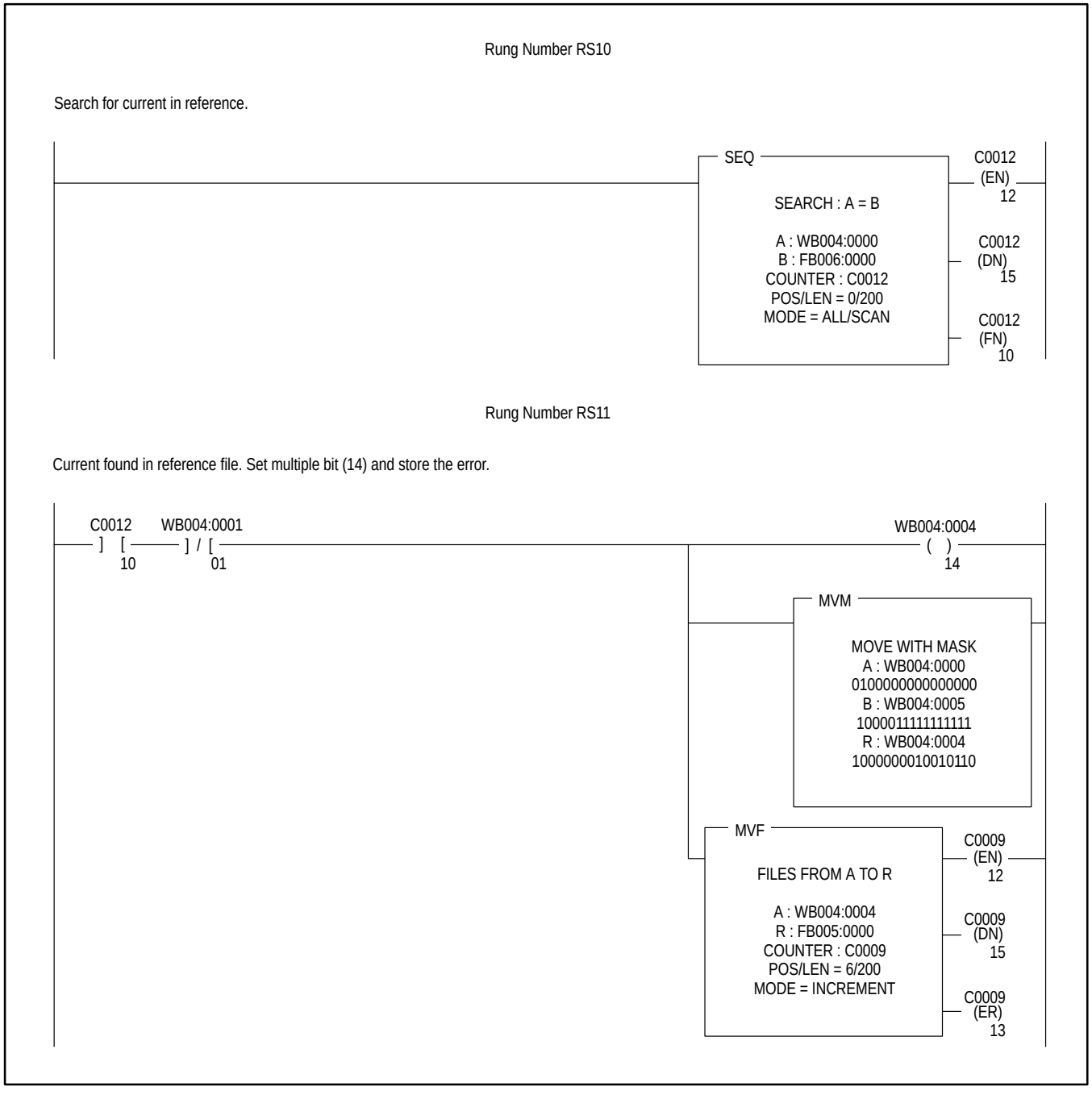


Figure 12.5
Change-of-State Diagnostic Ladder Program (continued)

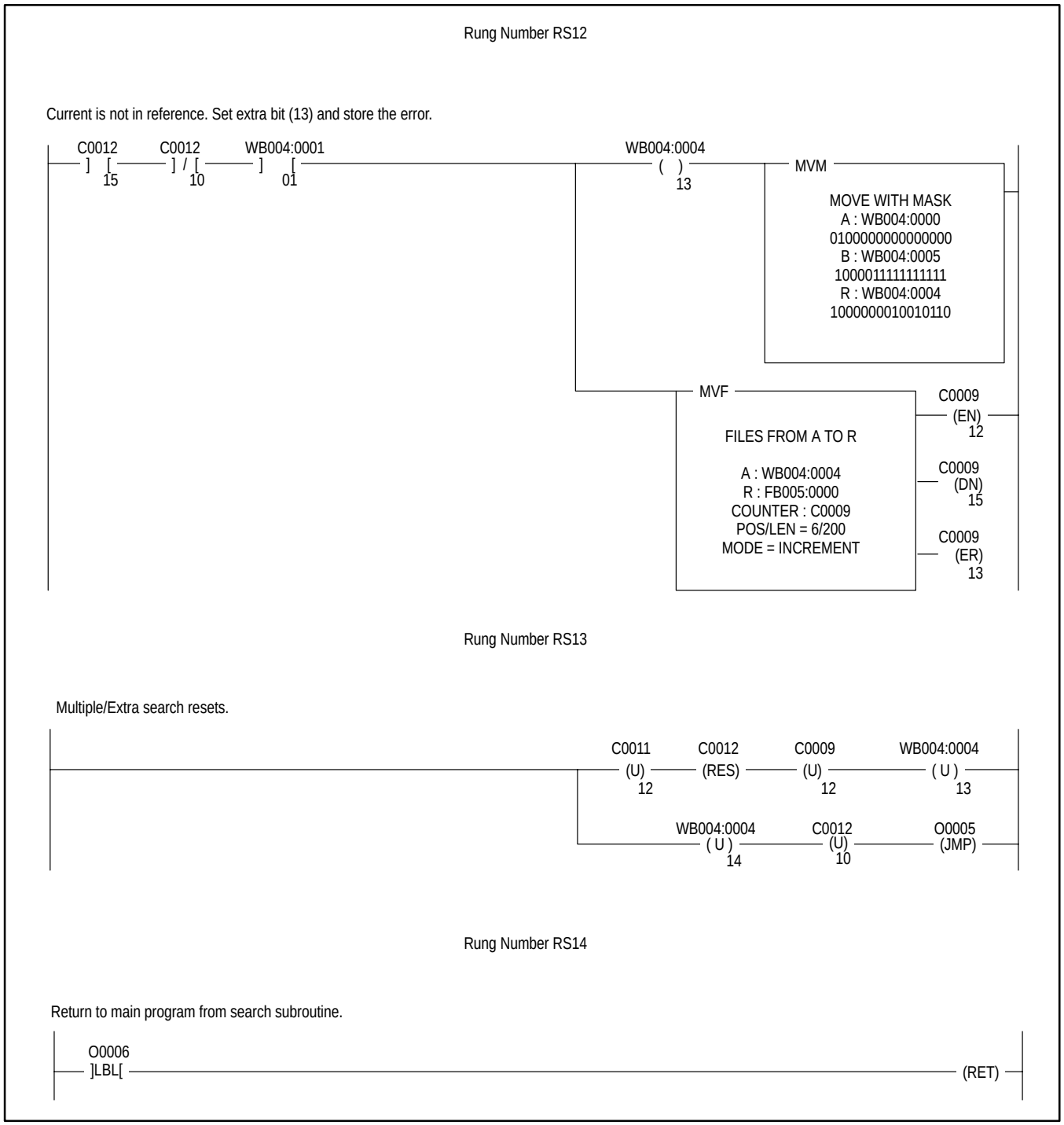
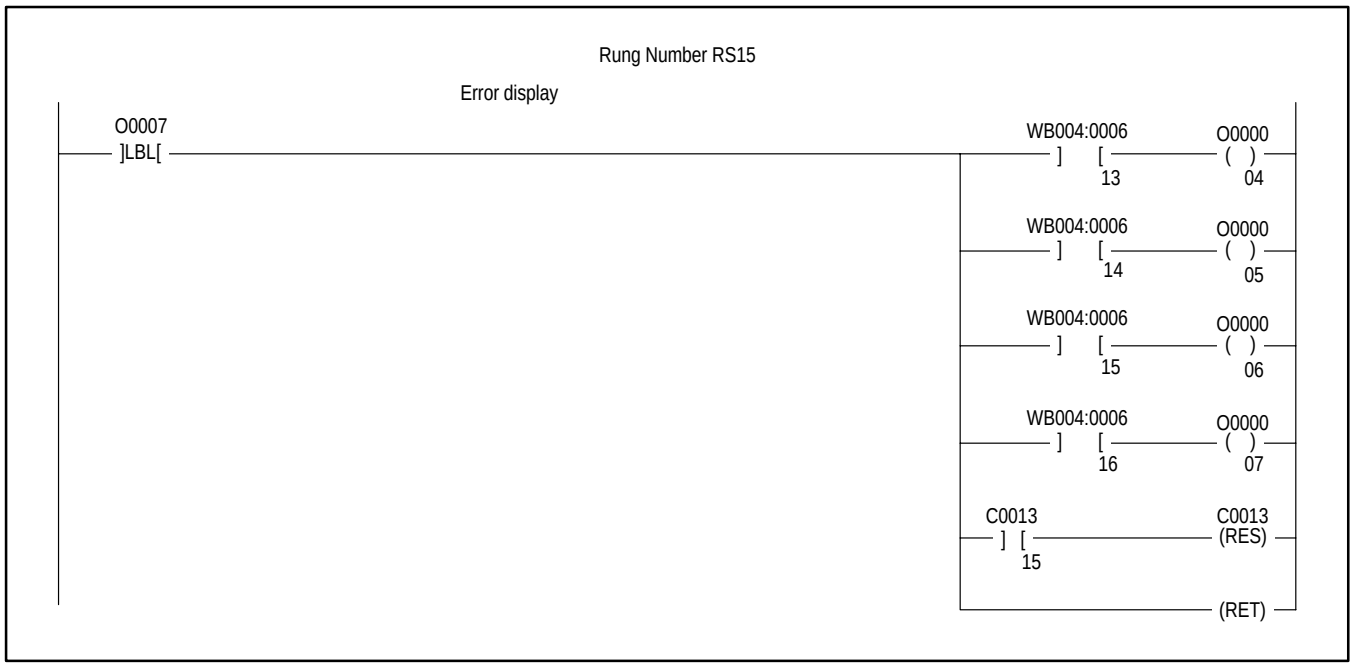


Figure 12.5
Change-of-State Diagnostic Ladder Program (continued)



12.2.1 Current Cycle Monitoring Logic (Rungs RM0 to RM5)

These rungs create a profile of input changes during the current machine cycle. The processor stores the input changes in binary file 1 (FB1):

Rung RM0 provides a seal in circuit for the machine "cycle in progress" signal using address I00/00. You set the "cycle in progress" bit from the start to the end of an automatic machine cycle. The SEARCH, TEACH, and CONTINUE contacts ensure that the machine cycle stops if these contacts turn on.

Rung RM1 provides a timer one-shot (T1) which sets its timing bit (T1/16) for one scan at the start of each cycle.

Rung RM2 provides an XOF instruction that zeros binary file 1 by exclusive or-ing it to itself. This file (FB1) stores the current machine cycle's input transitions (I/O state changes). The XOF instruction executes during the first scan of each machine cycle when T1/16 is set. The mode of operation is set for ALL/SCAN so that all the words in the file are cleared.

Important: The file size of FB1 should be large enough to accommodate the number of input transitions that occur in the machine sequence.

When the XOF instruction completes execution, the processor sets the counter done bit (C1/15) which conditions the move instruction in rung RM3. The output unlatch instructions clear the optional error outputs.

Rung RM3 provides a file move instruction that executes in the first scan of the machine cycle. The move instruction copies the initial input status into binary file 2 (FB2). This initial “snapshot” of input states is a reference that detects any input changes during subsequent scans.

Rung RM4 provides a diagnostic-detect instruction that detects inputs that change state. The DDT instruction compares bit-to-bit the inputs in the input file (FI0) to the initial “snapshot” in reference file (FB2) that was created in rung RM3. Each time that the processor detects a mismatch or a change of state in the input file, it loads the bit location and direction of change into the result word (WB3:0). This word stores the following information on the change of state:

Bit	Meaning
17	Direction of change (0 = on to off, 1 = off to on)
16	Match in current file (set in search routine)
15	Missing bit (set in search routine)
14	Multiple bit (set in search routine)
13	Extra fault (set in search routine)
12-7	Starting assigned rack number (16 max., 0-15)
6-4	Slot number (0-7)
3	Slot location (0 = lower, 1= upper)
2-0	Bits of an I/O slot

After loading the bit location and direction of change into the result word (WB3:0), the DDT instruction also updates the reference file (FB2) to reflect the change in the source file (FI0).

Rung RM5 provides a word-to-file move instruction that moves the input change information from the DDT instruction in rung RM4. Each time that the DDT instruction in rung RM4 senses an input change, the processor sets the DDT found bit (C3/10), and the MVF instruction in rung RM5 moves the bit number and state change information from word (WB3:0) into file (FB1). The MVF instruction is set for INCREMENT mode so that it sequentially loads all of the input changes into file (FB1) upon each transition of input (C3/10). As a result, the file (FB1) contains a listing or profile of each input state change that occurs during a machine cycle. The MVF counter is reset to zero at the start of each machine cycle.

12.2.2 Teach Logic (Rungs RM6 and RM7)

These rungs provide logic for automatically learning the proper input state changes:

Rung RM6 provides a timer-one-shot instruction that is conditioned by the teach input (I0/02). When the teach bit is set, the processor enables the timer-one-shot instruction by setting the enable bit (bit 16) for one scan.

Rung RM7 provides a file-move instruction that executes when the timer enable bit is set in rung (RM6). When this rung is true, the processor transfers the contents of file (FB1) to the reference storage location in binary file (FB6).

Important: The teach input (I0/02) is an input condition that you supply. You should turn on the input at the end of a known good automatic cycle.

When you turn on the teach input, the processor creates the known good profile of input changes and stores it in file (FB6). To reteach or learn a new profile, turn on the teach input again at the end of another correct machine cycle.

12.2.3 Fault Detection/Search Logic (Rungs RM8 to RM 10, RS0 to RS15)

These rungs set up and execute a subroutine that locate and display faulted inputs should the machine malfunction and stop during a cycle:

Rung RM8 provides a timer-one-shot instruction that is conditioned by the search input (I0/01). When the search bit is set, the processor enables the timer-one-shot instruction by setting the enable bit (bit 16) for one scan.

Rung RM9 provides a jump to subroutine instruction that tells the processor to execute a subroutine. The rung is designed to conserve program scan time. If the machine is operating properly, the processor does not need to scan the subroutine. The jump to subroutine instruction only executes if a fault has occurred.

If a fault occurs, the processor executes the JSR instruction and scans the subroutine specified by label (004). The detection logic consists of three sections, each identifying one of three types of inputs failures that could occur:

- missing
- multiple
- extra

Searching for Missing Faults (Rungs RS0 to RS6)

These rungs detect **missing** failures that are caused by an input that changes states during the teach mode but does not change to the same state during the current machine cycle. For example, a limit switch that fails on or off, never changes states, or never returns to its initial state.

Rung RS0 loads the first word from the current file.

Rung RS1 is the exit from the search for the missing fault sequence.

Rung RS2 uses a search-equal instruction to locate the reference file word in the current file.

Rung RS3 sets bit 16 in the reference word if it is in the current file.

Rung RS4 sets bit 16 in the reference file indicating a find in the current file.

Rung RS5 sets bit 15 in the reference file if the reference word is not in the current file RS5), bit 15 is set in the reference file.

Rung RS6 resets the missing fault search sequence.

Search for Multiple and Extra Faults (Rungs RS7 to RS14)

These rungs detect **multiple** and **extra** failures. An input that recycles causes multiple failures. For example, a limit switch that creeps off its position and as a result changes states several times. An input that does not change during the teach mode but changes during the current machine cycle causes extra failures. For example, an operator depressing the wrong pushbutton, or an auxiliary contact from a motor starter tripping because of an overload.

The program searches the entire current file for multiple faults and then repeats the routine for extra faults.

Rung RS7 loads the words from the current file into storage word (B4:0) sequentially.

Rung RS8 is the exit from the search for the multiple and extra fault sequences. The processor sets bit (B4:1/01) when both searches are completed.

Rung RS9 determines whether a search for fault is required on the current word based on the results of the search for missing faults.

Rung RS10 searches the reference file for the current word.

Rung RS11 identifies and saves multiple faults.

Rung RS12 identifies and saves extra faults.

Rung RS13 resets the bits for the next multiple or extra search.

Rung RS14 exits the search routine and returns to the main program.

Displaying Fault Logic (Rungs RM10 and RS15)

At the completion of the search cycle, the processor stores the errors in file (FB5) in the following order:

1. missing

2. multiple

3. extra

Important: The change-of-state diagnostic routine does not detect inputs that occur out of sequence.

Rungs RM10 and RS15 load the input faults into word (B4:6), one at a time. Each toggle of the continue input (I0:0/03) loads each error into the word. The display rungs also set the following bits to specify the type of fault:

The processor sets this bit	To indicate this type of fault
00:0/06	missing
00:0/05	multiple
00:0/04	extra

The processor also sets bit O0:0/07 to show the state of the faulted input. These bits are reset in rung RM2. You can remove the display rungs RM10 and RS15 along with the bit resets in rung RM2 without impacting program operation.

12.2.4 Multiple Machine Sequences

You can use the change-of-state diagnostic routine on applications that have more than one machine sequence. Such applications require a unique teach file for each sequence. All other files remain the same. The search mode needs to compare the current file with the proper teach file. This can be accomplished using a pointer.

Figure 12.6 shows four rungs of logic from the change-of-state diagnostic routine. Note that the rung numbers originate from different parts of the program. These rungs have been modified to accommodate multiple

machine sequences. The multiple machine sequence modification to the change-of-state diagnostic routine uses pointer indirect (PIND:0) which indirectly addresses (B6:0). You set the initial address in the data table for the pointer using the data monitor:

- PSEC:0 = 8
- PFIL:0 = 6
- PWRD:0 = 0

In Figure 12.6, rung RM1 includes a move instruction on the timer one-shot. The move instruction moves from word (B4:7) to word (PFIL:0) and loads the proper teach file number into pointer zero. You must provide the proper file number to the word (B4:7). The file number depends on the machine sequence that executes. The teach file is in file (FB6).

Rungs RM7, RS0, and RS10 replace the fixed teach address (FB6) with the variable pointer address (FPIND:0). In executing the search, the processor compares the proper teach file to the current file, and stores the mismatches in the error file (FB5).

Figure 12.6
Altering the Change-of-state Program for Multiple Machine Sequences

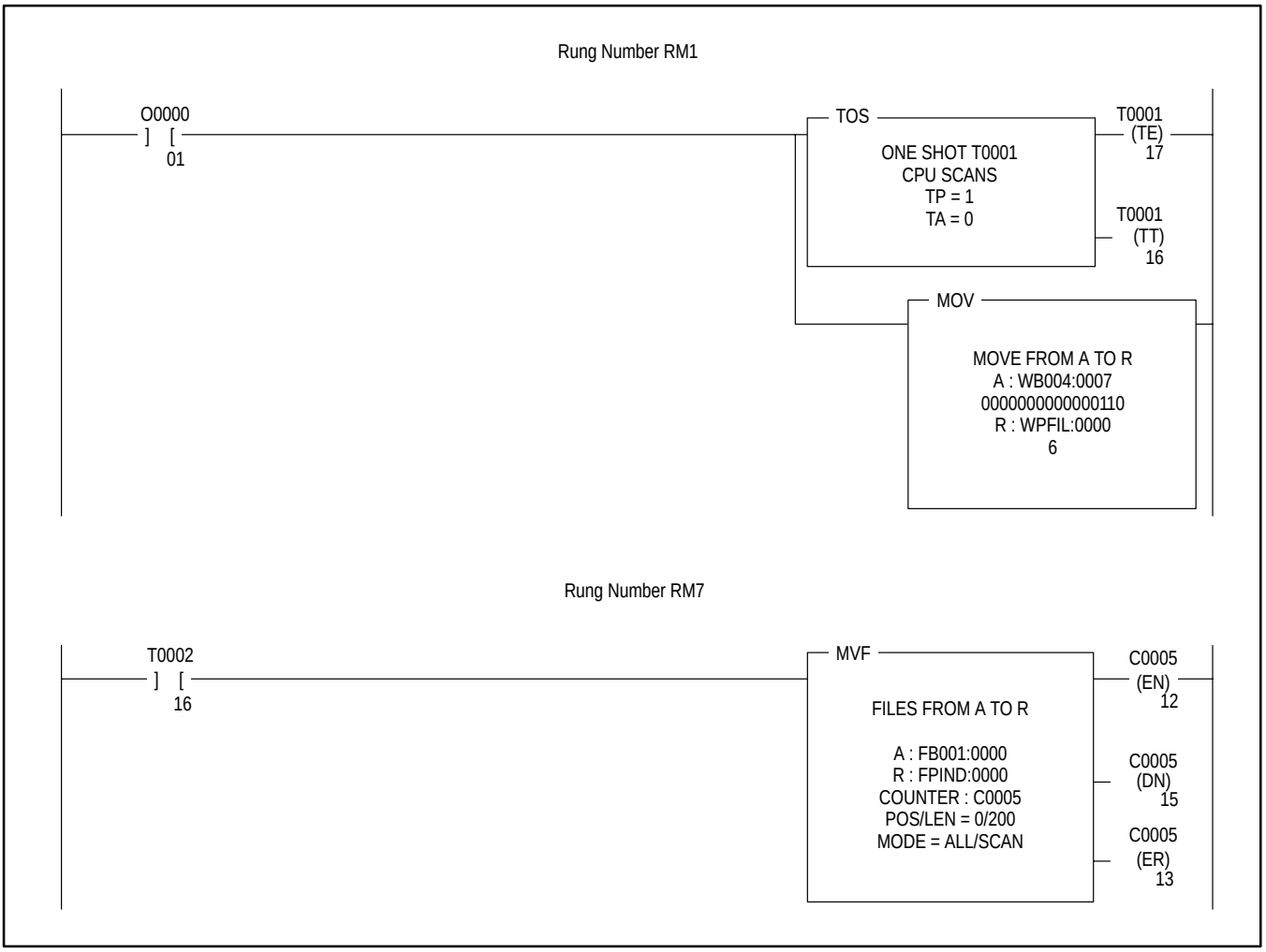
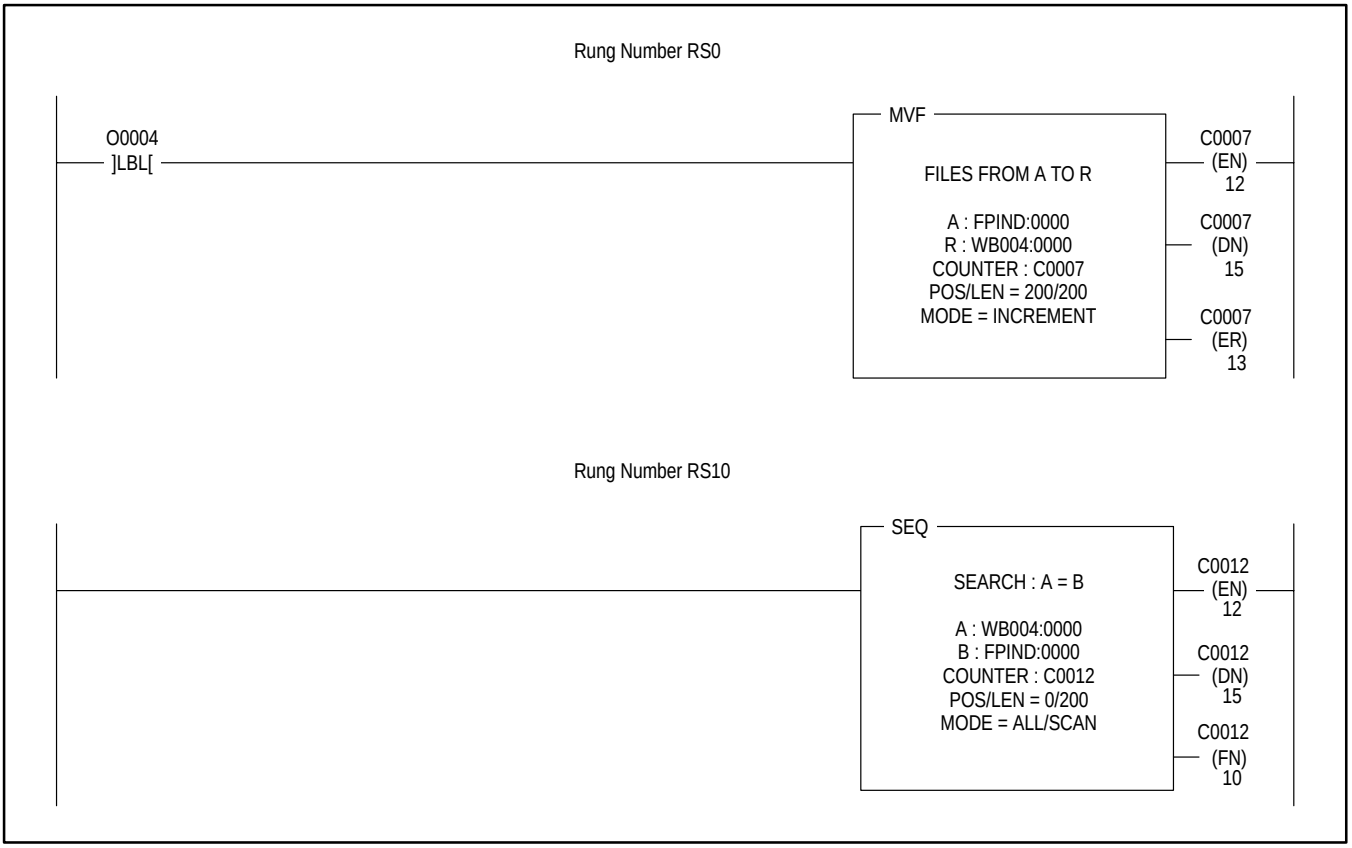


Figure 12.6
Altering the Change-of-state Program for Multiple Machine Sequences (continued)



12.2.5 Generating Reports on Input Faults

Figure 12.7 shows a rung and a message procedure for generating reports on input faults. You can place the ladder rung anywhere after rung RM9. The search one-shot bit (T3/16) enables the message instruction which executes the message procedure @COS. You can execute the message procedure on a I/O Scanner-Message Handling Module (cat. no. 1775-S4B) or a Peripheral Communication Module (cat. no. 1775-GA).

Figure 12.7
Message Procedure that Generates a Report on Input Faults

The following message procedure @COS accesses data stored in the error file and prints it out in a usable form.

```
1     A = $CPRE:6                                   ; Determine length of error file
2     B = 0                                         ;Establish constants
3     Y = 1920
4     Z  = 112
5     V = 1                                         ; Starting rack number
6     P 'INPUT FAULTS'                              ;Print header
7     P 'RACK#  GROUP#  BIT#  STATE  TYPE'
8     L:
9     IF ($B5:(B) . EQ. 0) GO L1                    ; If no errors, exit
10    STATE = ? 'OFF'                               ; Determine state
11    IF($B5:(B)/017) STATE =? 'ON'
12    TYPE =? 'MISSING'                             ;Determine error type
13    IF ($B5:(B)/014) TYPE =? 'MULTIPLE'
14    IF ($B5:(B)/013) TYPE =? 'EXTRA'
15    C = $B5:(B)                                   ;Access error file
16    E = (C .BAND. Y)                              ;Decode rack#
17    E1 = (E>>7)
18    RACK = E1 + V
19    F = (C .BAND. Z)                              ;Decode group#
20    GROUP = (F >> 4)
21    SLOT = $B5:(B)/3
22    BIT = (C .BAND. 7)
23    IF (B .EQ. A) GO L1                           ; If end of file exit
24    P '  'RACK'          'GROUP'          'SLOT"BIT'       'STATE'       'TYPE
25    B = B + 1                                     ;Increment to next error word
26    GO L
27    L1:
```

Sample Output:

INPUT RACK#	FAULTS GROUP#	BIT#	STATE	TYPE
3	1	01	ON	MISSING
1	1	02	ON	MISSING
6	0	01	OFF	MISSING
1	1	02	OFF	MISSING
0	7	00	ON	MULTIPLE
1	1	00	OFF	MULTIPLE
0	0	10	ON	EXTRA
5	1	06	ON	EXTRA

Chapter 13

Controlling Ladder Program Execution

13.0 Chapter Objectives

In this chapter, we describe instructions and features that you can use to control ladder program execution. After reading this chapter, you should:

- understand how the program control instructions work
- distinguish between a master-control-reset and a jump-to-label instruction
- understand the methods for recovering from major faults
- understand the real-time-interrupt feature
- understand the context function

13.1 Applying Program Control Instructions

You can use program control instructions in the ladder program to tell the processor to treat a portion or portions of the program in a different way. For example, you can use program control instructions to:

- turn off all non-retentive outputs in a section of a ladder program
- jump over a section of a program that does not need to be executed when certain conditions are true
- jump to the subroutine section of the ladder program, execute a subroutine, then continue executing the main program

The program control instructions include:

- master control reset
- jump to label
- label
- jump to subroutine
- return
- no operation
- end

13.1.1 Master Control Reset (MCR)

Required parameters: None

Description: You can use master-control-reset instructions to create program zones:

If the MCR rung is	Then the processor
true	executes the rungs in the master-control-reset zone based on each rung's individual input conditions.
false	resets all non-retentive output instructions in the master-control-reset zone regardless of each rung's individual input conditions.

In programming master-control-reset instructions, note that:

- You cannot nest one master-control-reset zone within another.
- If a master-control-reset zone continues to the end of the ladder program, you do not have to program a master-control-reset instruction to end the zone.

In many situations, an overriding set of conditions may be necessary for the entire controlled process, involving many output devices. By using master-control-reset instructions in the ladder program, you can control these devices.

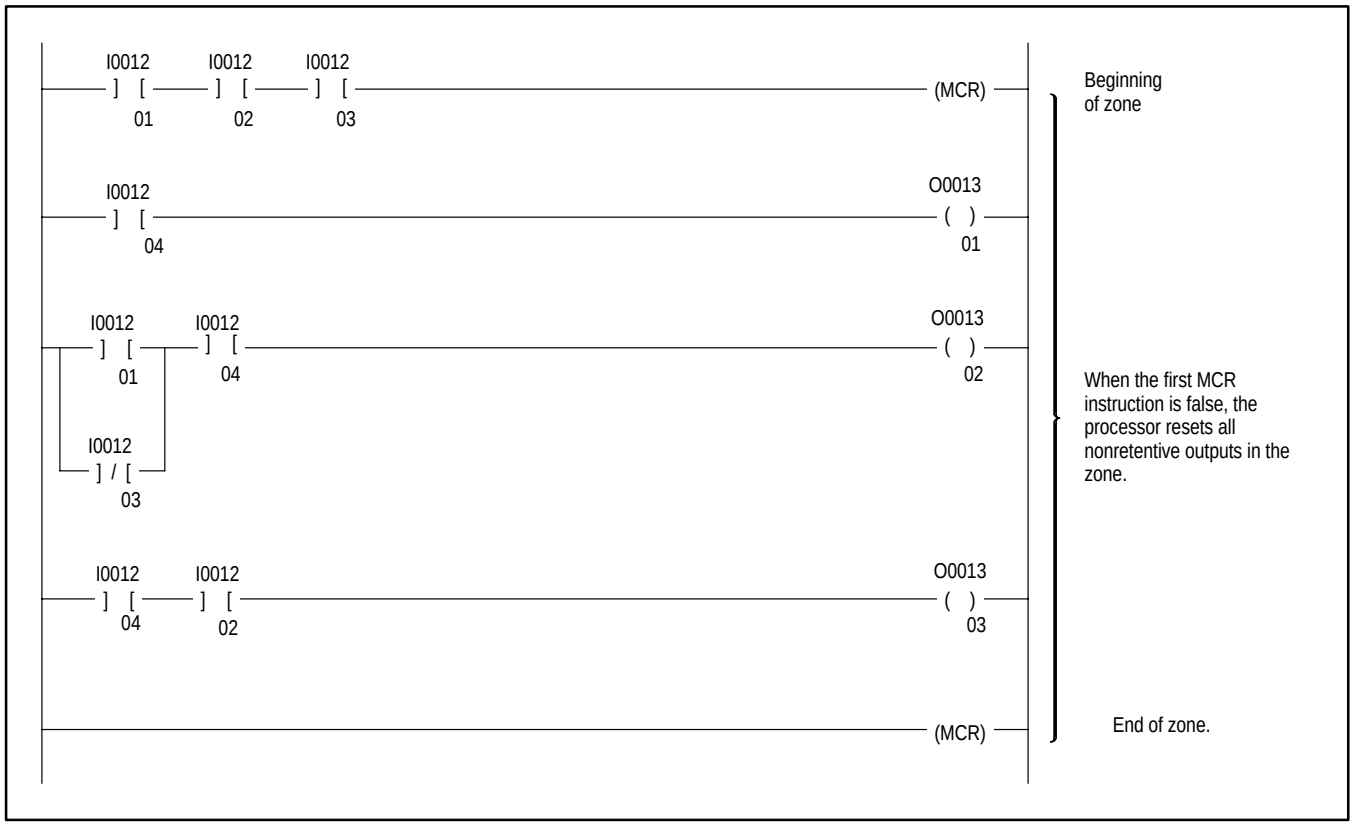
Relay control systems often use one master relay to control the overall function of several operations, based on certain necessary conditions. This control action is similar in some respects to the function of the master-control-reset instruction.

However, the master-control-reset instruction is not a substitute for a hard-wired master control relay that provides emergency stop capability to the application.

WARNING: We strongly recommend that you install a hard-wired master control relay to provide emergency I/O power shutdown. The wiring for this device is described in the PLC-3 Controller Installation and Operation Manual (publication 1775-6.7.1).

Example: Figure 13.1 shows an example master-control-reset zone. Notice that master-control-reset instructions begin and end the zone. If the rung containing the first master-control-reset instruction is true, the processor executes the rungs in the master-control-reset zone based on the rung input conditions. Otherwise, the processor resets the non-retentive output instructions in the master-control-reset zone.

Figure 13.1
Example Rungs Showing a Master-Control-Reset Zone



13.1.2 Jump to Label (JMP)

Required Parameters: Label number

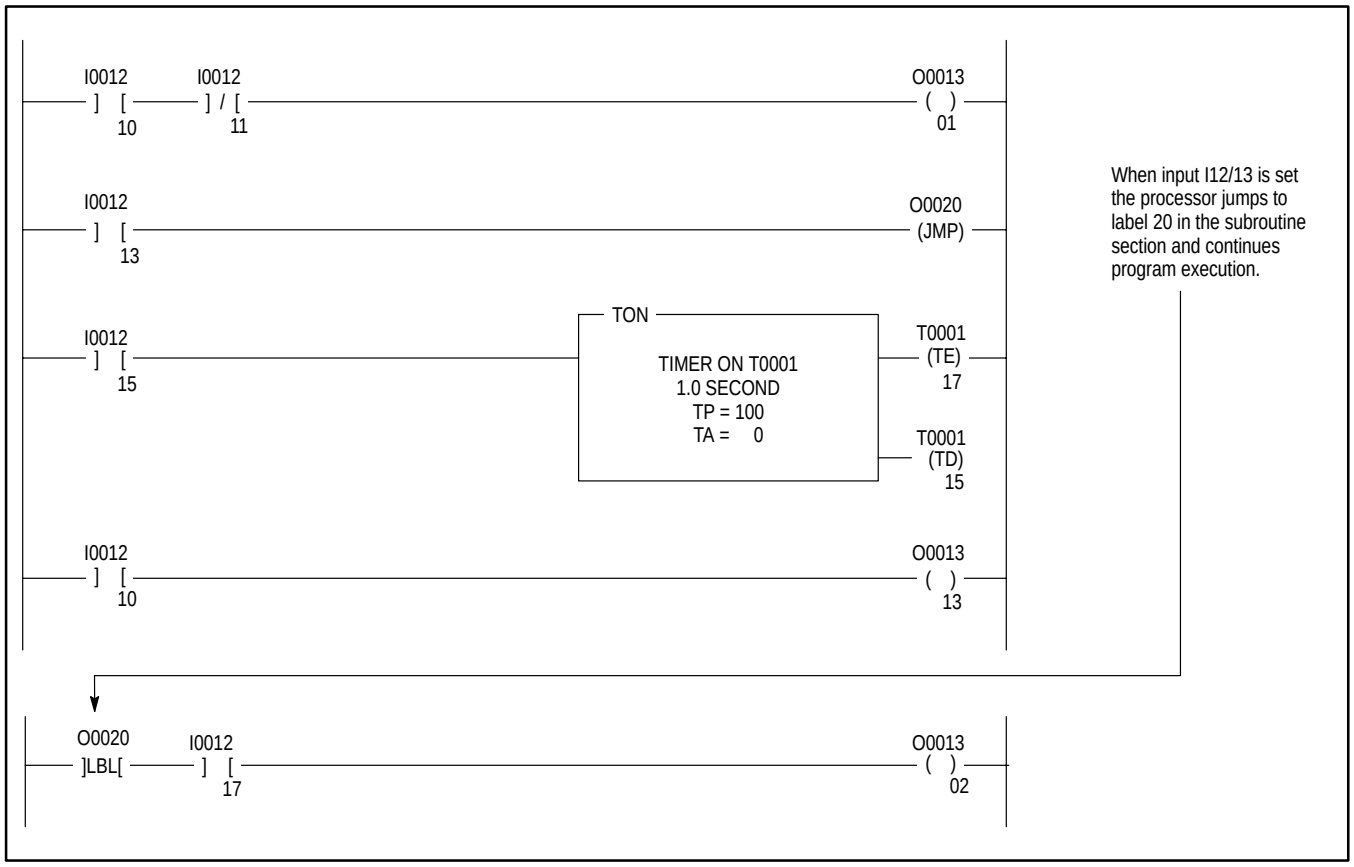
Description: You can use the jump and label instruction to skip a portion of the ladder program:

If the jump rung is	Then the processor
true	skips the rungs between the jump rung to the label rung and continues the program scan by executing the rung containing the label
false	ignores the jump instruction and executes the next rung

The label instruction tells the processor where to jump to in the program.

Example: Figure 13.2 shows rungs that use the jump instruction.

Figure 13.2
Example Rungs Using the Jump-to-label Instruction



13.1.3 Label (LBL)

Required Parameters: Label number (0 - 255) and/or comment number (0 - 32,767).

Description: The label instructions has three uses:

- marking a rung in the ladder program as the target for a jump-to-label or jump-to-subroutine instruction
- marking fault and real-time-clock-interrupt routines
- adding comments or application type information to selected rungs

When used as a jump target, the label instruction must be the first instruction on a rung and requires a label number that tells the processor where to jump to in the ladder program:

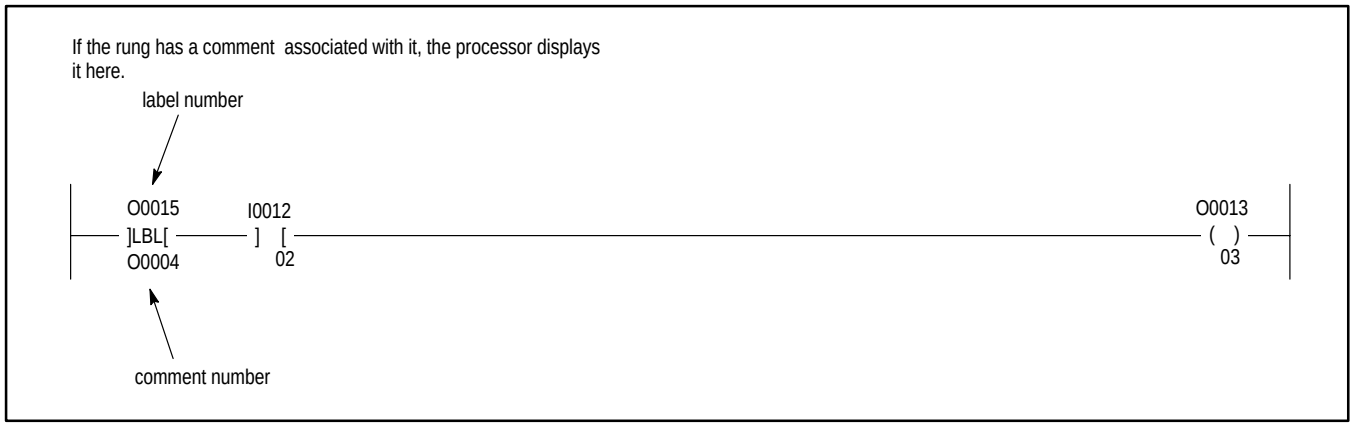
Label Numbers	Used with
2-255	jump-to-label and jump-to-subroutine instructions. When the processor encounters one of these instructions, it jumps to the label instruction having the same label number and continues executing.
0	fault routine. When the processor declares a major fault, it goes to label 0 and executing the fault subroutine (refer to section 13.2.1).
1	real-time clock interrupt subroutine. The processor goes to label 1 in the subroutine section and executes it at a time interval that you specify through the LIST function (refer to section 13.3).

You can also specify a rung comment number (0 to 32,767). Then, through the program loader, you can enter a comment or application information to correspond to this number so that you can fully document your ladder program. For detailed information on entering rung comments, refer to the user's manual for your program loader.

Important: Do not program labels in parallel branches.

Example: Figure 13.3 shows a rung containing a label. In this rung, 15 is the label number and 4 is the comment number.

Figure 13.3
Example Rung for a Label Instruction



13.1.4 Jump to Subroutine (JSR)

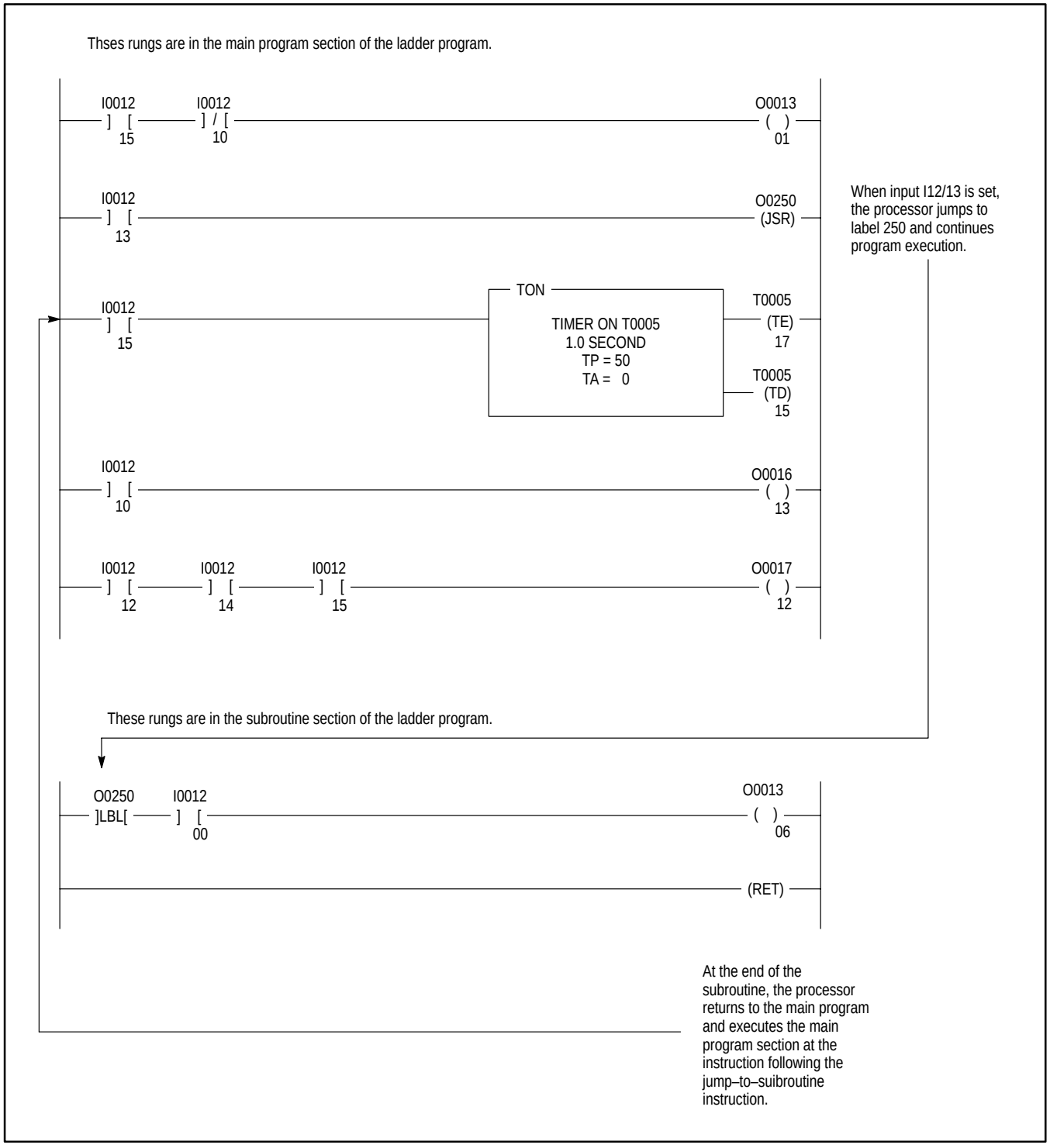
Required Parameters: Label number

Description: As we described in chapter 4, the ladder program consists of the main program, subroutine, and fault routine sections. When a rung containing a jump-to-subroutine instruction is true, the processor moves to the subroutine section and executes the subroutine defined by the label number.

Within the subroutine section, a label instruction with the same label number as the jump-to-subroutine instruction in the main program section begins the subroutine. A return instruction ends the subroutine and tells the processor to move back to the main program section. You can nest subroutine execution to 32 levels.

Example: Figure 13.4 shows example rungs for a subroutine.

Figure 13.4
Example Rungs for a Subroutine



13.1.5 Return (RET)

Required Parameters: None

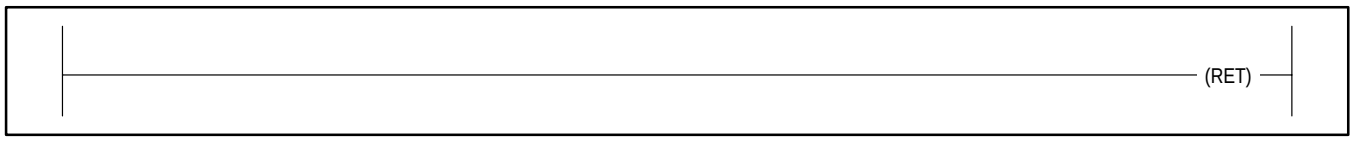
Description: The return instruction concludes the subroutine and tells the processor to resume executing at the rung following the corresponding jump-to-subroutine instruction.

Every subroutine must contain an executable return instruction. The rung containing a return instruction can be conditional. By using this method, you can program the processor to execute only a part of the subroutine if certain conditions are true. However, if you use this method, make sure that you program another return instruction at the end of the subroutine to exit when the conditions on the first return instruction are false.

CAUTION: The jump-to-subroutine instruction requires a return instruction in the subroutine. If you omit the return instruction, the processor executes the entire subroutine, performs housekeeping tasks, and begins executing from rung 0 in the main program section. It does not execute the balance of the program following the jump-to-subroutine instruction. This could cause improper operation.

Example: Figure 13.5 shows a rung containing a return instruction.

Figure 13.5
Example Rung for a Return Instruction



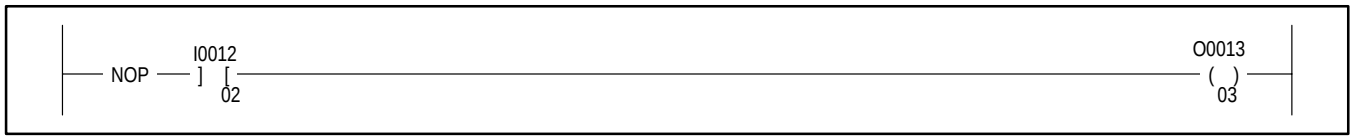
13.1.6 No Operation (NOP)

Required Parameters: None

Description: You can program a non-operation instruction on a rung where you'd like to do additional editing. When you're ready to edit the rungs you marked with the no operation instruction, you can use search editing functions through the program loader. You can then replace the no-operation instruction with another instruction.

Example: Figure 13.6 shows a rung containing a no-operation instruction.

Figure 13.6
Example Rung for a No-operation Instruction



13.1.7 End (END)

Required Parameters: None

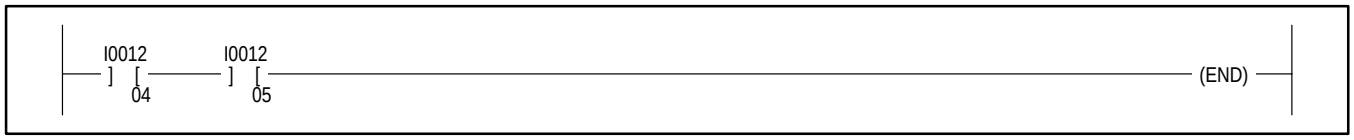
Description: Although the processor automatically generates an end-of-program symbol, you can enter a “temporary” end instruction. When the processor encounters this end instruction, it resets the watchdog timer (to zero), and begins executing the ladder program at the first instruction in the main program section.

The end instruction is an output instruction. You can program an end instruction on a rung using input instructions to control the true/false state of the rung.

Important: Do not confuse the end instruction with the end-of-program symbol. You cannot program instructions on the rung with the end-of-program symbol (EOP).

Example: Figure 13.7 shows a rung containing an end instruction.

Figure 13.7
Example Rung for an End Instruction



13.2 Recovering from Major Faults

When the processor detects a major fault, the processor shuts down by resetting the outputs and forcing the mode of operation to program load. The processor provides two methods for recovering from major faults:

- fault routine
- clear fault command

The fault routine method can prevent a shut down or execute a routine before the shut down occurs. You can use the clear fault command method after a major fault has occurred.

13.2.1 Using a Fault Routine

You can enter a label with label number 0 as the first instruction in a fault routine to have the processor execute a fault routine when it detects a major fault. The fault routine must end with a return instruction.

When the processor detects a major fault during program execution, it looks for label 0 in the fault routine section. If label 0 does not exist, the processor shuts down.

If label 0 exists, the processor:

1. copies the major fault word in system status into the major fault word in the status section (13) of the data table (S0:1).
2. executes the fault routine if the following conditions are true:
   - The fault routine is not already executing.
   - The ladder program is executing.
   - The processor is not synchronously frozen.
   - All the necessary data table and ladder program sections exist.
3. copies the status fault word to the system fault word. Then, if the system fault word is zero, it resumes normal operations. Otherwise, it shuts down.

In programming note that fault routines:

- must begin with a label with label number 0
- can consist of any valid instruction
- cannot call a subroutine
- must be short enough to avoid causing the program scan to exceed the watchdog timeout value

Figure 13.8 shows an example fault routine. If a major fault occurs, the processor looks for label 0 in the fault routine section of the ladder program. In this example fault routine:

Rung number	Tells the processor to
RF0	examine the memory-parity-error status bit and execute the procedure @FAULT on peripheral-communication-module number one if the bit is set
RF1	examine the watchdog timeout status bit and execute the procedure @ROUTINE on peripheral communication module number one if the bit is set

The procedures in the message instructions send information to an RS-232-C device connected to peripheral communication module number one that a memory parity error has occurred or the watchdog timer has timed out. Both of these faults are major faults. So after the processor executes the message instruction, it shuts down. You can create fault routines that correct the fault, in which case, the processor resumes normal operation after executing the fault routine.

Figure 13.9 shows you GA Basic procedures @ FAULT and @ ROUTINE for the example fault routine. For detailed information on using GA Basic, refer to the Peripheral Communication Module (cat. no. 1775-GA) User's Manual (publication 1775-6.5.4).

Figure 13.8
Example Rungs for a Fault Routine

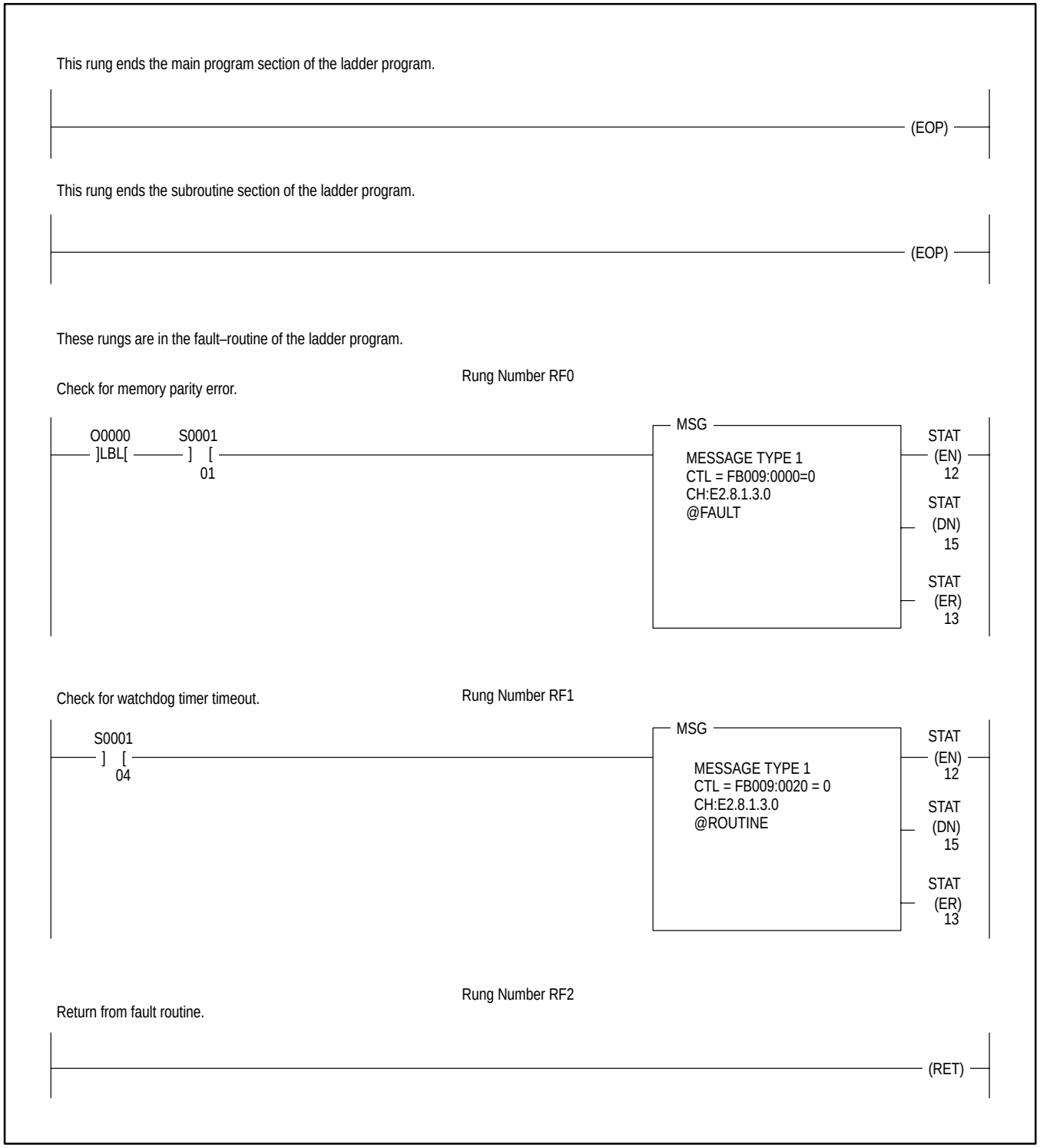


Figure 13.9
GA Basic Procedures for Example Fault Routine

```
@FAULT

PRINT 'MEMORY PARITY OCCURRED AT THE FOLLOWING TIME'
PRINT
PRINT 'HOURS: '$S1:3 'MINUTES: '$S1:4
PRINT ' THE STAT OF THE OUTPUT BITS UPON MAJOR FAULT'
I = 0
OUTPUT:
PRINT '013/' I! + 20' WAS 1 (0:13/I)!10
I = I + 1
IF (I. LE. 15) GO OUTPUT
```

```
@FAULT EXECUTION

GA!>@FAULT
MEMORY PARITY OCCURRED AT THE FOLLOWING TIME:

HOURS: 19 MINUTES: 57
THE STATE OF THE OUTPUT BITS UPON MAJOR FAULT
013/00 WAS 0
013/01 WAS 1
013/02 WAS 1
013/03 WAS 0
013/04 WAS 1
013/05 WAS 1
013/06 WAS 1
013/07 WAS 1
013/10 WAS 1
013/11 WAS 1
013/12 WAS 0
013/13 WAS 0
013/14 WAS 0
013/15 WAS 1
013/16 WAS 1
013/17 WAS 1
```

```
@SUBROUTINE

PRINT 'WATCHDOG ERROR! PROGRAM SCAN EXCEEDED MAX. ALLOWABLE TIME.'
PRINT 'CHECK SYSTEM STATUS SELECTION IN LIST (SELECTION 5).'
PRINT 'THEN WATCHDOG SET UP (SELECTION 2).'
PRINT
PRINT 'FACTORS INVOLVED IN DETERMINING MAX. TIME ALLOWED ARE:'
PRINT '1. MAIN PROGRAM SCAN.'
PRINT '2. SUBROUTINE SCAN (INCLUDE FREQUENCY PER SCAN)'
PRINT '3. FAULT ROUTINE IF CALLED UPON'
PRINT '4. REAL TIME INTERRUPT SCAN (INCLUDE FREQUENCY PER SCAN)'
PRINT '5. ON–LINE EDITING'
PRINT '6. ADJUSTING WATCHDOG REFERENCE IN LIST'
```

```
@ ROUTINE EXECUTION

GA1>@ROUTINE
WATCHDOG ERROR! PROGRAM SCAN EXCEEDED MAX. ALLOWABLE TIME.
CHECK SYSTEM STATUS SELECTION IN LIST (SELECTION 5).
THEN WATCHDOG SET UP (SELECTION 2).

FACTORS INVOLVED IN SETERMINING MAX. TIME ALOWED ARE:
1. MAIN PROGRAM SCAN
2. SUBROUTINE SCAN (INCLUDE FREQUENCY PER SCAN)
3. FAULT ROUTINE IF CALLED UPON
4. REAL TIME INTERRUPT SCAN (INCLUDE FREQUENCY PER SCAN)
5. ON–LINE EDITING
6. ADJUSTING WATCHDOG REFERENCE IN LIST
```

13.2.2 Using the Clear Fault Command

You can also recover from major faults by using the clear fault command. This command resets the least significant set bit in the major fault word in system status. To use this command, refer to the user's manual for your program loader.

13.3 Real-time Interrupt

The real-time interrupt feature of the controller allows for high speed updating of critical data table information. To program a real-time interrupt routine, you enter a label with label number 1 as the first instruction for a subroutine. Between this label and return instruction, you can program rungs to update the critical data table locations. Then, the processor executes this real-time-interrupt routine at time intervals that you set through the LIST function in milliseconds (0 - 65,535). When the processor executes the return instruction for the subroutine, it continues executing the main program section.

In programming, note that real-time-interrupt routines:

- must be programmed in the subroutine area
- must begin with a label with label number 1
- should conclude with a return instruction to resume execution of the main program
- can consist of any valid instruction
- cannot call a subroutine
- do not execute when the processor is executing a fault routine
- must be short enough to avoid causing the program scan to exceed the watchdog timeout value
- must have an interrupt interval that is long enough so that the subroutine is completed before the next interrupt.

CAUTION: Make sure that the real-time interrupt interval does not exceed the time interval that you specified in LIST. Otherwise, the processor declares a minor fault and sets bit 16 in status file 0, word 0.

13.3.1 Calculating the Interrupt Interval

To determine the minimum value to set the interrupt interval at, add the maximum time that the processor takes to execute the real-time-interrupt routine to the housekeeping time.

When performing housekeeping, the processor can be in one of the following modes:

In this mode	The processor performs these tasks	In less than
normal housekeeping	updates system status, module status, and the status section of the data table	3ms
check summing	computes the checksum of the ladder program and normal housekeeping if data table word S0:5 exists	4ms
gapping, testing edits, and assembling edits	performs each operation as requested through on-line programming or report generation and normal housekeeping	15ms for each operation

Important: If an on-line edit requires the data table to change size or a new section of memory to be created, then the time given for gapping above is incorrect. To calculate the time for gapping, use the following formula:

$$15 + ((A - 500) \times 0.005) = \text{minimum time in milliseconds}$$

where:

- 5 = constant for on-line programming and assembly
- A = size of largest section in data table for the active context

You can use the memory map command through your program loader to determine the size of the largest section in the data table. Refer to the user's manual for your program loader for detailed information.

13.4 Switching Contexts

The controller can store multiple ladder programs and execute any one of them at a given time. The controller features the context function to accomplish this. A context consists of the ladder program and the following information:

- data table
- messages
- symbols
- forces

With the context function, the processor can store up to 15 separate ladder programs. Contexts are numbered 0 to 15, with 1 to 15 storing separate ladder programs. The processor uses context 0 to store global data used by all contexts. You select the ladder program that you want to execute by selecting the corresponding context number through the LIST function. You must be in program-load mode to switch from one context to another and you can run a program in one context while editing a program in another. For detailed information on the context selection, refer to the PLC-3 Controller Installation and Operation Manual (publication 1775-6.7.1).

Chapter 14

Addressing Memory and Monitoring Controller Status

14.0 Chapter Objectives

In chapter 3, we described the data table addressing method for addressing words and bits in the data table sections. In this chapter, we describe methods for addressing the other memory areas. After reading this chapter, you should:

- be able to use extended addressing to address any user-accessible area in memory
- understand how the status bits are organized within the status section of the data table
- be able to monitor status bits in your ladder program

14.1 Using Extended Addressing

Extended addressing lets you access any user-accessible area in memory. In programming a processor, you can use extended addressing to:

- specify that a memory communication, report generation, data highway, or GA Basic command execute from a message instruction in the ladder program (refer to chapter 16)
- force an input or output on or off through the data monitor
- copy data from an area into a data table location using the move-status instruction (refer to chapter 6)

The format for an extended address is given below:

E<level 1>.<level 2>.<level 3>. ... .<level n>

Level 1 is the value that corresponds to the area that you want to access:

0 = system status
2 = module status
3 = data table
4 = ladder program
5 = message
6 = system symbols

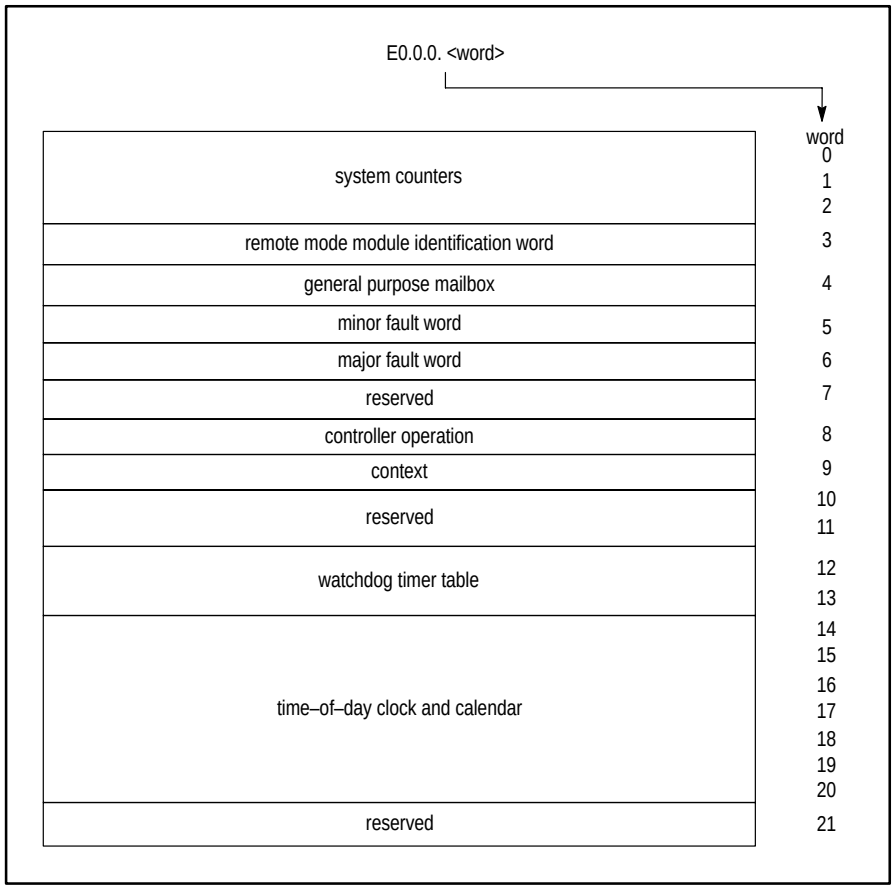
8 = converted procedures
10 = force table

Level 2 to n represent the parameters that you need specify for the area you are accessing. Table 14.A summarizes the levels for the memory areas. We describe in detail the organization for each area in the following sections.

14.1.1 System Status

To address the system status area, enter (Figure 14.1):

Figure 14.1
Word Organization for the System Status Area



For example:

This address	Corresponds to
E0.0.0.9	the system status area (E0), context (0), section (0), context word (9)

Table 14.A
PLC-3 Extended Addressing Table

Area	E#	.x	.x	.x	.x	.x
system status	E0	context=0	section=0	word=0-20	not used	not used
module status	E2	module type= 1–memory 2–main processor 3–S4A I/O scanner 5–communication adapter 6–expansion 7–S4B I/O scanner 8–peripheral communication 9–data highway II interface 14–memory communication	thumbwheel switch=1-15	module data	module data	module data
data table	E3	context=1-15	section= 1–output image 2–input image 3–timers 4–counters 5–integers 6–floating point 7–decimal 8–binary 9–ASCII 10–high order int. 12–pointers 13–status	file= 0-999 0-999 0 0 0-999 0-999 0-999 0-999 0-999 0-999 0 0-999	structure= 0 0 0-9999 0-9999 0 0-9999 0 0 0 0-9999 0-9999 0	word= $0\text{-}7777_8$ $0\text{-}7777_8$ 0-CTL, 1-PRE, 2-ACC 0-CTL, 1-PRE, 2-ACC 0-9999 0-9999 0-9999 0-9999 0-9999 0-9999 0-SEC, 1-FIL, 2-WRD 0-9999
ladder program	E4	context=1-15	section= 0–program status 1–main 2–subroutine 3–fault routine	rung= 0-32,767	instruction= 0-32,767	word= 0-32,767
message	E5	context=1-15	section= 1–report generation 2–rung comments 3–terminal (MACROS) 4–data highway 5–assistance (HELP)	message= 0-32,767	word= 0-32,767	not used
system symbols	E6	context=1-15	type=1	symbol= 0-32,767	word= 0-32,767	not used
converted procedures	E8	context=1-15	section= 1–report generation	message= 0-32,767	word= 0-32,767	not used
force table	E10	context=1-15	force type= 0–status 1–forced output 2–forced input	rack= not used 0-64 0-64	word=0 0-15 0-15	bit= 0–input force enabled/disabled 1–output forces enabled/disabled not used not used

14.1.2 Module Status

To address the module status area, enter:

E2.<module type>.<thumbwheel>.<module data>.<module data>...

Module type specifies the module that you want to access:

1 = memory
2 = main processor
3 = I/O scanner-programmer
5 = communication adapter
6 = expansion
7 = I/O scanner message-handling
8 = peripheral communication
9 = data highway
14 = memory communication

Thumbwheel specifies the thumbwheel setting on the module and can be a value from 1 to 15.

Module data specifies parameters for the module that you are accessing. Refer to the user's manual for the module for detailed information.

For example:

This address	Corresponds to
E2.5.1	the module status area (E2), communication adapter module (5), thumbwheel setting (1)

14.1.3 Data Table

To address the data table area, you can use the data table addressing method described in chapter 3. If you want to use extended addressing, enter:

E3.<context>.<section>.<file>.<structure>.<word>

Context specifies the context containing the data table that you want to access and can be a value from 1 to 15.

Section specifies the section that you want to access:

1 = output image
2 = input image
3 = timers
4 = counters
5 = integers
6 = floating point
7 = decimal
8 = binary

9 = ASCII
10 = high order integer
12 = pointers
13 = status

File specifies the file number for the data table section and can be a value from 0 to 999.

Important: The file level dos not apply for the timer, counter, and pointer sections. If you are specifying one of these sections, enter 0 for the file.

Structure specifies the timer, counter, or pointer number and can be a value from 0 to 9999.

Important: The structure level does not apply for the output image, input image, integer, decimal, binary, ASCII, and status sections. if you are specifying one of these sections, enter 0 for the structure.

Word specifies the word number for the data table section:

Section	Acceptable Word Numbers
output or imput image	$0\text{-}7777_8$
timers or counters	0 for the control word 1 for the preset word 2 for the accumulated value word
integer, floating point, decimal, binary, ASCII, high order integer, status	0-9999
pointers	0 for the section 1 for the file 2 for the word

Refer to section 14.2 for detailed information on status file organization.

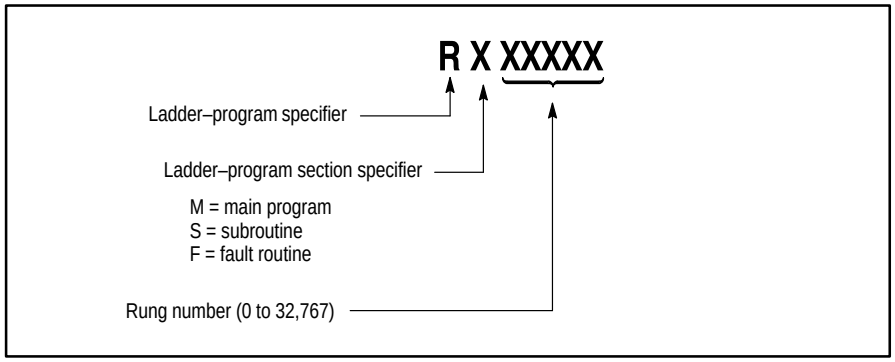
For example:

This address	Corresponds to
E3.1.2.3.0.0	the data table area (E3), context (1), input image table (2), file (3), word (0)
E3.1.8.1.0.5	binary file 1, word 5
E3.3.3.10.0	the control word (0) for timer 10 in context (3)

14.1.4 Ladder Program

To address the ladder program area you can use the ladder-program addressing method to access individual rungs (Figure 14.2).

Figure 14.2
Ladder-program Addressing Method



If you want to access the individual instructions on a rung, you can use extended addressing by entering:

E4.<context>.<section>.<rung>.<instruction>.<word>

Context specifies the context containing the ladder program that you want to access and can be a value from 1 to 15.

Section specifies the ladder program section that you want to access:

0 = program status (Figure 14.3)
1 = main
2 = subroutine
3 = fault routine

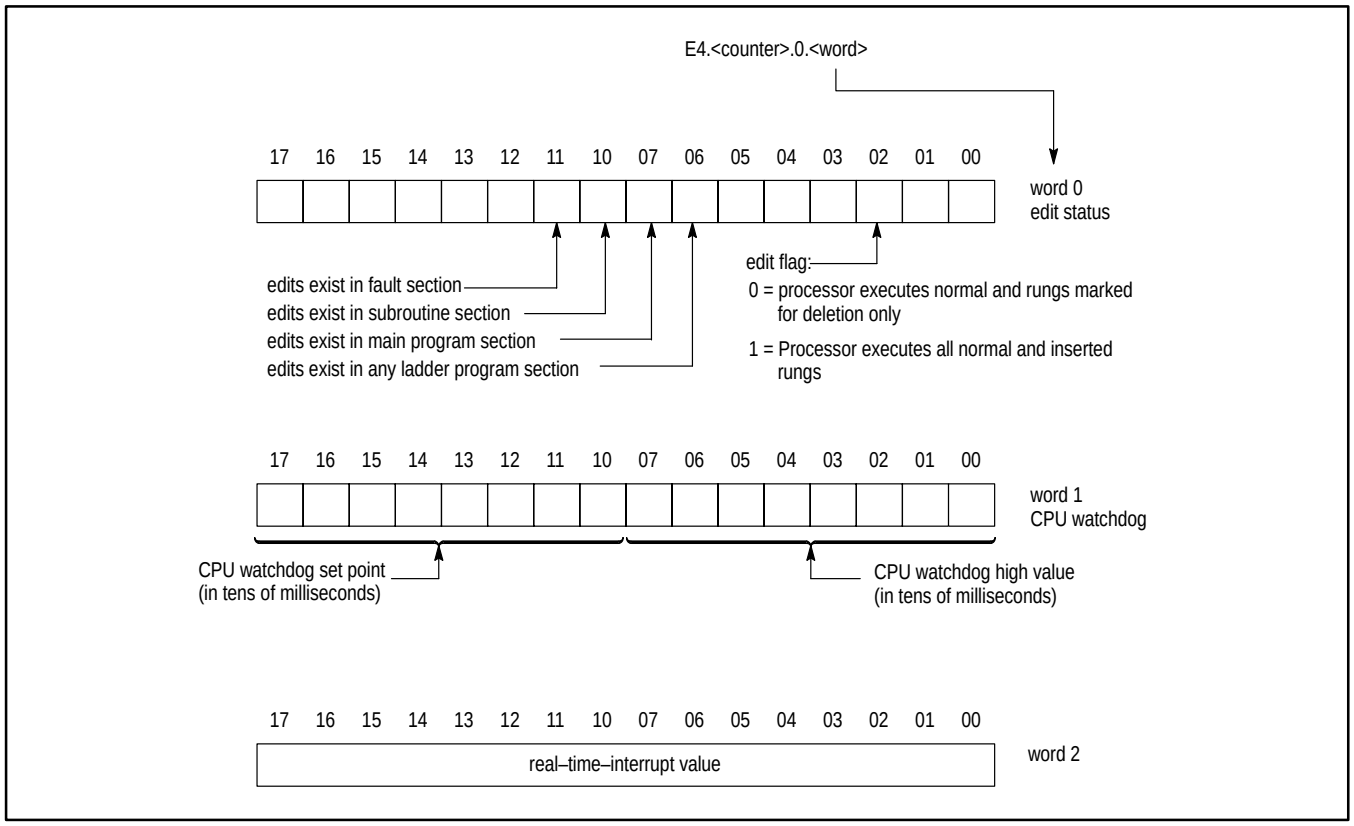
Rung specifies the rung number that you want to access and can be a value from 0 to 32,767.

Instruction specifies the instruction number that you want to access and can be a value from 0 to 32,767.

For example:

This address	Corresponds to
RM5	the main program section (M), rung number (5)
E4.1.1.5.1.0	the ladder program area (E4), context (1), main program section (1), rung (5), the first instruction on the rung (1)

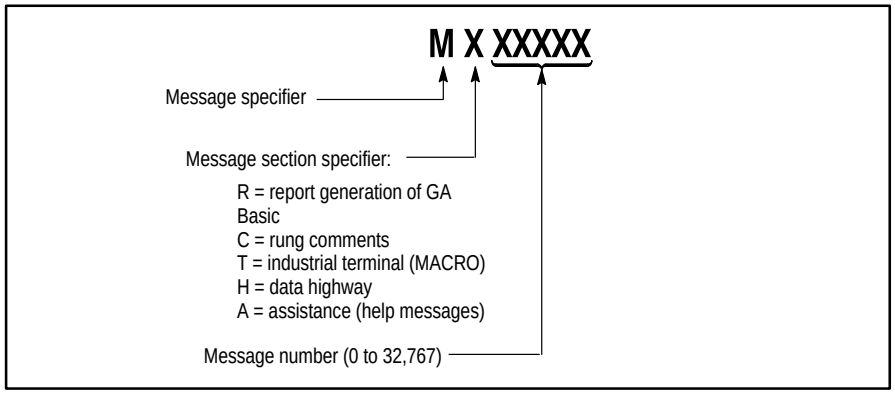
Figure 14.3
Word Organization for the Ladder-program Status Section



14.1.5 Message

To address the message area you can use the message addressing method to access individual messages (Figure 14.4).

Figure 14.4
Message Addressing Method



If you want to access the individual words that make up the message, you can use extended addressing by entering:

E5.<context>.<section>.<message>.<word>

Context specifies the context containing the message that you want to access and can be a value from 1 to 15.

Section specifies the section within the message area that you want to access:

1 = report generation or GA basic
2 = rung comments
3 = terminal (MACROS)
4 = data highway
5 = assistance (HELP)

Message specifies the message number and can be a value from 0 to 32,767.

Word specifies the word number for the message:

Word	Stores
0	first two letters in the message
1	second two letters in the message
.	
.	
.	
n	(n+1)th two letters in the message up to 32, 767

For example:

This address	Corresponds to
MH10	the data highway section (H), message number (10)
E5.1.4.10.1	the message area (E5), context (1), data highway section (4), message number (10), second two letters in the message (1)

14.1.6 System Symbols

To address the system symbols area, enter:

E6.<context>.1.<symbol>.<word>

Context specifies the context that the symbol was created in and can be a value from 1 to 15.

Symbol specifies the value for the specific symbol and can be a value from 0 to 32,767.

Word specifies the word number for the symbol:

Word	Stores
0	first two letters in the symbol name
1	second two letters in the symbol name
2	third two letters in the symbol name
3	fourth two letters in the symbol name
4	symbol type: 1 = symbolic address 2 = message stored in section 5
5	number of levels in starting address
6	first three levels in starting address
7	fourth level in the starting address
8	fifth level in the starting address
. . .	
n	(n–3)th level in the starting address

For example:

This address	Corresponds to
E6.1.1.10.4	the system symbols area (E6), context (1), symbol type (1), symbol number (10), symbol type word (4)

14.1.7 Converted Procedures

The converted procedure area is reserved for processor operation with the Peripheral Communication Module (cat. no. 1775-GA). You cannot access it when programming. For detailed information on this area, refer to the user's manual (publication 1775-6.5.4).

To address the converted procedure area, enter:

E8.<context>.<section>.<procedure>.<word>

Context specifies the context containing the procedure that you want to access and can be a value from 1 to 15.

Section specifies the section within the converted procedure area that you want to access. Enter 1 for report generation.

Procedure specifies the value for the specific procedure and can be from 0 to 32,767.

Word specifies the word number for the procedure:

Word	Stores
0	first two letters in the procedure
1	second two letters in the procedure
.	
.	
.	
n	(n+1)th two letters in the procedure up to 32, 767

For example:

This address	Corresponds to
E8.1.1.1.1	the converted procedure area (E8), context (1), report generation section (1), converted procedure (1), word that stores the first two letters in the procedure (1)

14.1.8 Force Tables

To address the force tables area, enter:

E10.<context>.<force type>.<rack>.<word>

Context specifies the context containing the input(s) and/or output(s) that you want to force.

Force type specifies the section of the force table that you want to access:

0 = status
1 = forced output
2 = forced input

Rack specifies the assigned I/O rack number that contains the input(s) and/or output(s) that you want to force and can be a value from 0 to 64 in decimal.

Important: The rack number does not apply for status force type (0). The extended address is E10.<context>.0.0.

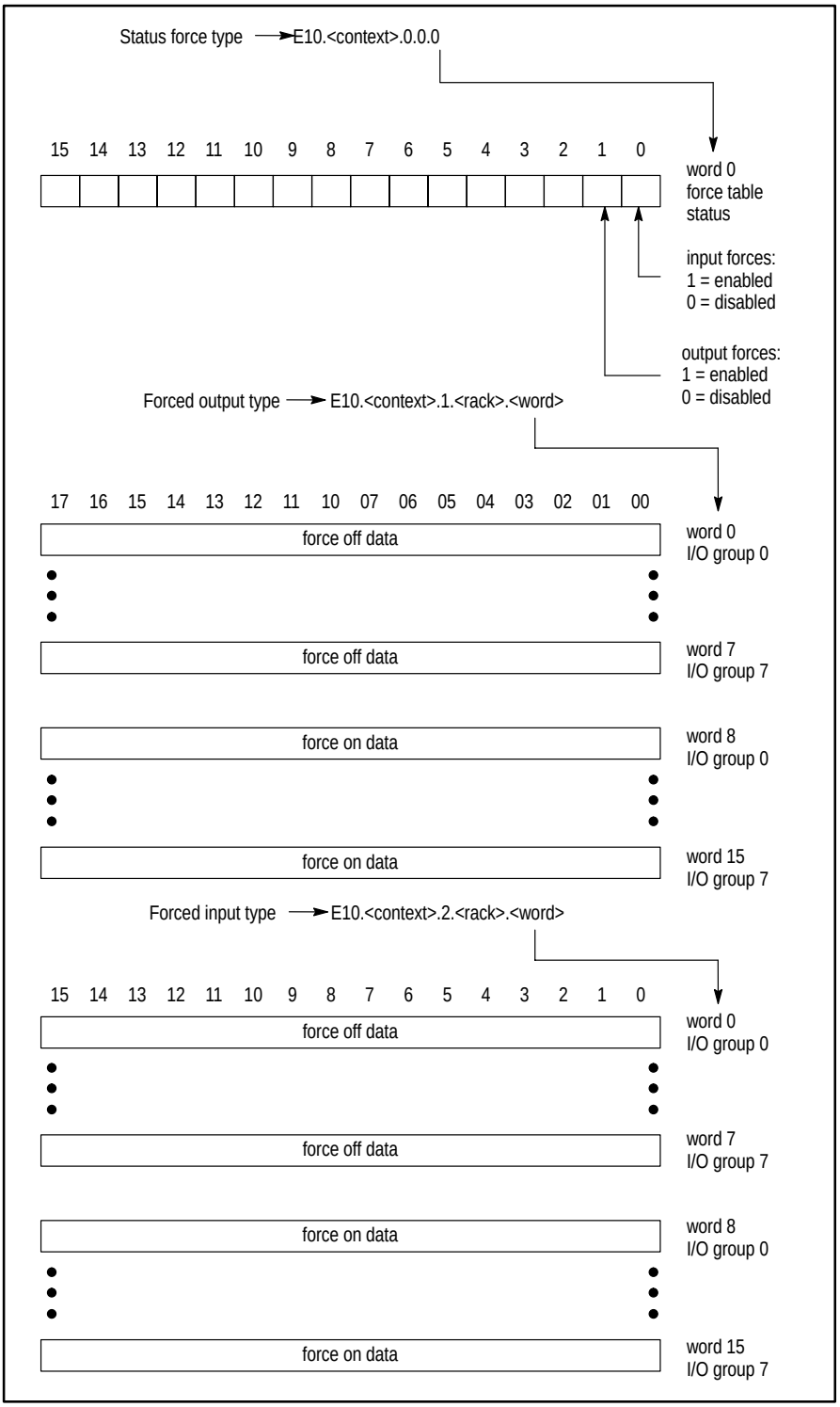
Word specifies the word number for the force type and can be a value from 0 to 15 in decimal (Figure 14.5).

Important: If you are addressing the status force type, enter 0 for the word.

For example:

This address	Corresponds to
E10.1.1.2.12/6	force table area (E10), context (1), forced output type (1), assigned I/O rack (2), forced on data for I/O group (12), bit (6)

Figure 14.5
Word Organization for the Force Table Area



14.2 Using the Data Table Status Files

The data table status section (section 13) is accessible to your ladder program. Prior to each program scan, the processor copies status information from the status area into files in the status section. Figure 14.6 shows the organization of the data table status files. By monitoring the status bits in your ladder program, you can keep track and diagnose problems. We recommend that you do not use the data table status section for word or bit storage.

14.2.1 Fault, Operating Mode, and Program Checksum Status (Status File 0)

Status file 0, word 0 through 5 store the following information:

Arithmetic operation status (S0:0) – The processor sets these bits to indicate the status of an executing arithmetic instruction. The processor resets these bits before the next arithmetic instruction executes (Figure 14.7).

Major faults (S0:1) – If a fault routine exists, the processor copies the major fault bits from the status area (Figure 14.8).

Minor faults (S0:2) – The processor clears the minor fault bits from the system status area after copying them into status file, word 2 (Figure 14.9).

Controller operating mode (S0:3) – The processor sets these bits to indicate the current operating mode of the controller (Figure 14.10).

Figure 14.6
Data Table Status Section Organization

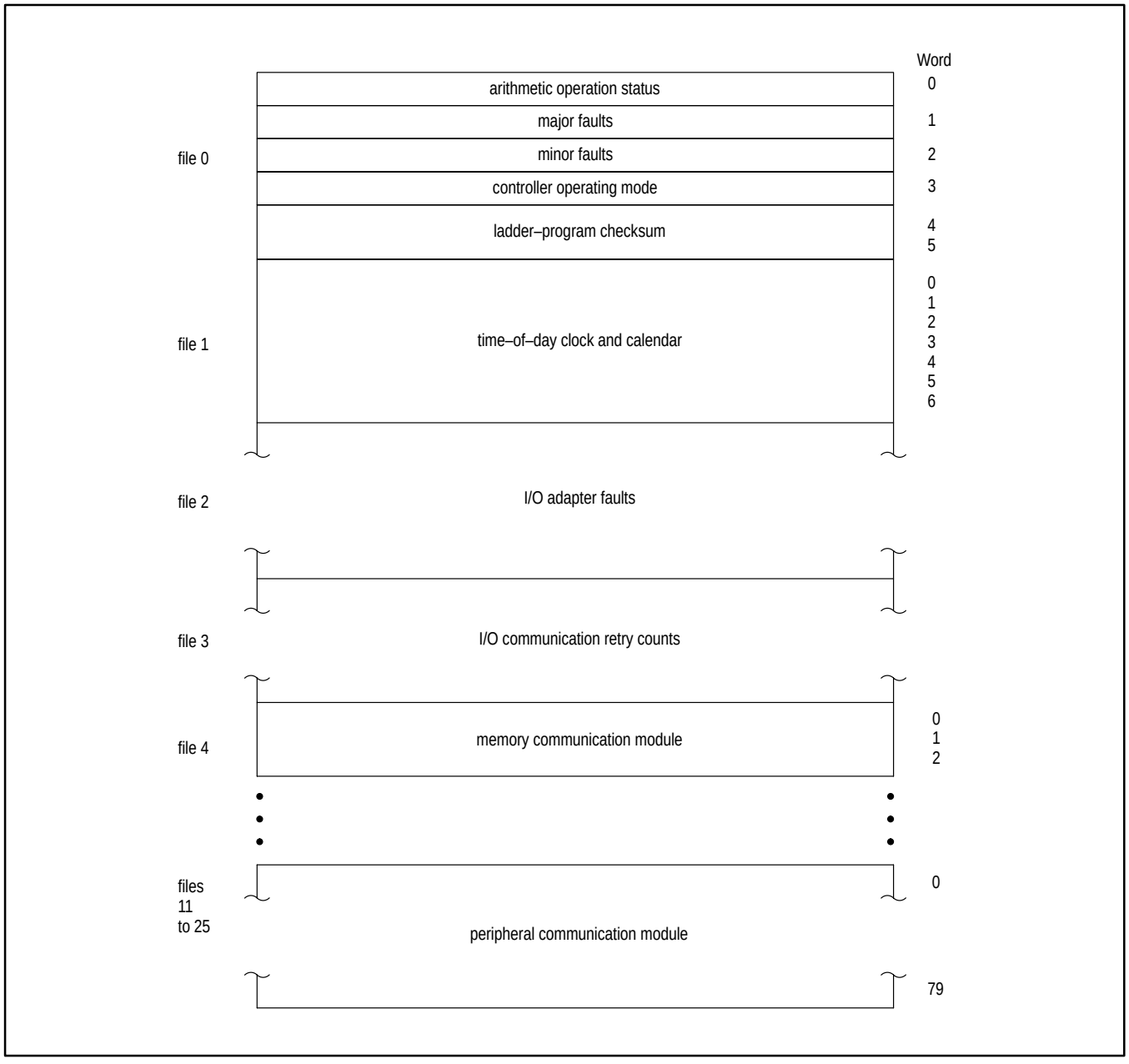
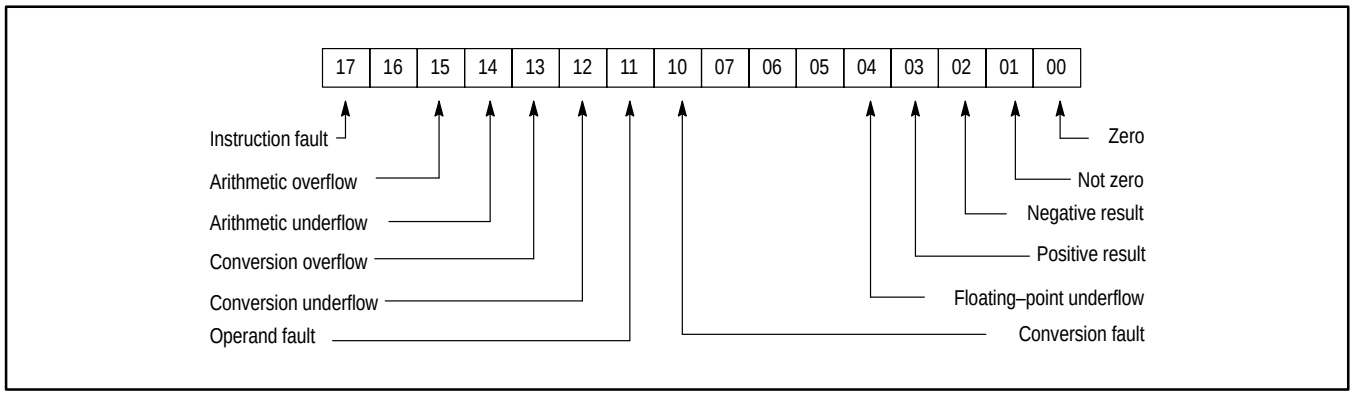
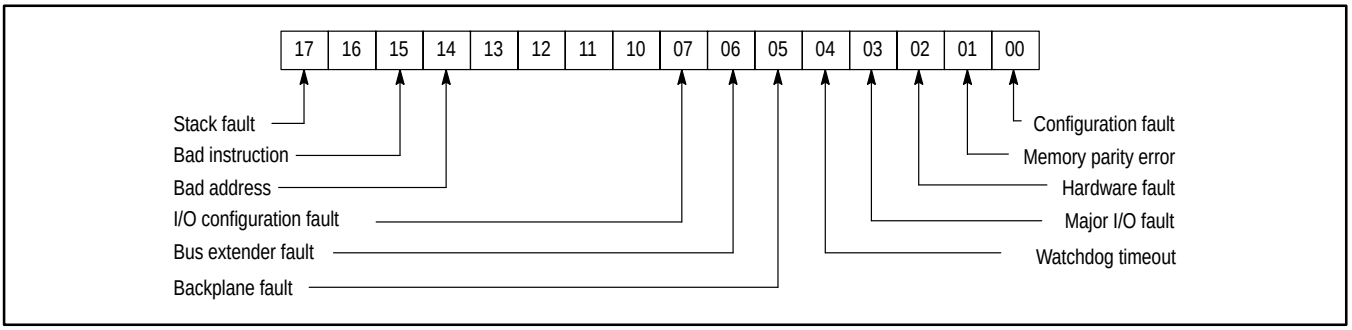


Figure 14.7
Arithmetic and Data Conversion Status Bits (Status File 0, Word 0)



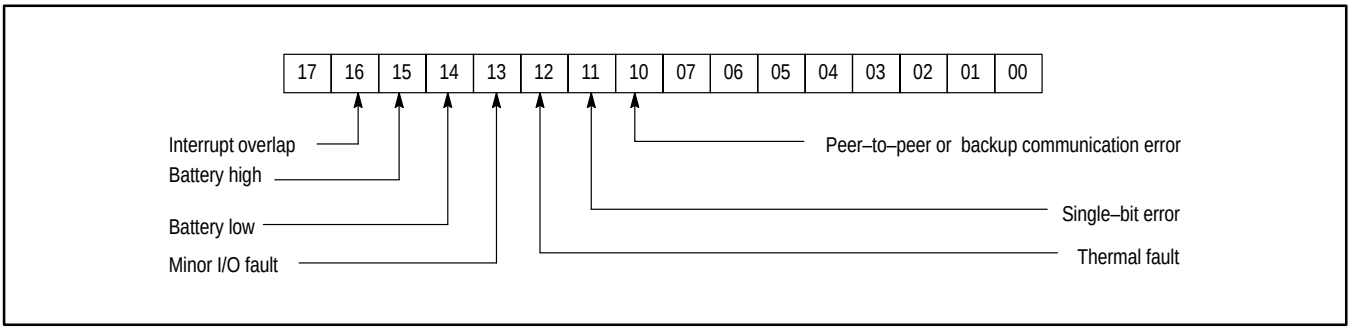
Bit	Title	The processor sets this bit when
00	zero	an instruction result is zero
01	not zero	an instruction result is not zero
02	negative result	an instruction result is negative
03	positive result	an instruction result is positive
04	floating point underflow	the absolute value of a floating-point value is too small to store at the specified address (zero can be stored)
10	conversion fault	a correct value cannot be stored in the specified section (e.g. a negative value in the decimal section)
11	operand fault	dividing a value by zero or taking the square root of a negative value
12	conversion underflow	a negative number is too large to store at the specified address
13	conversion overflow	a positive number is too large to store at the specified address
14	arithmetic underflow	an arithmetic result is a negative number that is too large to process
15	arithmetic overflow	an arithmetic result is a positive number that is too large to process
17	instruction fault	either bit 4, 10, 11, 12, 13, 14, or 15 is set

Figure 14.8
Major Fault Status Bits (Status File 0, Word 1)



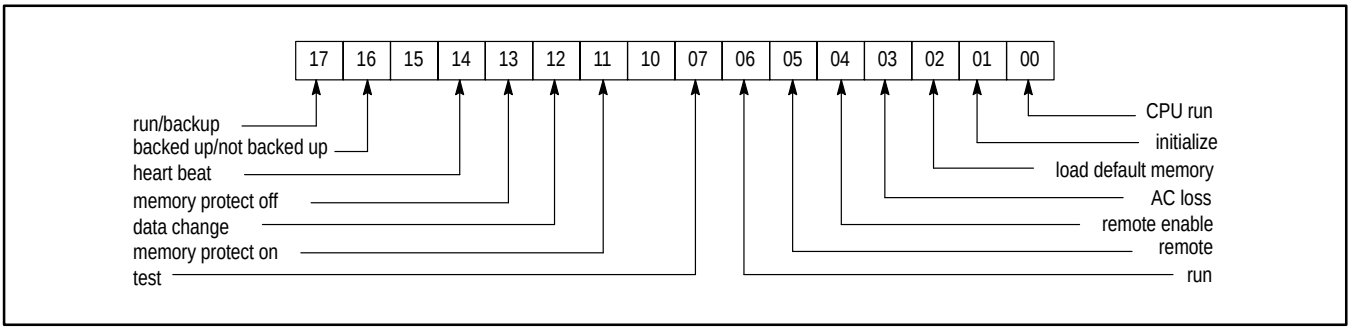
Bit	Title	The processor sets this bit when
00	configuration fault	a memory structure or module configuration problem exists
01	memory parity error	any module detects a memory parity or double-bit error when accessing a memory module
02	hardware fault	a module detects a problem in its own or another module's circuitry
03	major I/O fault	communication disrupts between a scanner and I/O adapter and a communication break is specified as a major fault through LIST
04	watchdog timeout	the actual program scan exceeds the watchdog timeout preset value specified through LIST
05	backplane fault	any module detects an error during transfer of data on the backplane
06	bus extender fault	a problem exists in the bus extender circuit in an expansion module
07	I/O config. fault	invalid data in the module status prevents I/O configuration
14	bad address	an address or a label reference does not exist or specified section, file, word, or label has not been created in memory
15	bad instruction	an undefined opcode is encountered (e.g. using a 1775-L1 processor to execute a ladder program containing instructions that can only execute on a 1775-L2, -L3 processor)
17	stack fault	the ladder program contains a return instruction with no corresponding jump-to-subroutine instruction or subroutines are imbedded more than 32 levels deep

Figure 14.9
Minor Fault Status Bits (Status File 0, Word 2)



Bit	Title	The processor sets this bit when
10	peer-to-peer or backup communication error	the primary or backup processor detects a backup communicaiton error, the master processor detects a peer-to-peer communication error, or the data transmitted exceeds the size of the specified input file
11	single-bit error	an error correcting (EDC) memory module detects single-bit errors
12	thermal fault	a thermistor on the processor detects incoming ambient air temperature higher than 60°C or 140°F
13	minor I/O fault	communication disrupts between a scanner and an I/O adapter
14	battery low	a memory module contains a battery that lacks sufficient charge
15	battery high	a memory module containing a lithium battery has an overvoltage condition. If you are using a nicad battery, do not monitor this bit.
16	interrupt overlap	the real-time interrupt routine exceeds the set point specified through LIST

Figure 14.10
Controller Operating Mode Status Bits (Status File 0, Word 3)



Bit	Title	The processor sets this bit when
00	CPU run	the ladder program is executing until the end of program code is reached
01	initialize	the operating mode changes from program load to run or test for the first program scan
02	load default memory	loading the default memory configuration through LIST
03	AC loss	the system restarts for the first program scan. You can monitor this bit to reset output devices after a power down condition.
04	remote enable	you select remote enable through LIST from the front panel
05	remote	a remote device controls the operating mode. This bit remains set until the device releases control. Remote device control can be overridden by a local device such as the industrial terminal on front panel.
06	run	you put the controller in the run mode
07	test	you put the controller in the test mode
11	memory protect on	the mode select keyswitch is in the memory protect position. All memory areas are protected and you cannot force I/O.
12	data charge	the mode select keyswitch is in the data change position. Data can be changed in the data table and you can force I/O; all other areas of memory are protected.
13	memory protect off	the mode select keyswitch is in the memory protect off position. You can change data in any memory area and force I/O.
14	heart beat[1]	
16	backed up/not backed up	backup switch number two on 1775-S4A scanner number one is in the down position. When set (backup selected), a major fault causes the controller to deactivate. This bit is reset if backup switch number two is in the up position. When reset (no backup selected), a major fault changes the operating mode to program load.

Bit	Title	The processor sets this bit when
17	run/backup	the controller has full control over I/O, reading input data and writing output data to the I/O. This bit is reset when the controller only has control over reading input data.

[1]The processor resets the heart-beat bit (14) every time that it does housekeeping which must occur at least every 2.55 seconds. The number-one scanner checks the bit to make sure that housekeeping has occurred:

If the bit	Then the scanner
is reset	sets the bit and allows operation to continue
remains set for 3s	shuts down the controller

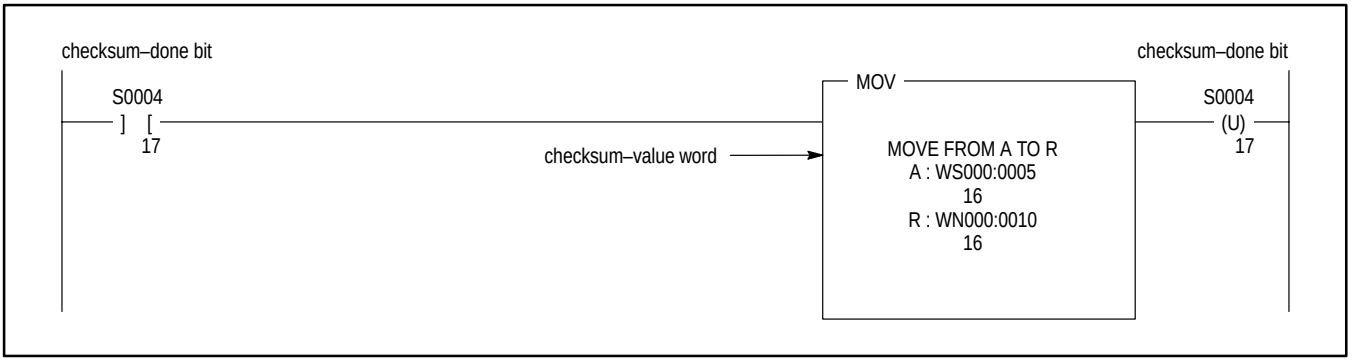
Ladder-program checksum (S0:4 and S0:5) - This checksum is a total of all words in the ladder program that you can use as an initial indication of whether the same ladder program is loaded into two processors, or whether the program in one processor has changed. However, identical checksums do not necessarily show that the programs are identical.

Important: You must create status words S0:4 and S0:5 before the processor calculates a ladder-program checksum.

Bit 17 (S0:4/17) is the done bit for the checksum calculation. Word 5 (S0:5) stores the checksum value.

Once you create these status words, the processor continuously calculates the checksum which adds 1ms to the program scan time. To monitor separate checksum calculations, you can use the rung shown in Figure 14.11.

Figure 14.11
Example Rung that Monitors Checksum Calculations

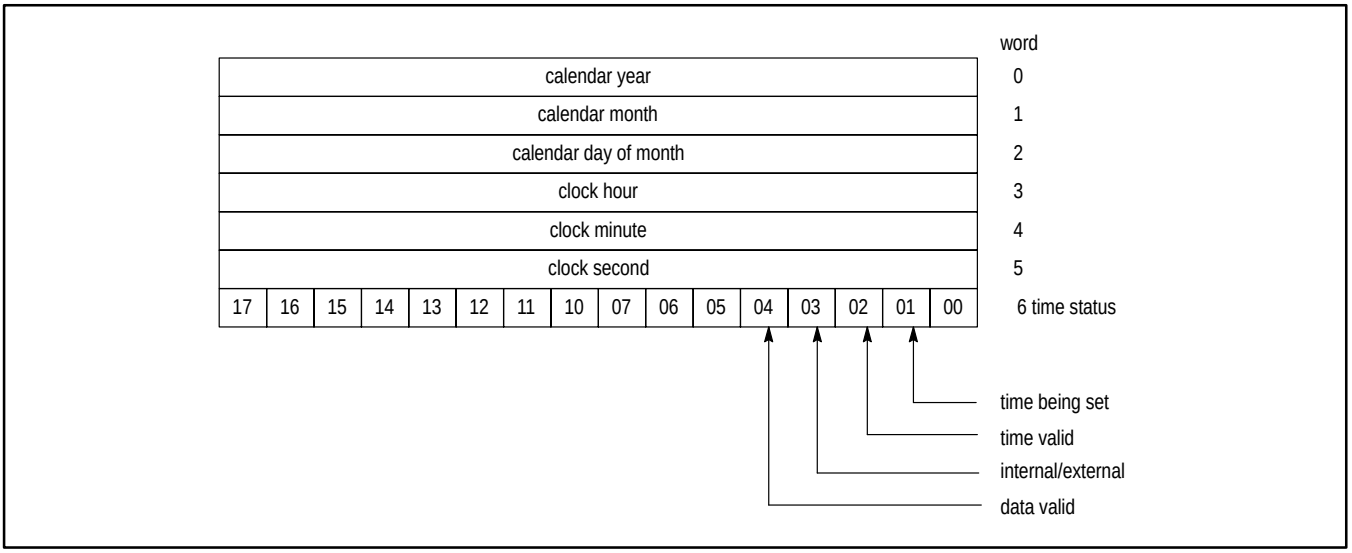


In this rung, when the checksum-done bit is set, the processor moves the checksum value into an integer word. Then it unlatches the checksum-done bit.

14.2.2 Time-of-Day Clock and Calendar (Status File 1)

Figure 14.12 shows the word organization for status file 1

Figure 14.12
Time-of-day Clock and Calendar (Status File 1)

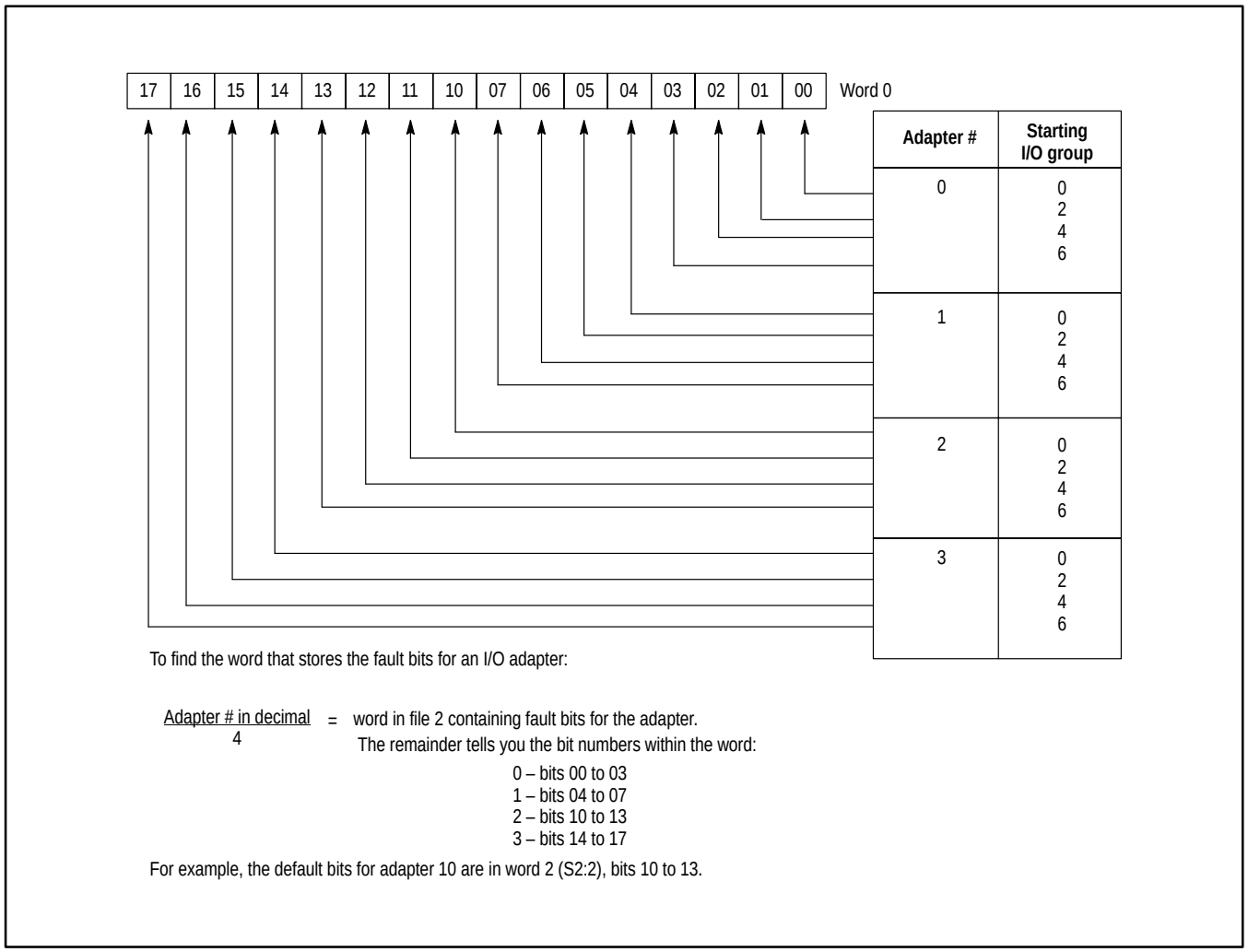


Bit	Title	The processor sets this bit when
01	time being set	the current time is being changed
02	time valid	the time has been entered through LIST
03	internal/external	a module other than the processor controls the real-time clock and calendar
04	date valid	the date has been entered through LIST

14.2.3 I/O Adapter Module Faults (Status File 2)

Figure 14.13 shows the word organization for status file 2.

Figure 14.13
I/O Adapter Fault Status Bits (Status File 2)



Important: The processor does not create status file 2 at power-up. It creates the file if the ladder program contains an instruction that references a word in status file 2. The particular word referenced determines the length of the file.

For example, if you want to monitor the adapter fault status bit corresponding to assigned I/O rack number 15, I/O group 4, you would program an examine-on or-off instruction for status file 2, word 3 bit 6 (S2:3/06).

When the processor encounters this address reference, it creates status file 2, words 0 through 3. Then, the scanner can update the file with the appropriate fault status.

Inputs that Control Outputs in Different I/O Chassis

WARNING: If an I/O fault occurs in an I/O chassis containing inputs, input image table words corresponding to the faulted chassis remain in their last prefaulted state. This action could cause an unexpected operation with possible injury to personnel.

The input image table retains the status of inputs from the faulted chassis. If an I/O fault should occur in a chassis containing inputs, outputs in an unfaulted chassis can remain on according to the last state input conditions in the ladder program.

To guard against this condition, you can monitor the I/O fault status bits to detect a fault in the chassis containing the input modules that condition the critical outputs. Along with monitoring these bits, you can use one of the following program techniques to ensure that critical outputs are properly controlled when an I/O fault occurs:

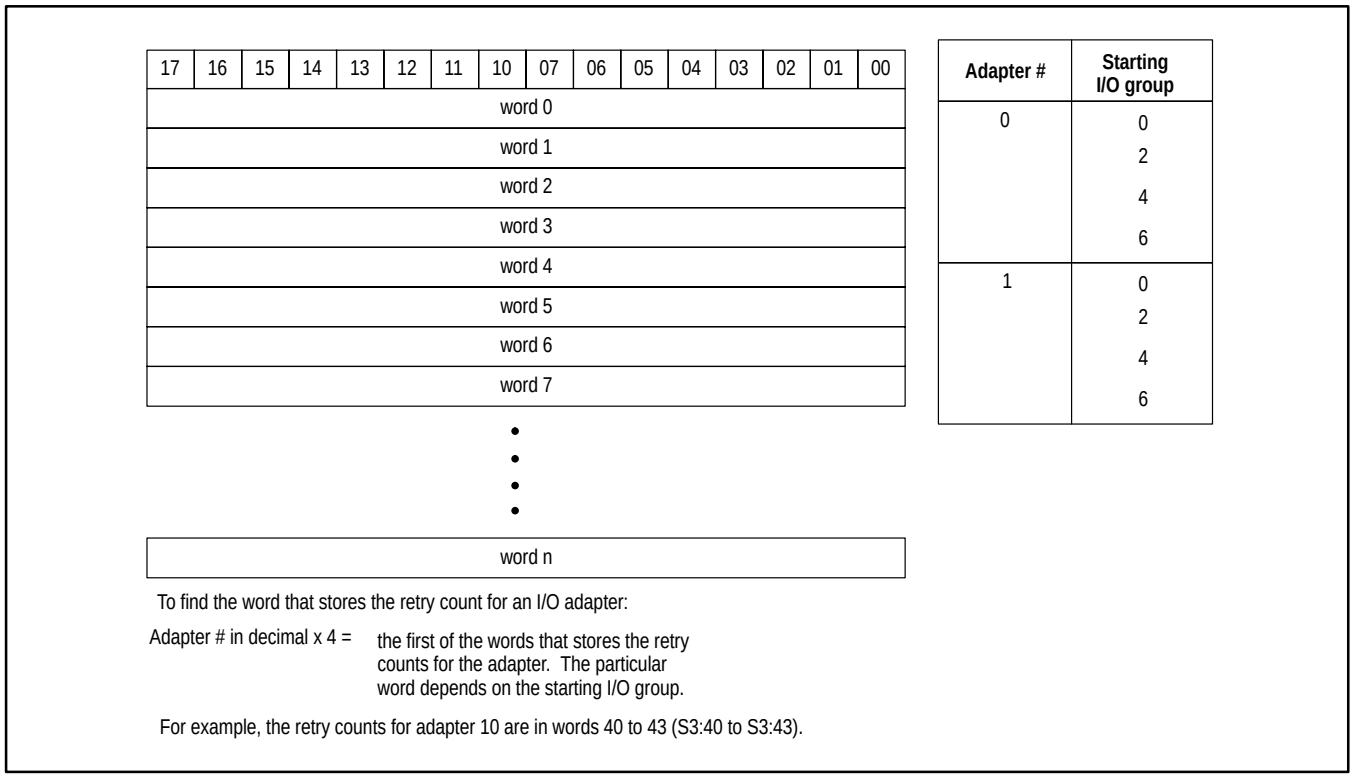
- Program critical outputs in a master-control-reset (MCR) zone so that if an I/O chassis faults, the processor disables the outputs within the zone. You must condition the outputs in the MCR zone with non-retentive program instructions.
- Program critical outputs in a subroutine so that if an I/O chassis faults, the processor jumps to the subroutine to control the critical outputs. Outputs that are jumped over in the ladder program remain in their last prefaulted state until or unless other instructions control them.

14.2.4 I/O Communication Retry Counts (Status File 3)

A communication retry is a re-transmission of data that occurs when the original transmission is unsuccessful. If the I/O adapter does not respond or sends invalid data, when the scanner communicates to the I/O adapter, the scanner executes a retry.

You can track the retry count by monitoring status file 3 (Figure 14.14)

Figure 14.14
I/O Communication Retry Counts (Status File 3)



Important: The processor does not create status file 3 at power-up. You must create it in memory by using the create command.

If you find that the retry count is high for an I/O channel, there could be a loose connection or a noise problem with the twinaxial cable (cat. no. 1770-CD) that connects from the scanner's terminal arm to the I/O adapter. Retries could also result from improper installation of I/O terminator resistors along the I/O channel. Refer to the PLC-3 Family Controller Installation an Operation Manual (publication 1775-6.7.1) for detailed information.

Upon executing a retry, if the I/O adapter responds properly, normal operation continues. If the I/O adapter does not respond properly, the scanner continues to execute retries until:

- the I/O adapter returns a valid response, or
- the processor declares a major or minor I/O fault based on how you configure the rack list in LIST.

Adapters on channel	Number of consecutive retries that execute before the processor declares an I/O fault
1	10
2	8
3 to 7	6
8 to 16	4

If the processor declares an I/O fault, it sets the following bits:

- I/O adapter fault status bit in status file 2 that corresponds to the I/O adapter
- major or minor I/O fault bit in system status

Then, the scanner continues scanning the I/O chassis. When it returns to the faulted I/O adapter, it attempts to reset the input or output and moves on the next I/O adapter.

Important: Upon declaring a major or minor I/O fault, if normal communication returns to the I/O adapter, the processor does not reset the I/O adapter fault bit in status file 2 or the major I/O or minor I/O fault bit in system status.

If you do not want normal communication to return to a faulted I/O adapter, set the processor-reset-lockout switch on the I/O chassis. With this switch set, the I/O does not reply to the scanner once a fault is declared until you cycle power at the I/O chassis or press the rest button on the I/O adapter.

Retries for a Peer-to-peer or Backup Communication Channel

If you configure an I/O channel for peer-to-peer or backup communication, and communication problems occur between two scanners (master, slave, primary, or backup) on the channel, a retry executes. Then, if the communication occurs properly, normal operation continues.

If communication problems continue to occur, the scanners:

- set the peer-to-peer or backup-communication minor fault status bit in system status
- flash the LED corresponding to the channel on their front edges

Also, the scanner configured as the master on a peer-to-peer communication channel declares a peer-to-peer communication minor fault when it cannot successfully communicate with a slave after two consecutive retries.

Important: Once set, the peer-to-peer and backup-communication minor status bit remains set until you reset it.

14.2.5 1775-MX Module (Status File 4) and 1775-GA Module (Status Files 11 to 25)

For detailed information on the status files for these modules refer to the:

- PLC-3 Family Controller Backup Concepts Manual (publication 1775-6.3.1)
- Peripheral Communication Module (Cat. No. 1775-GA) User's Manual (publication 1775-6.5.4)

Chapter 15

Executing Block Transfers

15.0 Chapter Objectives

In this chapter, we describe how to use the block-transfer feature with the controller. After reading this chapter, you should understand:

- parameters needed to execute a block-transfer instruction
- how the block-transfer control file words
- how block-transfer-read and -write instructions operate
- how to troubleshoot and prevent block-transfer errors
- how to reduce scan time when executing block transfers

15.1 Applying Block Transfers

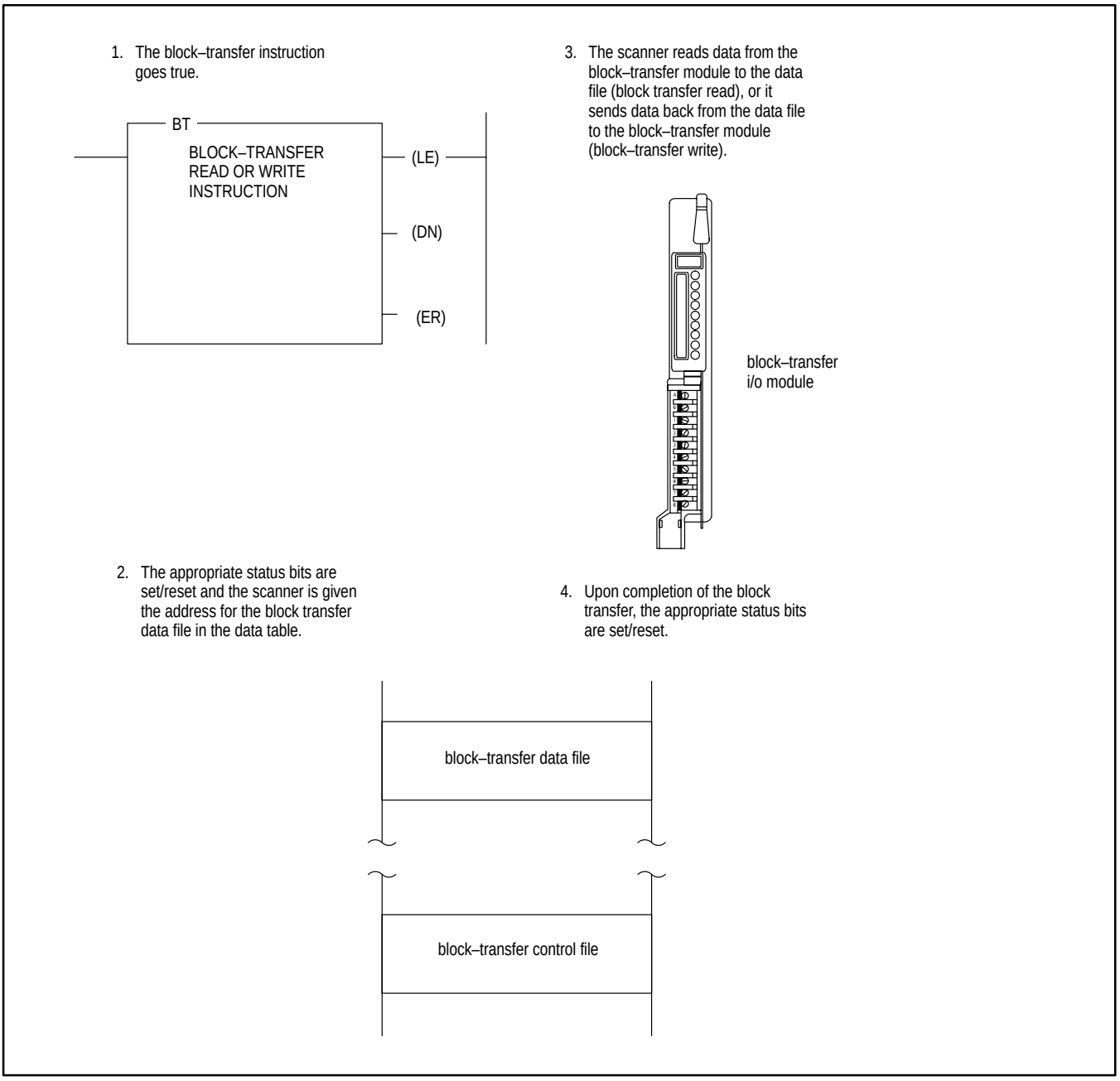
Block transfer is a program feature that allows you to transfer data to and from 1771 I/O modules. You can use block transfer with Intelligent I/O modules such as Analog I/O Modules (cat. no. 1771-IF, -OF) or the Thermocouple/Millivolt Input Module (cat. no. 1771-IXE). By using a block-transfer instruction, you can transfer up to 64 16-bit data words from block-transfer I/O modules to the data table or vice versa (Figure 15.1).

There are two types of block-transfer instructions:

- Block-transfer-read instruction transfers data from a block-transfer I/O module to the data table.
- Block-transfer-write instruction transfers data from the data table to a block-transfer I/O module.

Most block-transfer modules execute both a block-transfer read and write. Such I/O modules are called bidirectional-block-transfer I/O modules because in communicating with the processor, data travels both to and from the I/O module.

Figure 15.1
Block-transfer Operation



15.2 Defining Parameters for a Block Transfer

To execute block transfers, you need to provide the processor with the following information:

Rack tells the processor the assigned rack number of the I/O chassis that contains the I/O module. This value is called the rack address and can be a value from 0 to 76 octal.

Group tells the processor which I/O group within that chassis contains the I/O module and can be a value from 0 to 7.

Module tells the processor if the I/O module is in the left or right slot of the I/O group:

If the I/O module is in the	Then the slot number is
right slot	1
left slot or a double-slotted I/O module	0

Data tells the processor the data file address:

If you are executing a	Then this address tells the processor where to
block-transfer write	get the information that is to be sent to the block-transfer-I/O module
block-transfer read	place the information that is received from the block-transfer-I/O module

Length tells the processor how many words to transfer. You can transfer up to 64 words. A length of 0 tells the processor to ask the I/O module how many words to read or write. If you are using zero for the length, create 64 words in your data file.

Cntl tells the processor where to find the block-transfer control file. This control file tells the processor where to get the data for the block-transfer instruction and must be in the binary section of the data table. We describe the contents of this file in the next section.

15.3 Block-transfer Control File

Figure 15.2 shows the block-transfer control file. This file contains four parts:

Word	Description
0	block-transfer status word
1	I/O module location word
2-5	block-transfer-write data table address and data file length
6-9	block-transfer-read data table address and data file length

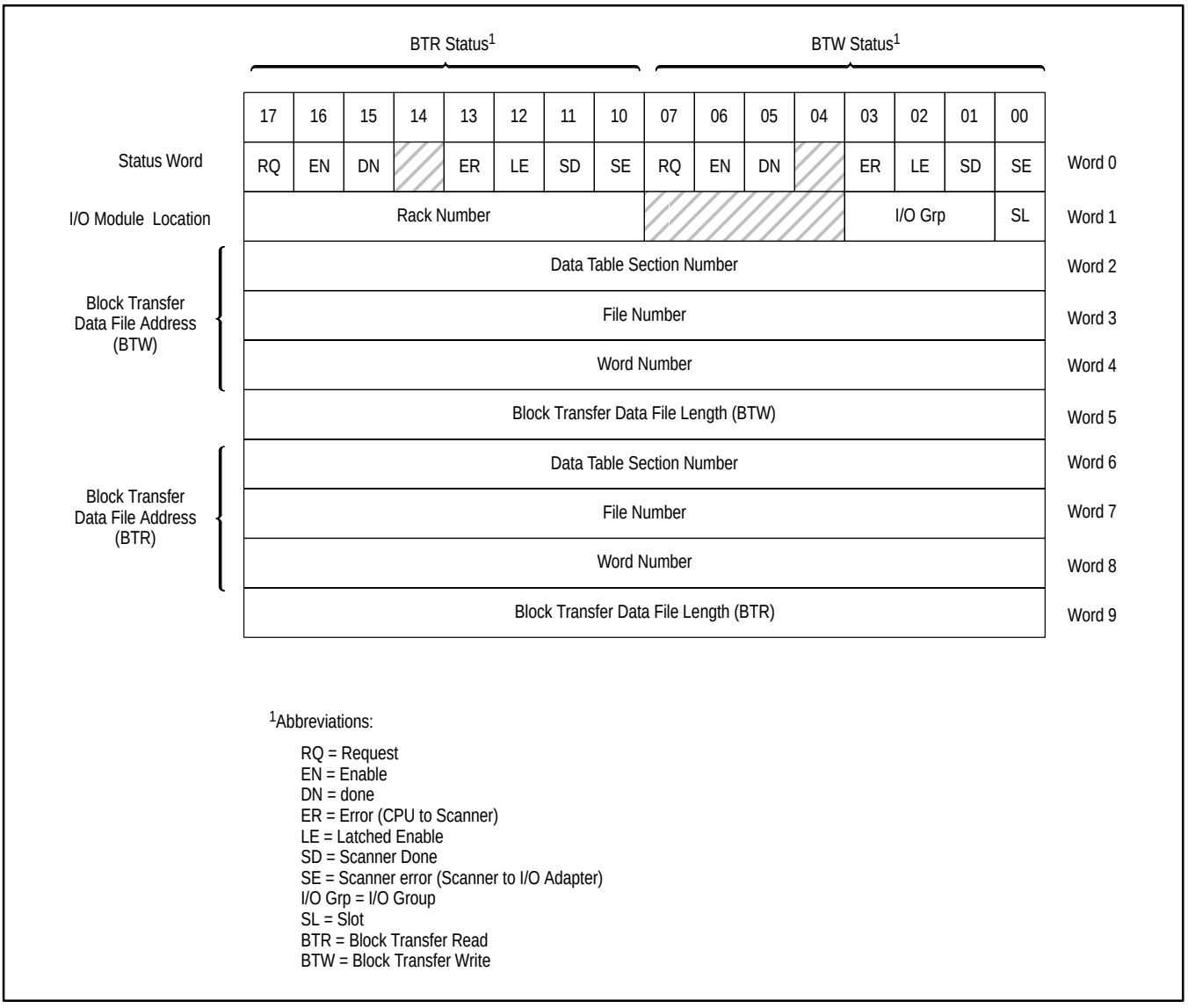
Upon entering a block-transfer instruction and the starting address for the control file on a programming device, the processor automatically creates the control file. You can use this file for block-transfer-read, -write, or -bidirectional operations. The processor requires separate block-transfer control files for each block-transfer I/O module.

15.3.1 Block-transfer Status Word

Word 0 of the block-transfer control file contains the status bits for block transfer read (bits 10 through 17) and block transfer write (bits 00 through 07):

Bits	Correspond to
00, 10 (SE)	the scanner detecting an error
01, 11 (SD)	the scanner informing the processor that a block transfer has been completed
02, 12 (LE)	the processor initiating a block transfer
03, 13 (ER)	the scanner or processor detecting an error
05, 15 (DN)	block-transfer completed
06, 16 (EN)	block-transfer enabled
07, 17 (RQ)	block-transfer request initiated by the scanner

Figure 15.2
Block-transfer Control File



[1]Abbreviations:

RQ = Request
EN = Enable
DN = done
ER = Error (CPU to Scanner)
LE = Latched Enable
SD = Scanner Done
SE = Scanner error (Scanner to I/O Adapter)
I/O Grp = I/O Group
SL = Slot
BTR = Block Transfer Read
BTW = Block Transfer Write

15.3.2 I/O Module Location Word

Word one of the block-transfer control file contains information on the I/O module's location:

Bits	Correspond to
00	slot number for the block-transfer I/O module (1 = upper slot, 0 = lower slot)
01-03	I/O group number for the block-transfer I/O module (0-7)
04-07	unused
10-17	assigned rack number for the chassis containing the block-transfer I/O module

15.3.3 Block-transfer-write Information

Words two through five of the block-transfer control file contain the data table address and the data file length for a block-transfer-write instruction:

Word	Corresponds to
2	data table section number
3	file number
4	word number
5	block-transfer-write data file length

15.3.4 Block-transfer-read Information

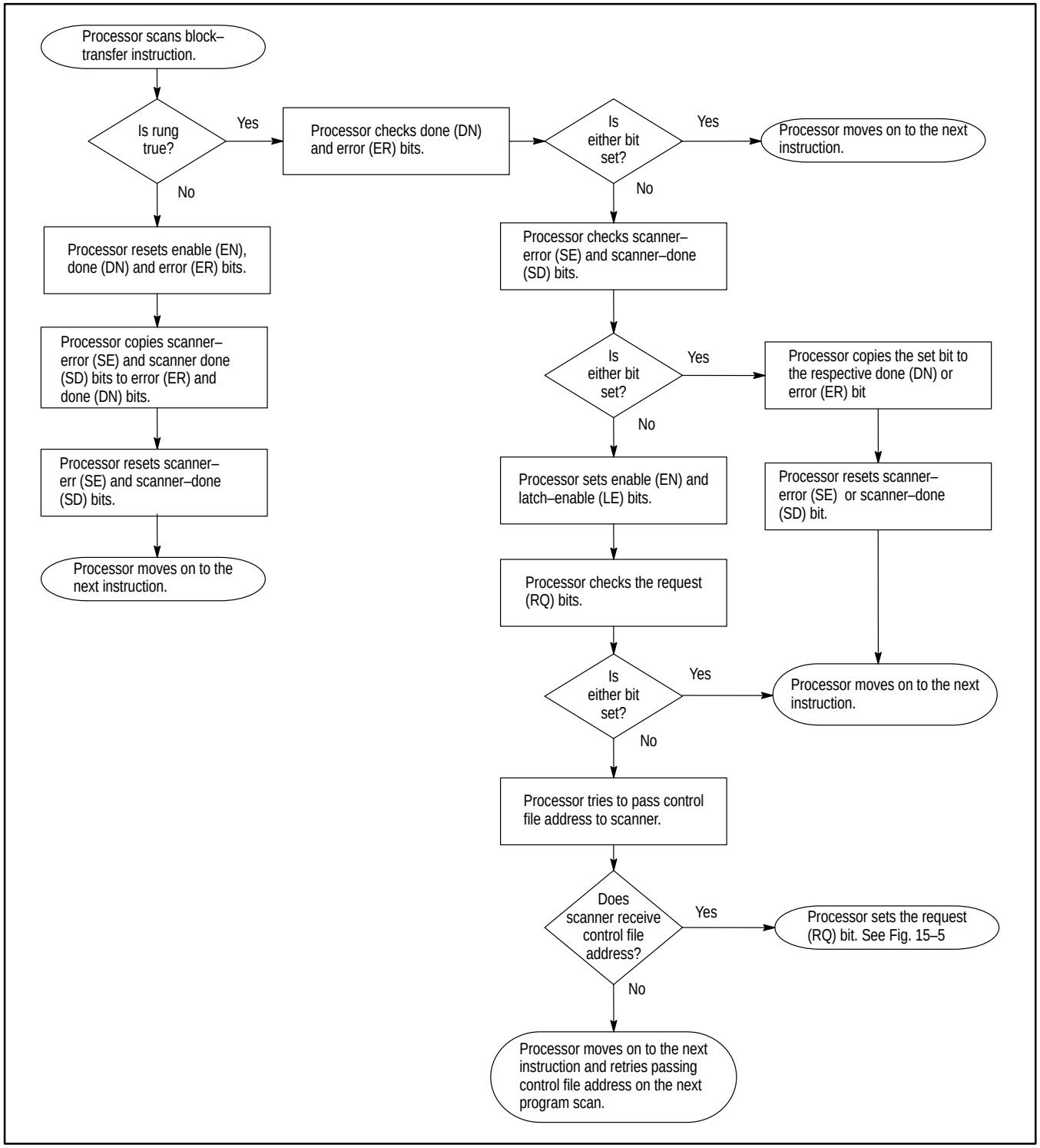
Words six through nine of the block-transfer control file contain the data table address and the data file length for a block-transfer-read instruction:

Word	Corresponds to
6	data table section number
7	file number
8	word number
9	block-transfer-read data file length

15.4 Block-transfer Instruction Operation

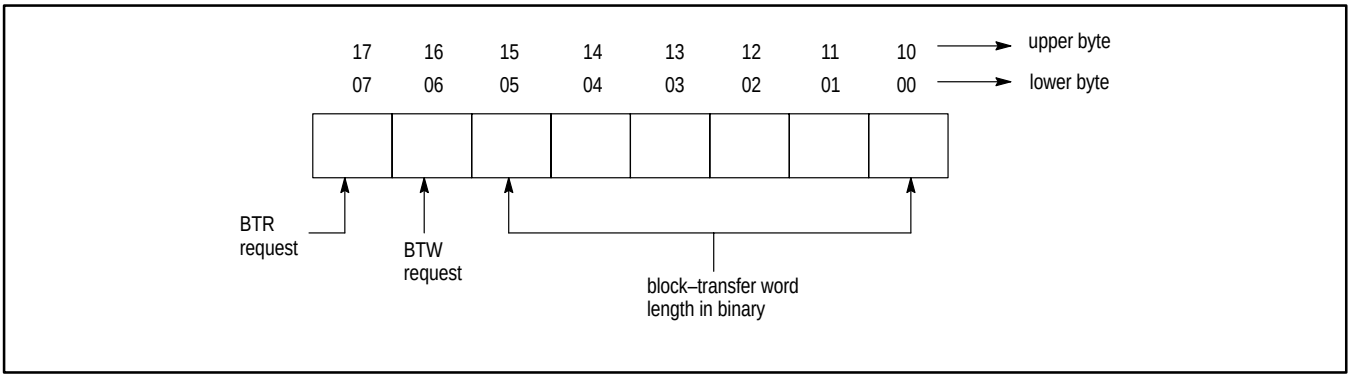
Operation of a block-transfer instruction involves communication between the processor and an I/O module via the scanner. When the processor scans a rung that contains a block-transfer instruction, it monitors the bits in the control file and sends the control file address to the scanner (Figure 15.3).

Figure 15.3
Flow Chart Showing Processor Functions for a Block-transfer Operation



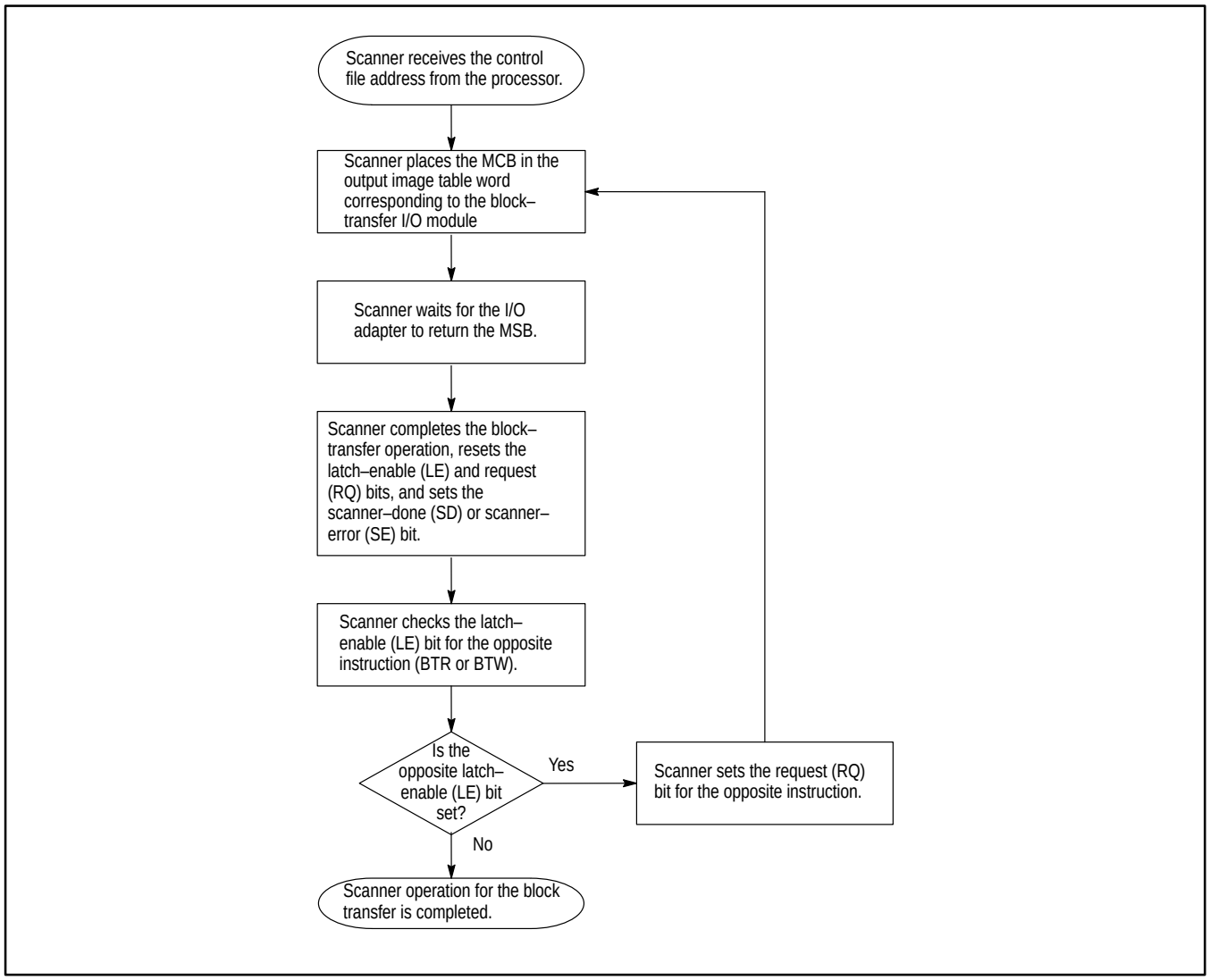
After the scanner receives the control file address from the processor, it generates a module control byte (MCB) and places it in the output image table corresponding to the block-transfer module's I/O rack location (Figure 15.4).

Figure 15.4
Module Control Byte Located in Output Image Table



During normal I/O scanning, the scanner sends the MCB to the I/O chassis. When the block-transfer module is ready to execute the block transfer, it returns a module status byte (MSB) to the scanner. The format of the MSB is the same as the MCB with the upper two bits specifying the block-transfer direction, and the lower five bits specify in the block-transfer word length. If you specify a length other than 0, the MSB will match the MCB. If you specify 0, the scanner excepts the length value in the MSB sent back by the block-transfer module. This value must be between 1 and 64. Upon receiving the MSB, the scanner immediately executes the block-transfer (Figure 15.5).

Figure 15.5
Flow Chart Showing Scanner Functions for a Block-transfer Operation



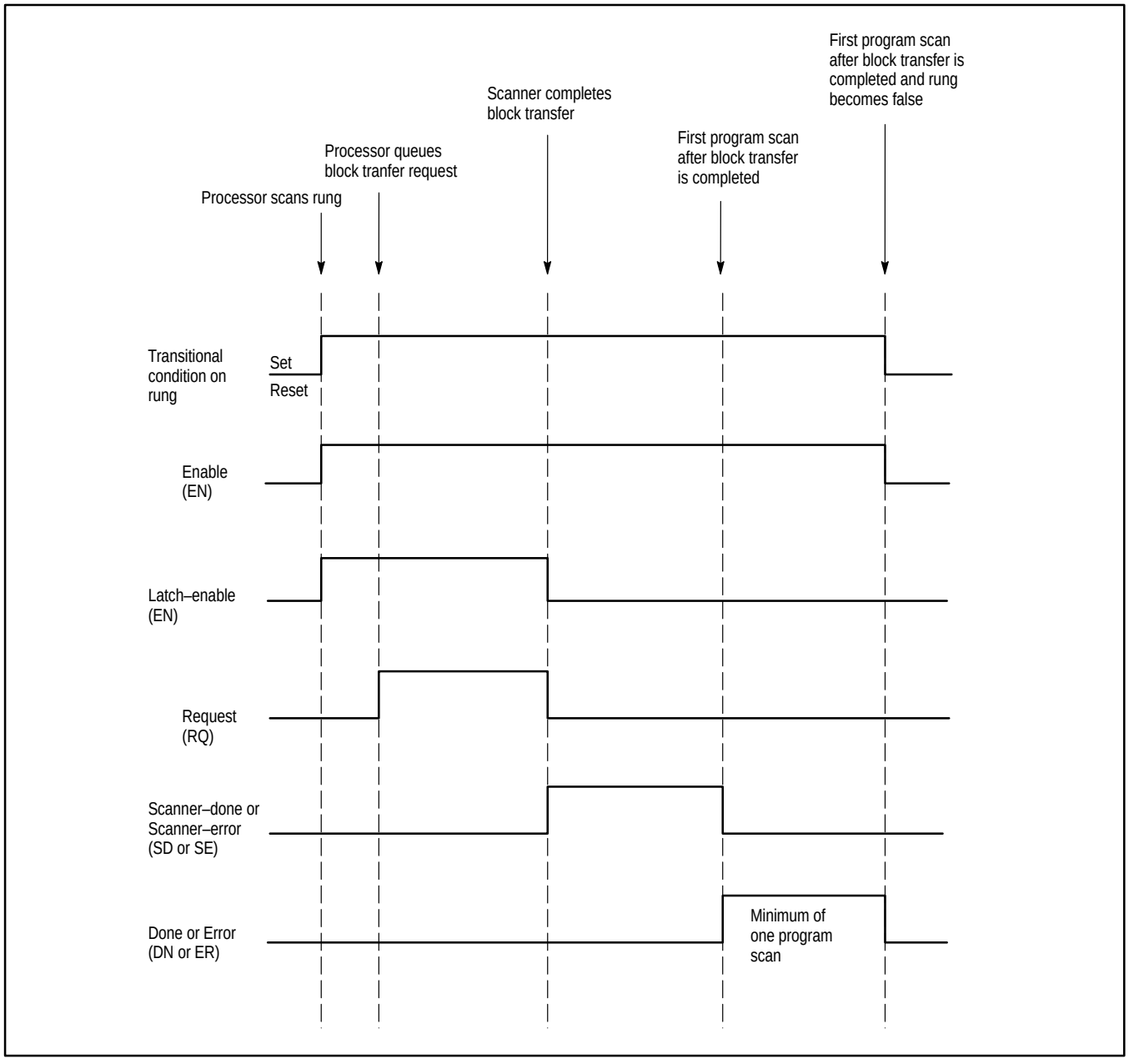
15.4.1 Executing a Block-transfer Read (BTR)

Required Parameters: I/O module location (RACK, GROUP, MODULE), data file address, file length, control file address.

Description: When a rung containing a block-transfer-read instruction goes from false to true, the processor transfers up to 64 words at a time from an intelligent I/O module to memory.

Figure 15.6 shows a timing diagram for a block-transfer-read operation.

Figure 15.6
Timing Diagram for Block-transfer Operations

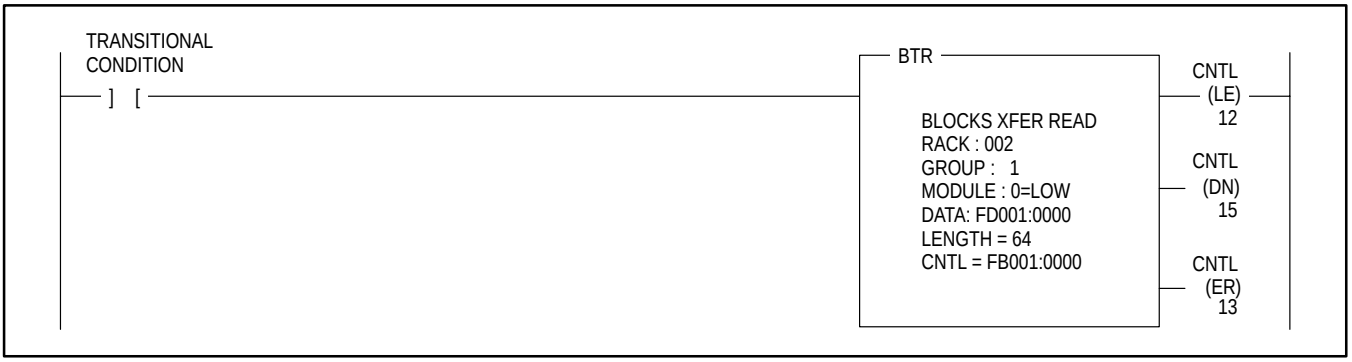


Example: Figure 15.7 shows a rung containing a block-transfer-read instruction.

If the rung goes from false to true, the processor initiates a block-transfer read of the specified block-transfer-read I/O module. In this block-transfer-read instruction:

This parameter	Tells the processor
rack (002)	the assigned rack that contains the I/O module
group (1)	the I/O group that contains the I/O module
module (0=LOW)	the slot within the I/O group that contains the I/O module
data (FD001:0000)	the data table location to read the data into
length (64)	the number of words that transfer
cntl (FB001:0000)	the location of the control file

Figure 15.7
Example Rung for a Block-transfer-read Instruction



15.4.2 Executing a Block-transfer Write (BTW)

Required Parameters: I/O module location (RACK, GROUP, MODULE), data file address, file length, control file address.

Description: When a rung containing a block-transfer-write instruction goes from false to true, the processor transfers up to 64 words at a time from memory to an intelligent I/O module.

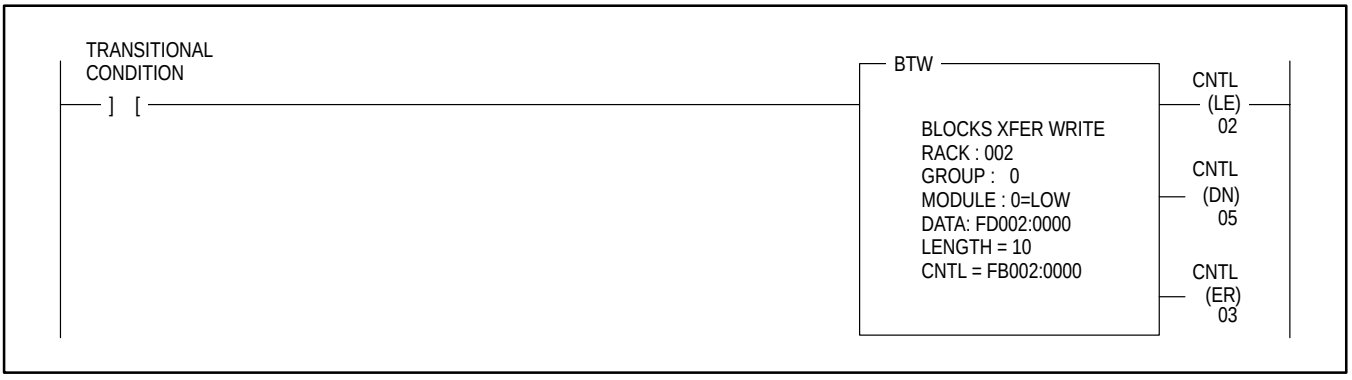
Figure 15.6 shows a timing diagram for a block-transfer-write operation.

Example: Figure 15.8 shows a rung containing a block-transfer-write instruction.

If the rung goes from false to true, the processor initiates a block-transfer write to the specified block-transfer-write I/O module. In this block-transfer-write instruction:

This parameter	Tells the processor
rack (002)	the assigned rack that contains the I/O module
group (0)	the I/O group that contains the I/O module
module (0=LOW)	the slot within the I/O group that contains the I/O module
data (FD002:0000)	the data table location that transfers to the I/O module
length (10)	the number of words that transfer
cntl (FB002:0000)	the location of the control file

Figure 15.8
Example Rung for a Block-transfer-read Instruction

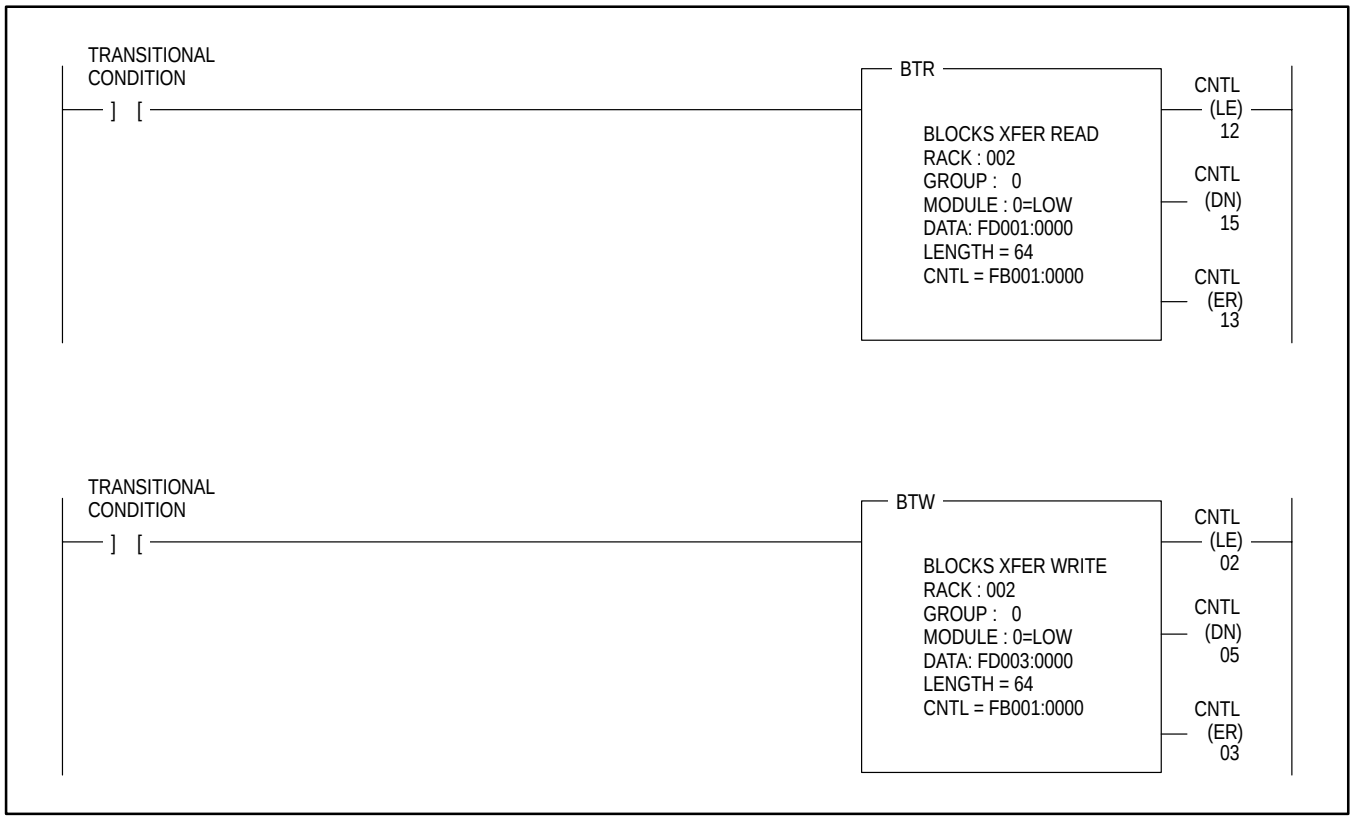


15.4.3 Executing a Bidirectional Block Transfer

To program block-transfer-read and write instructions consecutively to an I/O module that supports both operations, use the following rungs (Figure 15.9):

In these rungs, you enter the same control file for the block-transfer-read and -write instructions. Then, if the transitional conditions go from false to true, the processor executes a BTR instruction on the first rung and a BTW instruction on the second rung.

Figure 15.9
Example Rungs that Execute a Block Transfer to a Bidirectional-block-transfer I/O Module



15.4.4 Block-transfer Size Limit for 1775-S4A, -S4B, and -SR Scanners

Any individual block transfer is limited by a 64 word maximum size. 64-word block transfers can execute simultaneously on all four scanner I/O channels.

However, when using the number one 1775-S4A or -SR scanner, you must limit the total of the block transfers that take place on the four I/O channels at any one time to 72 words.

15.4.5 Example Block-transfer Diagnostic Program

Figure 15.10 shows example rungs that uses diagnostic counters and a watchdog timer with a block-transfer instruction. The watchdog timer begins timing after the processor sets latch-enable bit for the block-transfer operation. If an error occurs or the operation is not completed before the watchdog timer times out, the processor sets an alarm bit. The processor also keeps track of the number of errors that occur and the number of times that the watchdog timer times out.

The watchdog-timer-present value should be a maximum value beyond the time that the processor takes to execute the block-transfer operation. In this example, we use 0.5 seconds. For detailed information on calculating block transfer times, refer to the I/O Scanner-programmer Interface User's Manual (publication 1775-6.5.2).

Figure 15.10
Example Block-transfer Diagnostic Program

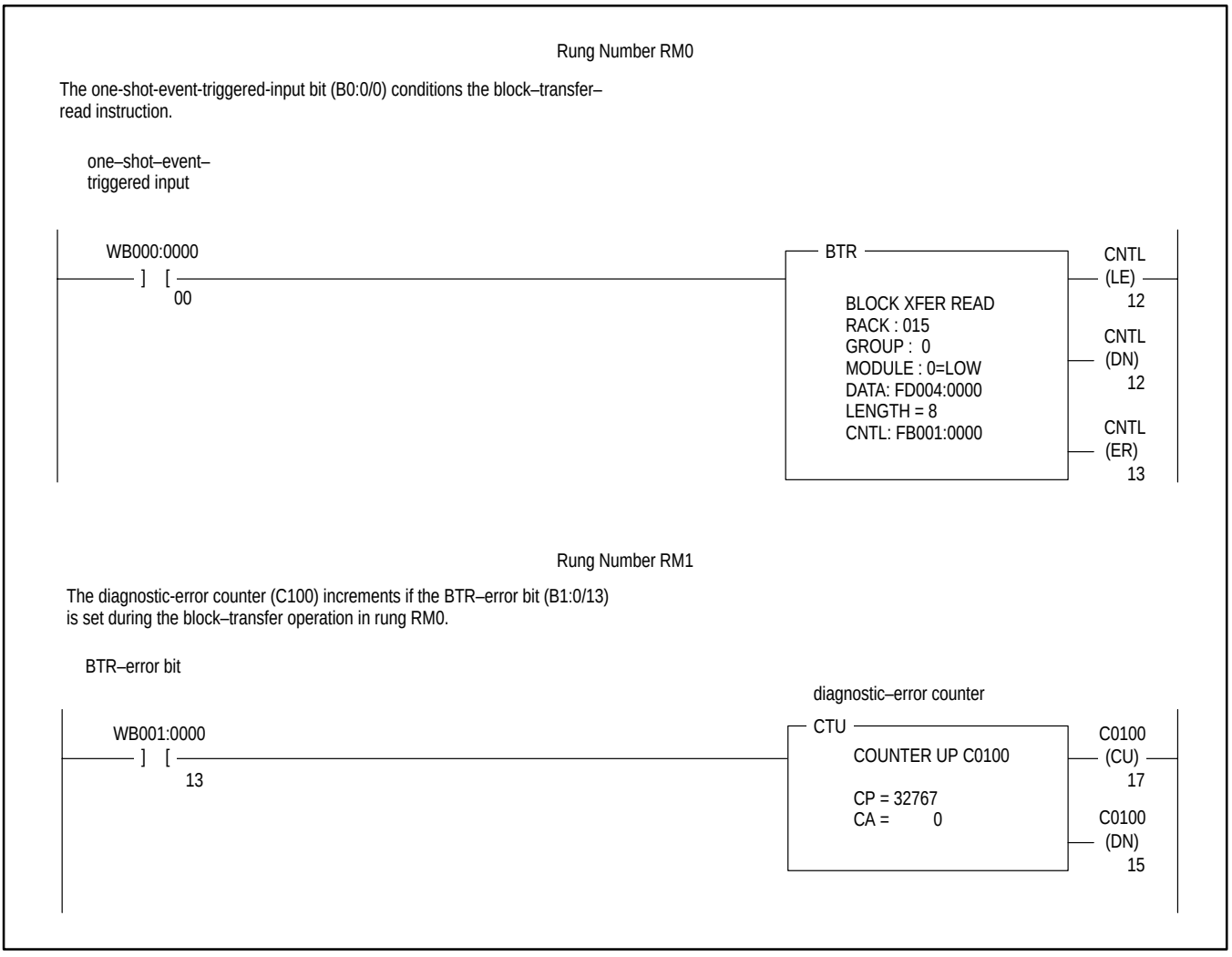
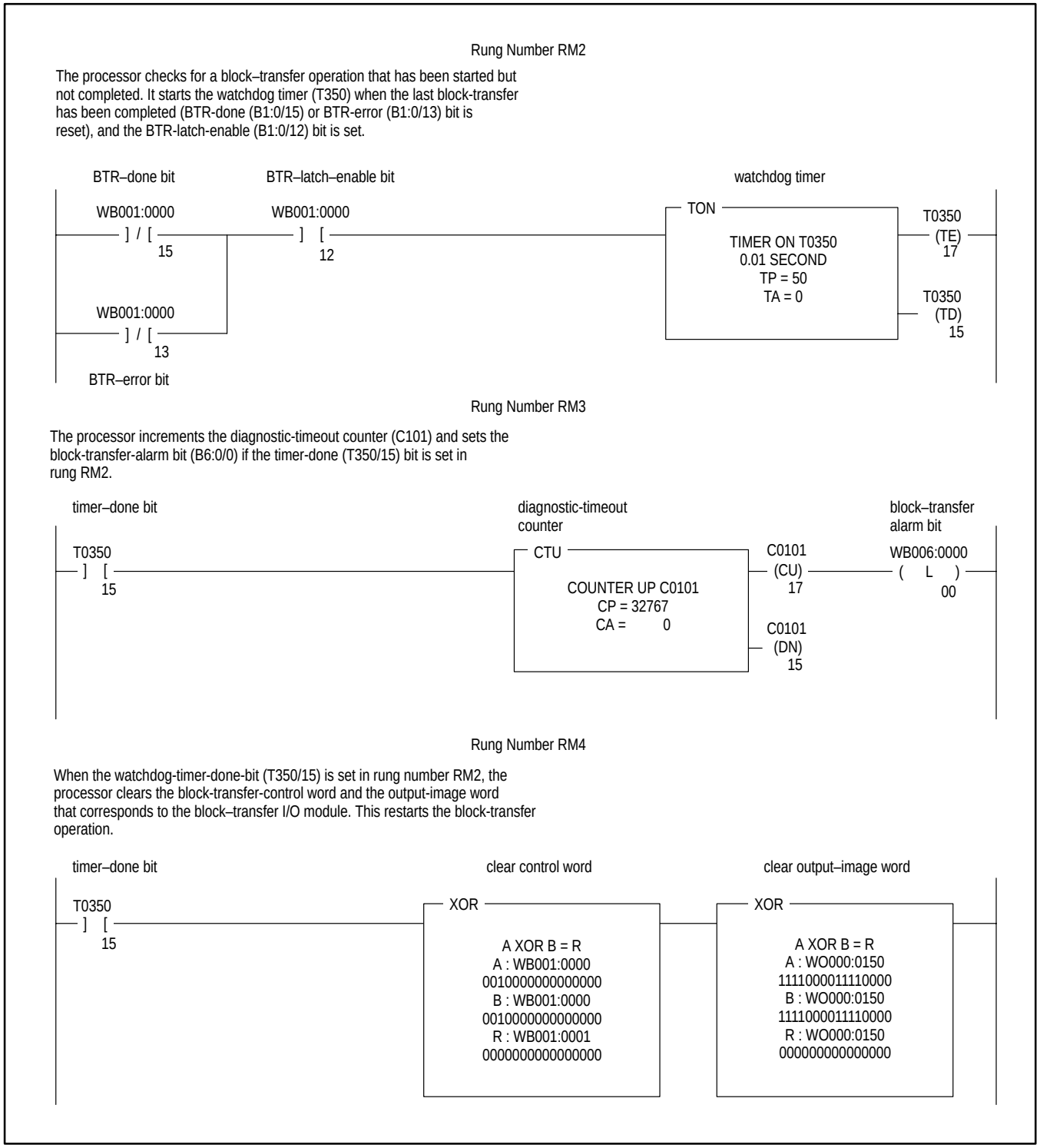


Figure 15.10
Example Block-transfer Diagnostic Program (continued)



15.5 Troubleshooting Block-transfer Errors

Once enabled, a block-transfer instruction in a ladder program sets either a done bit or an error bit. Typical block-transfer errors occur when you do not correctly enter the instruction such as when:

- The rack, group, and module numbers do not match the location of the installed module.
- You enter a file length that is greater than 64, or the file length does not coincide with the particular block transfer module.
- You did not create the data file, or the address entered does not match the file that you created.

Communication problems can result from improper connections between the scanner and the I/O adapter. When the scanner encounters a communication fault, it tries twice to complete the transfer. After two tries, it sets the error bit.

Chapter 16

Using the Message Instruction

16.0 Chapter Objectives

In this chapter, we describe how to use the message instruction to execute a task on a PLC–3 module. After reading this chapter, you should understand:

- how to apply the message instruction
- how the message control file works
- how the message instruction operates
- different message categories available with the controller
- how to use symbols to define memory addresses or messages

16.1 Applying the Message Instruction

You can use the message instruction to request that a specified target module execute a message procedure or command in the ladder program. Applications using the message instruction include executing a:

- report generation command or procedure on an I/O scanner–message module (cat. no. 1775–S4B)
- GA Basic command or procedure on a peripheral communication module (cat. no. 1775–GA)
- backup communication command between memory communication modules (cat. no. 1775–MX) in a PLC–3 backup system
- data highway procedure or command on a communication adapter module (cat. no. 1775–KA) or a Data Highway II interface module (cat. no. 1779–KP3)

To execute a message instruction, you need to provide the processor with the following information:

- address of the message control file. This file should be a binary file and can have a starting word address other than zero.
- extended address for the module that executes the message command or procedure
- message type which is always 1
- command or procedure name

16.2 Message Control File

Figure 16.1 shows the message control file. This file contains four parts:

Word	Description
0	message status word
1	message type word
2-9	module extended address
10-n	message contents

Upon entering a message instruction and the starting address for the control file on a programming device, the processor automatically creates the control file. You can use the status word to monitor message instruction operation.

CAUTION: If you are using the same data table file for other messages or purposes, make sure that you leave appropriate space.

16.2.1 Message Status Word

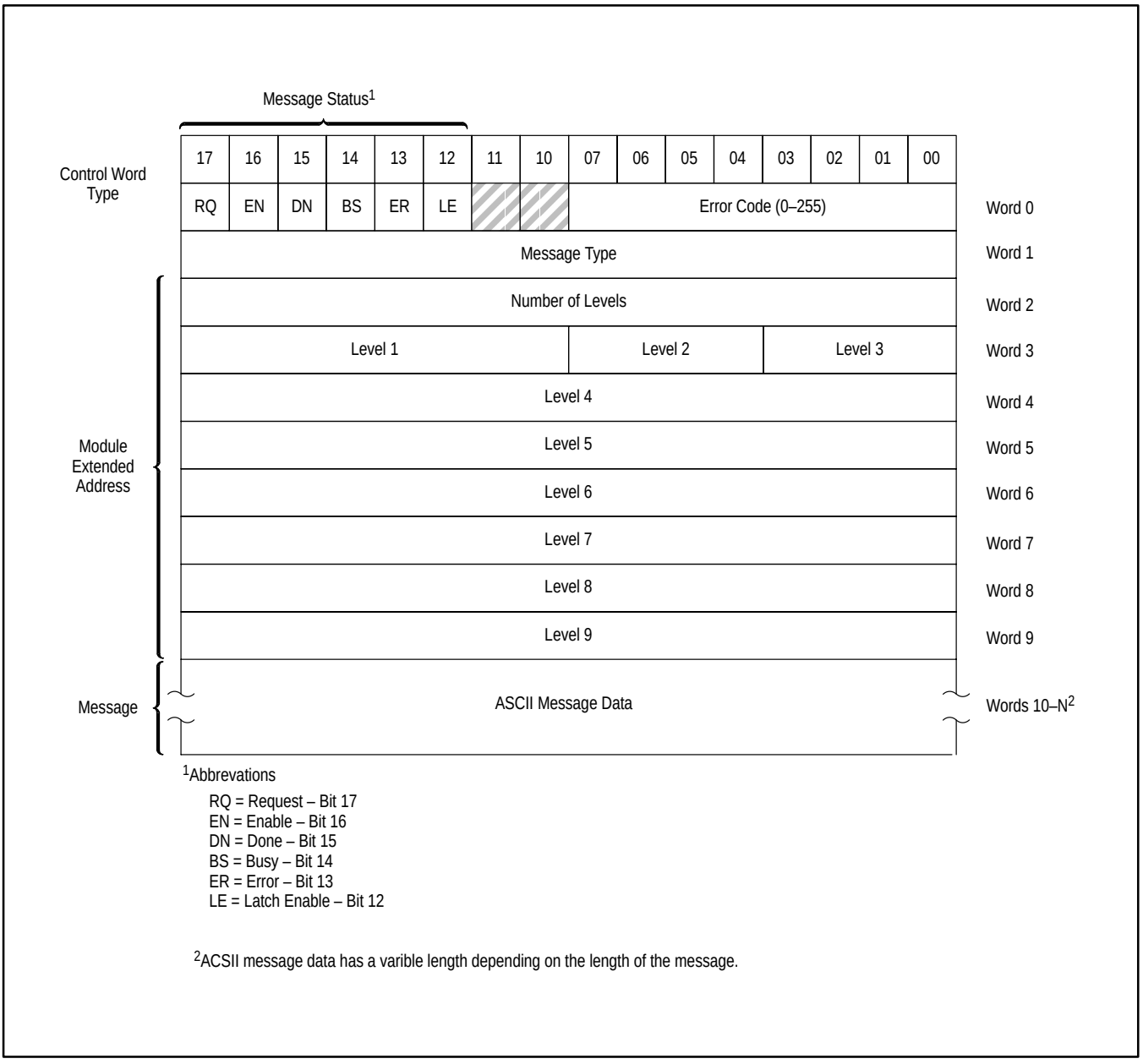
Word 0 of the message control file contains the following status bits:

Bits	Correspond to
00-07	the error code that corresponds to the message command error that caused the processor to set bit 13
10-11	unused
12 (LE)	message rung was true at some point and message execution has not been completed
13 (ER)	message execution error
14 (BS)	message execution in progress
15 (DN)	message execution completed
16 (EN)	message rung is true
17 (RQ)	message request has been received by the target module

16.2.2 Message Type Word

Word one of the message control file contains the message type. This word should always be 1.

Figure 16.1
Message Control File



[1]Abbrevations

RQ = Request – Bit 17
EN = Enable – Bit 16
DN = Done – Bit 15
BS = Busy – Bit 14
ER = Error – Bit 13
LE = Latch Enable – Bit 12

[2]ACSII message data has a varible length depending on the length of the message.

16.2.3 Module Extended Address

Words two through nine of the message control file contain the extended addressing specifying the module that executes the message procedure or command:

Word	Corresponds to
2	number of levels in the extended address
3	first three levels in extended address
4	fourth level in extended address
5	fifth level in extended address
6	sixth level in extended address
7	seventh level in extended address
8	eighth level in extended address
9	ninth level in extended address

Normally, you need only three levels to specify a module address. However, you may need more levels as required. For example, the extended address that specifies local channel 0 of a peripheral communication module with its thumbwheel set to one requires five levels, E2.8.1.3.0.

For detailed information on:

- extended addressing, refer to chapter 14
- target module, refer to the corresponding user's manual

16.2.4 Message Contents

Words 10 and on contain the message. The processor stores two ASCII characters per word.

16.3 Using the Message Instruction (MSG)

Required Parameters: control file address, module extended address, and message command or procedure name

Description: When a rung containing a message instruction is true:

1. The processor sets the enable (EN) bit (16) and latch enable (LE) bit (12) bits and clears the rest of the control word.

2. The processor attempts to pass the address of the control file to the target module that is to execute the command or procedure by using the extended address. Upon passing the address, it sets the request (RQ) bit (17). If it cannot pass the address, it does not set the request bit and tries again on the next scan. If the target module does not exist, it sets error code 255.

3. The target module sets the busy bit (bit 14) when it starts processing.

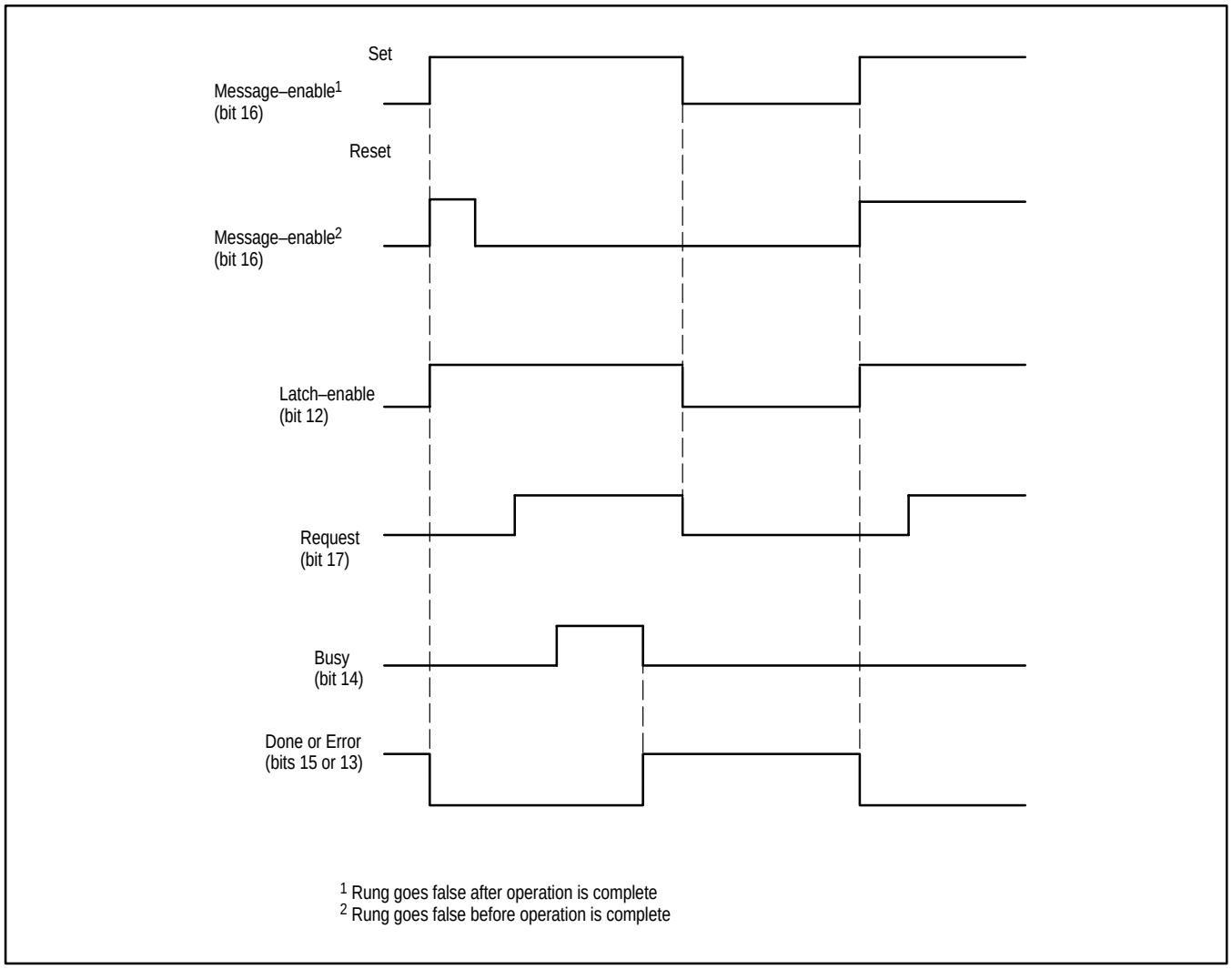
4. The target module resets the busy (BS) bit and set the done (DN) bit (15) or error (ER) bit (13) at the completion of the requested message.

5. The processor resets the request (RQ), busy (BS), enable (EN), and latch enable (LE) bits when the rung becomes false.

6. The processor resets the done (DN) or error (ER) bit when the rung becomes true again.

Figure 16.2 shows you timing diagrams for the message instruction.

Figure 16.2
Message Control File



Example: Figures 16.3, 16.4, 16.5, and 16.6 shows rungs that execute message instruction if the input condition(s) on the rung are true.

Figure 16.3 shows a message instruction that executes a report generation command on an I/O scanner–message handling module. In this message instruction:

This parameter	Tells the processor
CTL (FB5:20)	the location of the message control file
CH (E2.7.1)	the extended address for the I/O scanner-message handling module with its thumbwheel set to 1
PRINT $D2:0	the command to be executed by the module

Figure 16.3
Example Rung that Executes a Report Generation Command on an I/O Scanner-Message Handling Module

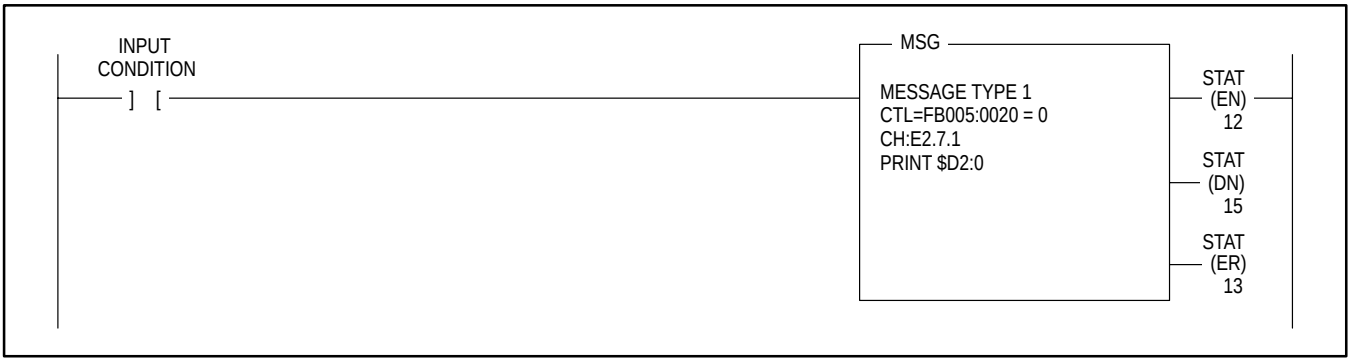


Figure 16.4 shows a message instruction that executes a GA Basic procedure on a peripheral communication module. In this message instruction:

This parameter	Tells the processor
CTL (FB6:0)	the location of the message control file
CH (E2.8.1.3.0)	the extended address for local RS-232-C channel 0 on the peripheral communication module with its thumbwheel set to 1
@TIME	the procedure to be executed by the module

Figure 16.4
Example Rung that Executes a GA Basic Procedure on a Peripheral Communication Module

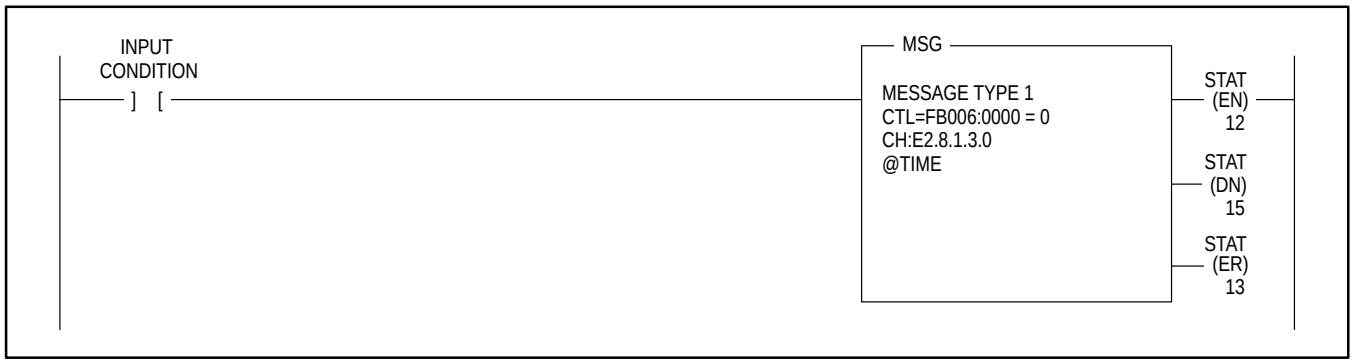


Figure 16.5 shows a message instruction that executes a backup communication command between memory communication modules in a PLC–3 backup system. In this message instruction:

This parameter	Tells the processor
CTL (FB0:0)	the location of the message control file
CH (E2.14.1)	the extended address for the memory communication modules
START $N2, T	the command to be executed between the modules

Figure 16.5
Example Rung that Executes a Backup Communication Command between Memory Communication Modules in a PLC-3 Backup System

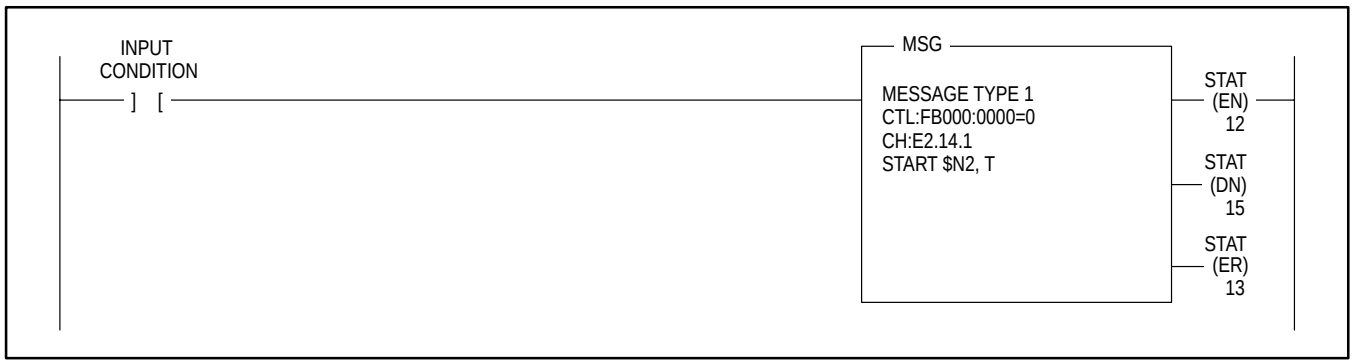
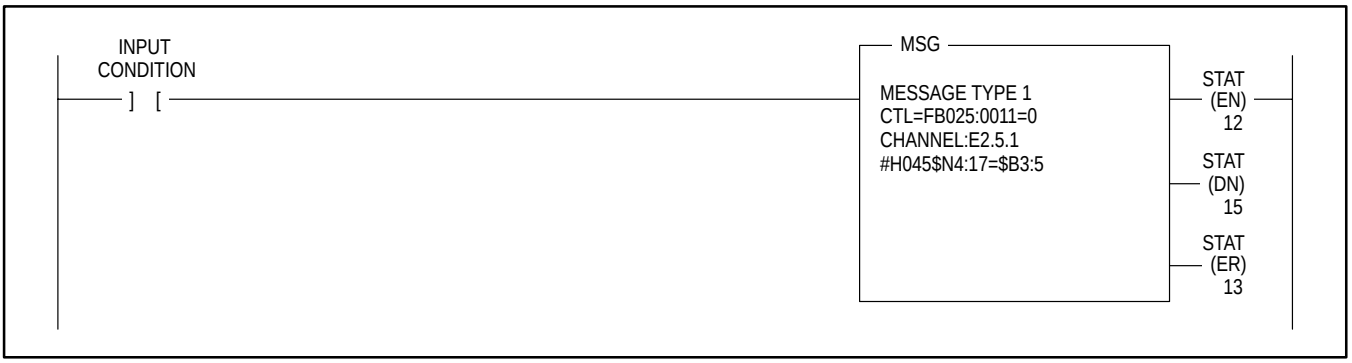


Figure 16.6 shows a message instruction that executes a data highway command on a communication adapter module. In this message instruction:

This parameter	Tells the processor
CTL (FB25:11)	the location of the message control file
CH (E2.5.1)	the extended address for the communication adapter module with its thumbwheel set to 1
#H045$N4:17=$B3:5	the command to be executed by the module

Figure 16.6
Example Rung that Executes a Data Highway Command on a Communication Adapter Module



16.4 Message Categories

In conjunction with the message instruction, the processor supports different message categories that can be executed by other PLC–3 modules. These message categories are stored in the message area (E5) and include:

- report generation or GA Basic
- rung comments
- terminal messages (MACROS)
- data highway messages
- assistance messages (HELP)

For detailed information on creating and executing message categories, refer to the user's manuals for the program loader and the module that executes the procedure.

16.4.1 Report Generation or GA Basic Procedures

Purposes: to use the controller's report generation capabilities.

Executed by: I/O scanner–message handling module or peripheral communication module

Message number range: MR0 – MR32767

Message address range: E5.1.1.0 – E5.1.1.32767

16.4.2 Rung Comments

Purpose: to document individual rungs in the ladder program.

Executed by: program loader

Message number range: MC0 – MC32767

Message address range: E5.1.2.0 – E5.1.2.32767

16.4.3 Terminal Messages (MACROS)

Purpose: to store groups of commands used by the program loader.

Executed by: program loader

Message number range: MT0 – MT32767

Message address range: E5.1.3.0 – E5.1.3.32767

Macro number MT0 is automatically executed at power up to initialize the program loader. By defining a set of commands as MT0, you can:

- change the display mode at power up
- display a message on the CRT at power up
- initialize the peripheral port

The commands and instructions in a macro execute in the order that you enter them. The program loader prompts you for missing entries, so that you can enter rung instructions without the addresses.

16.4.4 Data Highway Procedures

Purpose: to execute groups of Data Highway commands on a Data Highway or Data Highway II link.

Executed by: communication adapter module or Data Highway II interface module.

Message number range: MHO – MH32767

Message address range: E5.1.4.0 – E5.1.4.32767

16.4.5 Assistance Messages (HELP)

Purpose: to provide command and instruction descriptions.

Executed by: program loader

Message number range: MA0 – MA32767

Message address range: E5.1.5.0 – E5.1.5.32767

You can create help messages yourself or use a PLC–3 Assistance Message Data Cartridge (cat. no. 1775–ZB). This tape uses 25K words of memory.

16.5 Using Symbols

In addition to message categories, the processor supports symbols to represent data. Symbols are stored in the system symbols area (E6) and are used for two purposes:

- a **procedure name** assigns a name to a message or procedure.
- A **symbolic address** assigns a name to any valid extended address.

Through the program loader, you can use the symbol name to reference the procedure or address.

Chapter 17

Writing the Ladder Program

17.0 Chapter Objectives

Instructions and commands discussed in previous chapters are tools that you can use to write the ladder program. But writing a program involves more than just entering instructions and commands. After reading this chapter, you should understand how to use the following programming aspects to systematically develop and implement ladder programs:

- evaluating the process
- assigning I/O addresses
- assigning internal storage addresses
- evaluating application considerations
- managing memory

17.1 Evaluating the Process

Before developing a ladder program, you must understand the process that the program controls. Begin by examining the process. Determine the input conditions required to turn output devices on or off. Also, consider the sequences in which the output devices must operate, and the length of time that each output must remain in a given state. Wiring or logic diagrams are often useful when examining the process.

Once you understand the process, sketch the program logic. You can use any type of logic diagram, although the ladder–diagram type is the most commonly used with programmable controllers.

17.2 Assigning the I/O Addresses

Each input or output address corresponds to a specific assigned I/O rack, I/O group, and terminal (chapter 4). Therefore, do not assign I/O addresses arbitrarily. You must consider the locations of the input and output devices. The PLC–3 Family Controller Installation and Operations Manual (publication 1775–6.7.1) contains guidelines for I/O addressing.

17.3 Assigning Internal Storage Addresses

There are two primary considerations in assigning internal storage addresses:

- making efficient use of memory
- making the addresses easy to remember

To use memory efficiently, keep the addresses as low as possible. For example, if the program uses 10 counters, number them C0 through C9, and when storing 10 binary words, use WB0:0 through WB0:9. This

allocates memory for only the required words. If you number the counters C100 through C109, the processor allocates memory for 110 counters. The program uses 10 counters while 100 are unused. Similarly, if you store words in WB500:500 through WB500:509, the processor allocates memory for 500 unused words in binary file 500, and pointers are created for 500 unused files.

You can also use words in input file 0 that do not correspond to input I/O groups for storage. However, even if only one module in an I/O group is an input module, the I/O scan writes over the entire word and therefore cannot be used for storage.

When using files, the processor executes file instructions on file number 0 to 19 for a data table section faster than file numbers greater than 19.

Addresses are easier to remember if you number them in logical sequence. For example, if the 10 counters correspond to events occurring at ten different locations, determine how you would number the locations 0 through 9 and number the counters accordingly.

17.4 Evaluating Application Considerations

Many processes pose unique problems that require special attention. Although we cannot provide detailed solutions to all these problems in this manual, we discuss some of the more common problems, including:

- short pulses
- orderly shutdown requirements
- diagnostic needs

17.4.1 Short Pulses

To ensure that all rungs in the program see a given input condition, the input must remain in one state for at least as long as the sum of the I/O scan, the input module delay, and the program scan.

Any input pulse of shorter duration may be missed by the rung(s) dependent on that input. The ideal way to handle these pulses is to use external circuits to ensure that all inputs remain in one state for at least the required time. If you cannot add the necessary circuits, you can reduce the probability of missing these pulses in other ways.

One method is to repeat the rungs that examine the input pulse. For example, if an input pulse duration is equal to the sum of the input module delay, the I/O scan, and 70% of the program scan, repeat rungs dependent on the input at least twice in the ladder program, with approximately half the program between the two rungs (or one–third of the program between the rungs if the rung appears three times).

Another method is to use a real time interrupt routine. In the example above, the same effect can be obtained by setting the real–time–interrupt interval to a time equal to one–half the program scan.

17.4.2 Orderly Shutdowns

In many applications, an orderly shutdown can minimize damage either to I/O devices or the product in process when the shutdown occurs. Possible reasons for the shutdown include:

- Planned shutdown – In this case, a programmed shutdown sequence has very few restrictions. it can take as long as needed to avoid damage, and can fully control the equipment used in the process.
- I/O device fault – In this case, a programmed shutdown sequence has a few more restrictions. The loss of an I/O device reduces control over the process. and if the extent of the failure is unknown, the shutdown sequence must be able to handle the worst case.
- Controller fault – In this case, you can use a fault routine. Shutdown sequences in fault routines are not as extensive as other programmed shutdowns. Time is an important consideration in a supply which causes a shutdown to occur very quickly. Also, the fault routine executes only once, so the shutdown sequence must be completed in one pass.

17.4.3 Diagnostics

When writing the program, consider the diagnostic needs of the devices being controlled. Instructions like diagnostic detect and file bit compare are valuable tools in troubleshooting the system.

17.5 Managing Memory

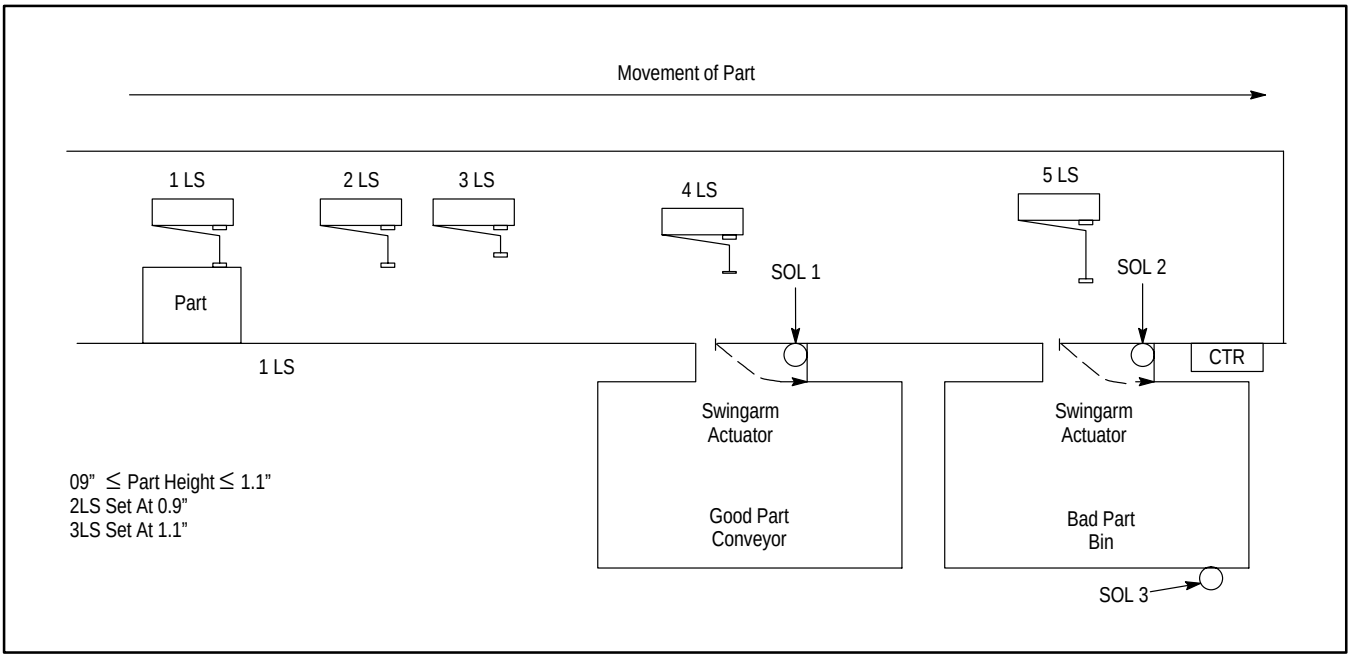
Memory management for the data stored in your controller is an important part of the ladder program. By documenting memory usage in your ladder programs, you can easily modify or make changes to it. To aid you in memory management of your controller, appendix C contains memory management forms.

17.6 Example Program

The following example exemplifies how to develop the ladder program.

The application involves separating good parts from bad parts. Figure 17.1 shows a part moving along a conveyor belt. Each part turns on a series of limit switches and is sorted according to its height. The desired height is 1.0”±0.1”.

Figure 17.1
Conveyor Belt Example



17.6.1 Separating Good Parts

If a part turns on limit switch 2LS but not 3LS, the part is greater than or equal to 0.9″ and less than or equal to 1.1″ making it a good part. As a good part:

1. Storage bit (SB3) latches on.
2. The part continues moving until it turns on switch (4LS) which turns on solenoid (SOL1).
3. Solenoid (SOL1) moves the swingarm actuator, directing the part into the good part conveyor.

17.6.2 Separating Bad Parts

If the part turns on both limited switches (2LS and 3LS) or does not turn on either switch, it is too large or too small making it a bad part. As a bad part:

1. Storage bit (4SB) latches on.
2. The part continues moving and although it turns on limit switch (4LS), it continues moving until it turns on limit switch (5LS) which turns on solenoid (SOL1).
3. Solenoid (SOL2) moves the swingarm actuator, directing the part onto the bad part bin.

Each time that a part enters the bad part bin, a counter increments. When the bin is full (count complete), SOL3 turns on which opens the bottom of the bin long enough to empty it. Then, the counter resets automatically.

17.6.3 Conveyor Operation for Good Parts

Each time that a new part enters the conveyor belt, limit switch (1LS) turns on which unlatches the storage bits and begins a new cycle.

Pushbutton switches (START or STOP) start or stop the conveyor motor. Motor starter (MS1) controls the conveyor motor, and a watchdog timer monitors the flow of parts. If parts should jam causing a delay between limit switches (1LS and 4LS), the timer times out which turns the conveyor motor off. Another watchdog timer detects if a part jams beneath limit switches (4LS or 5LS). A conveyor indicator (RUN) and a parts indicator (JAM) allow remote observation of the conveyor operation.

Additional documentation (not shown) would include a power distribution schematic showing a hardwired master control relay and emergency stop switches.

17.6.4 Developing the Ladder Program

Figure 17.2 shows the logic developed as a ladder program. Table 17.A shows data table addresses assigned to the hardwired devices. You should develop the ladder program by analyzing the logic required to operate the machine.

Table 17.A
Data Table Addresses for Hardwired Devices

Input Device	Address
pushbutton (STOP)	I00/00
pushbuttom (START)	I00/01
motor starter auxiliary	I00/02
limit switch (1LS)	I00/03
limit switch (2LS)	I00/04
limit switch (3LS)	I00/05
limit switch (4LS)	I00/06
limit switch (5LS)	I00/07

Output Device	Address
motor starter (MS1)	O00/00
conveyor indicator (RUN)	O00/01
good part solenoid (SOL1)	O00/02
bad part solenoid (SOL2)	O00/03
bin dump solenoid (SOL3)	O00/04
detect indicator (JAM)	O00/05

Internal Functions	Address
storage bit 1 (SB1)	B00/01
storage bit 2 (SB2)	B00/02
storage bit 3 (SB3)	B00/03
storage bit 4 (SB4)	B00/04
storage bit 5 (SB5)	B00/05
retentive timer (watchdog)	T0
timer (bin dump)	T1
timer (watchdog)	T2
counter	C0

Figure 17.2
Example Ladder Program for the Conveyor Belt

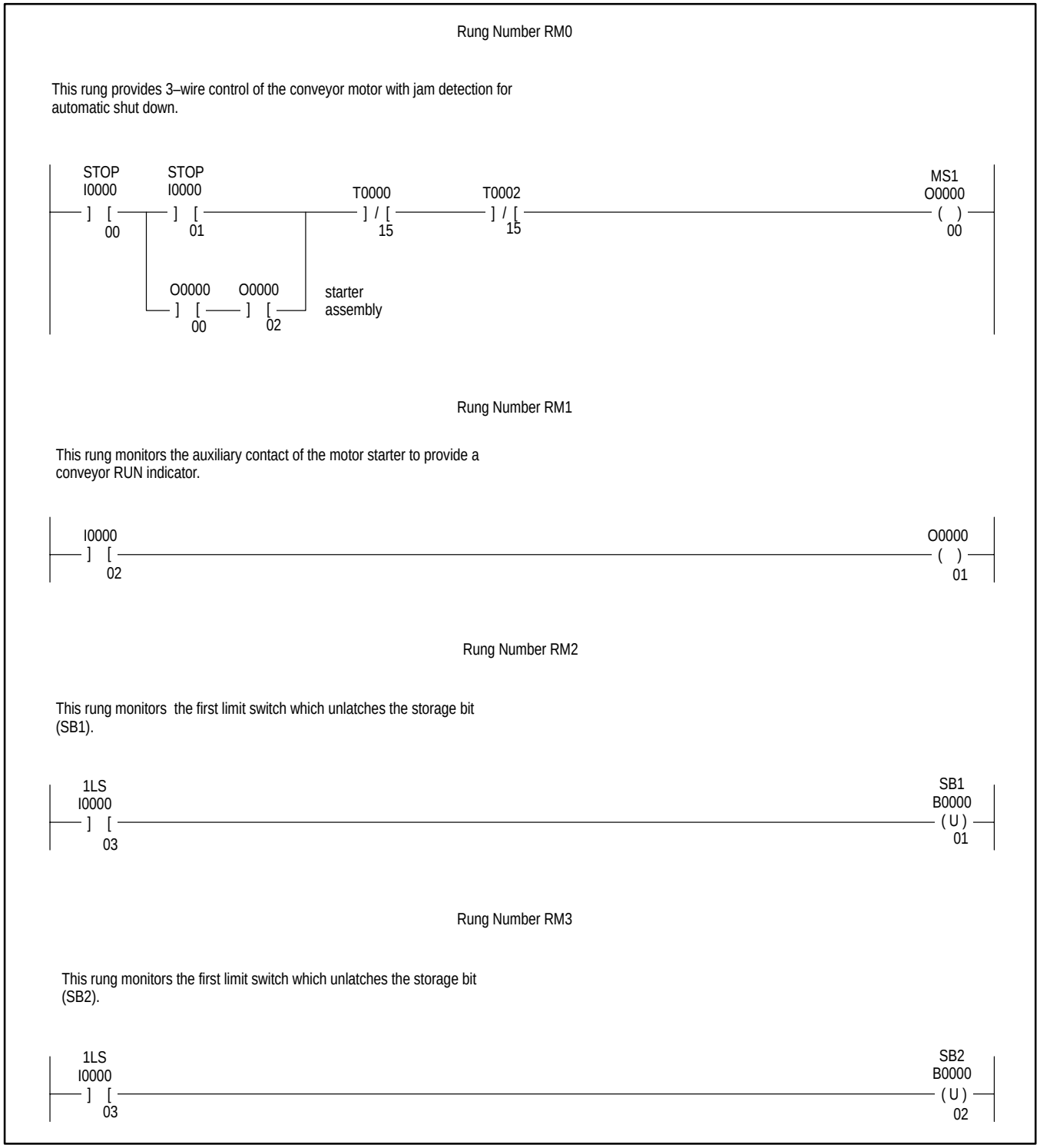
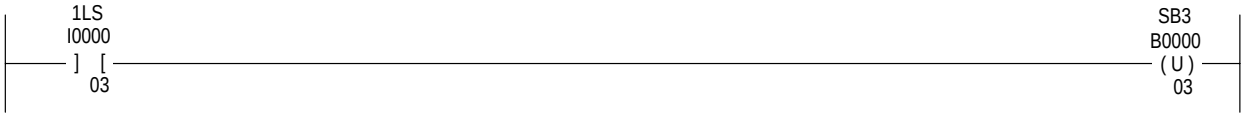


Figure 17.2
Example Ladder Program for the Conveyor Belt (continued)

Rung Number RM4

This rung monitors the limit switch (1LS) which unlatches the storage bit (SB3).



Rung Number RM5

This rung monitors the limit switch (1LS) which unlatches storage bit (SB4).

Rung Number RM6

This rung monitors the limit switch (1LS) which enables a retentive timer that is latched by the timer enable bit. If the timer times out, the processor detects a jam condition.

Rung Number RM7

This rung monitors limit switch (4LS) and the (START pushbutton) which resets the watchdog timer. If reset prior to five seconds, a jam did not occur between limit switched (1LS and 4LS). This rung does not detect a jam beneath limit switch (4LS) or to the right of 4LS.

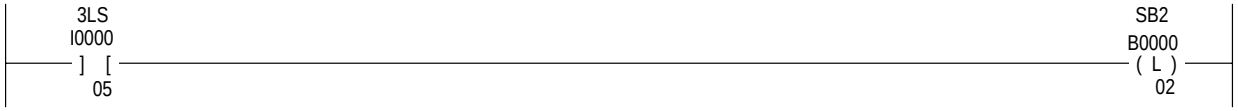
Figure 17.2
Example Ladder Program for the Conveyor Belt (continued)

Rung Number RM8

This rung monitors limit switch (2LS). When a part passes 2LS, storage bit (SB1) latches, if the part's height is greater than or equal to 0.9 inch. SB1 remains unlatched if the part's height is less than 0.9 inch.

Rung Number RM9

This rung monitors limit switch (3LS). When a part passes 3LS, storage bit (SB2) latches, if the part's height is greater than 1.1 inch. SB2 remains unlatched if the part's height is less than 1.1 inch.



Rung Number RM10

This rung latches storage bit (SB3), if a part is within tolerance.

Rung Number RM11

This rung latches storage bit (SB4), if a part is not in tolerance.

Figure 17.2
Example Ladder Program for the Conveyor Belt (continued)

Rung Number RM12

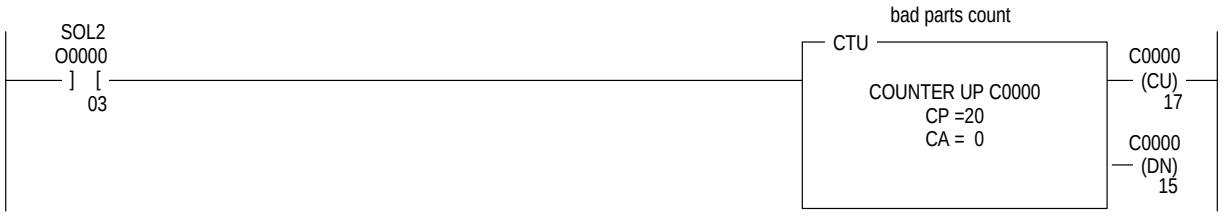
This rung monitors limit switch (4LS) which turns on solenoid (SOL1). SOL1 actuates the swing arm which directs good parts to the conveyor.

Rung Number RM13

THis rung monitors limit switch (5LS) which turns on solesnoid (SOL2). SOL2 actuates the swing arm which directs bad parts to the bad parts bin.

Rung Number RM14

This rung monitors solenoid (SOL2) to increment the up–counter one count for each bad part.



Rung Number RM15

This rung latches storage bit (SB5) when the counter done bit set (CA = 20).

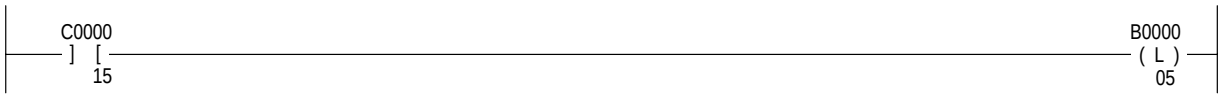


Figure 17.2
Example Ladder Program for the Conveyor Belt (continued)

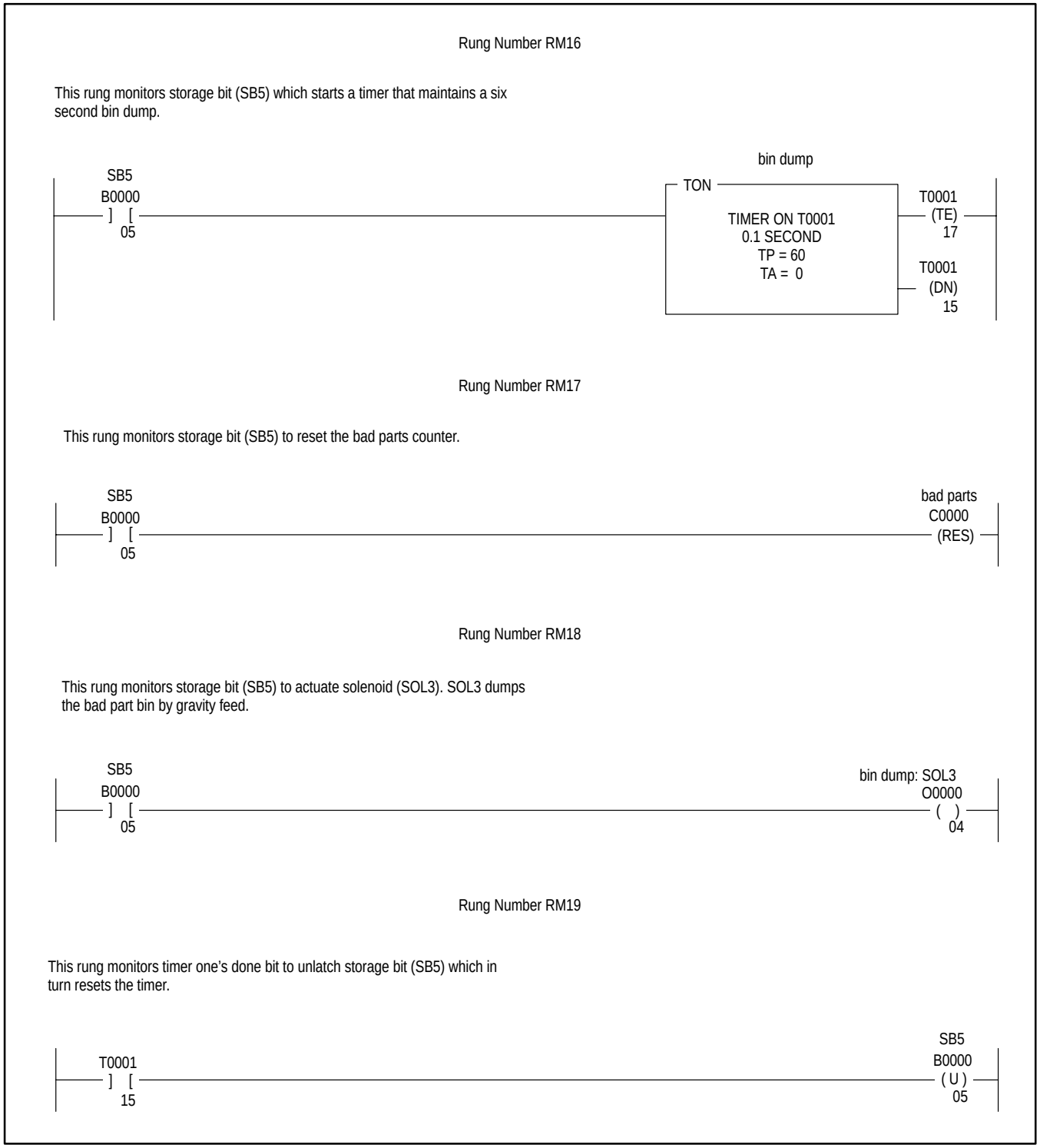
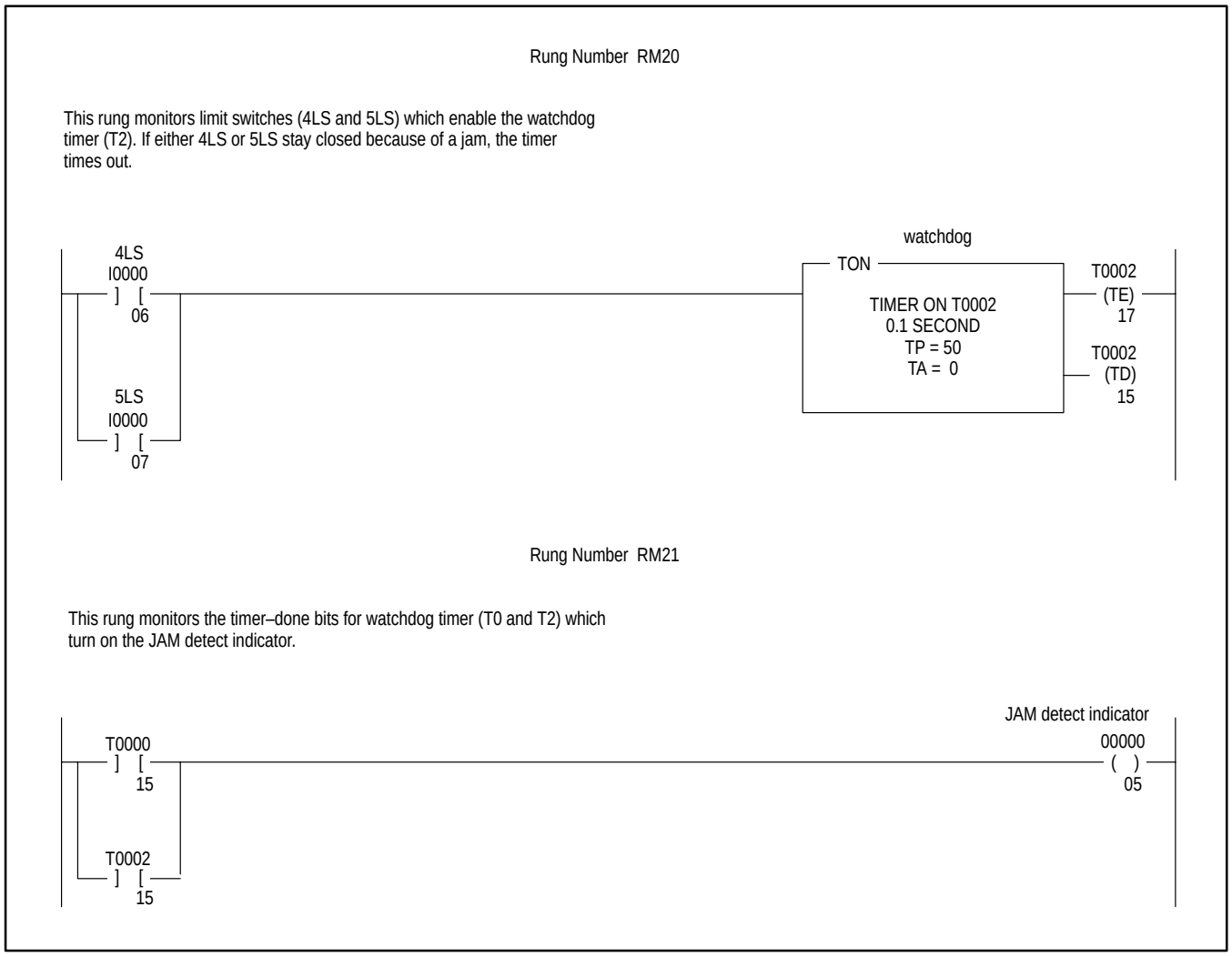


Figure 17.2
Example Ladder Program for the Conveyor Belt (continued)



Appendix A

Instruction Set Execution Times and Memory Usage

A.0 Introduction

Tables A.A to A.V give you typical times for the processor to execute the instruction set. The times given in these tables are typical times and may differ depending on your application.

Instruction type	Abbreviation	Table
start-of-rung		A.A
relay-type input	XIC, XIO	A.B
relay-type output	OTE	A.C
relay-type latch	OTL, OTU	A.D
branch	SB, BST, BND, OS, OSB	A.A
timers and counters	TON, TOF, RTO TOS, CTU, CTD	A.E A.E
reset	RES	A.E
data transfer	MOV MVM MVS	A.F A.G A.H
data comparison	EQU, NEQ, GRT GRT, GEQ LES, LEQ LIM	A.I A.J A.J A.K
arithmetic and logical	ADD, SUB MUL DIV SQR NEG AND, OR, XOR NOT	A.L A.M A.N A.O A.P A.Q A.F
file data transfer and data comparison	MVF MMF SEQ, SNE, SLS SLE, SGR, SGE	A.R, A.S A.T N/A N/A
file arithmetic and logical	ADF, SBF, MLF DVF, SQF, NGF ANF, ORF, XOF NTF	N/A N/A A.U N/A
bit shift register	BSL, BSR	N/A

Instruction type	Abbreviation	Table
FIFO register	FFL, FFU	N/A
indexed logic	XIN, XIF, BIN BIS, BIR	N/A N/A
diagnostic	DDT, FBC	N/A
program control	JMP, LBL, JSR RET, NOP MCR, END	A.V A.V N/A
block transfer and message	BTR, BTW, MSG	application dependent

N/A = Times not available when this manual was printed.

Table A.W gives you the number of words that the processor uses to store each instruction.

Table A.A
Execution Times for Start-of-rung and Branch Instructions

Instruction	Execution time in microseconds
start-of-rung	3
branch (SB, BST, OS, OSB, BND)	2

Table A.B
Execution Times for Examine-on and -off Instructions

	Execution time in microseconds for the data table sections				
Examine a bit from word	I, O	N, D, B	A, S	C, T	F
0:0 – 0:377	1	10	10	6	11
0:400 – 0:n	12	10	10	x	11
1:n – 19:n	11	11	11	x	12
20:n – n:n	20	20	16	x	21

Table A.C
Execution Times for an Output-energize Instruction

Energize a bit from word	And the rung is	Execution times in microseconds for the data table sections				
		O	I	N, D, B	A	F
0:0 – 0:377	false true	2 1	2 1	11 10	11 10	12 11
0:400 – 0:n	false true	12 12	12 11	11 10	11 10	12 11
1:n – 19:n	false true	12 11	12 11	11 11	11 11	12 12
20:n – n:n	false true	21 20	21 20	21 20	17 16	22 21

Table A.D
Execution times for Output-latch and -unlatch Instructions

Latch or unlatch a bit from word	And the rung is	Execution times in microseconds for the data table sections					
		O	I	N, D, B	A	F	C
0:0 – 0:377	false true	2 1	2 1	10 10	10 10	11 11	11 11
0:400 – 0:n	false true	12 12	11 11	10 10	10 10	11 11	x x
1:n – 19:n	false true	11 11	11 11	11 11	11 11	11 12	x x
20:n – n:n	false true	20 20	20 20	20 20	16 16	21 21	x x

Table A.E
Execution Times for Timer and Counter Instructions

Instruction	And the rung is	Execution times in microseconds for the time base or counter		
		.01, 0.1, 1.0	CPU scan	counter
Timer 0 to n	true false	10 8	18 8	x x
Counter 0 to n	true false	x x	x x	17 7
Reset	true false	10 3	10 3	10 3

Table A.F
Execution Times for Move and Logical-NOT Instructions

Move or Not word	And the rung is	Execution times in microseconds for the data table sections					
		O to O	I to I	I to O	I to B I to D I to N O to B	B to B D to D	F to F
0:0 – 0:1	true	22	20	22	22	25	32
0:400 – 0:n	true	29	27	28	26	25	32
1:n – 19:n	true	26	26	26	26	26	33
20:n – n:n	true	41	41	41	45	45	51

If the rung is false, the processor scans the rung in three microseconds.

Table A.G
Execution Times for a Move-with-mask Instruction

Move word	And the rung is	Execution times in microseconds for the data table sections			
		O, B to B	O, O to B	N, N to N	F, F to F
0:0 through 0:1 to 0:2	true	62	60	64	78
0:400 through 0:n to 0:(n+1)	true	65	67	64	78
1:n through 19:n to 19:(n+1)	true	66	66	65	84
20:n through n:n to n:(n+1)	true	94	90	94	107

If the rung is false, the processor scans the rung in three microseconds.

Table A.H
Execution Times for a Move-status Instruction

Move	And the rung is	Execution times in milliseconds for data table sections	
		to O or I	to F or B
system status to 0:1	true	0.9	1.5
module status to 0:1	true	1.2	1.7
system status to 20:1	true	1.6	1.5
module status to 20:1	true	1.8	1.7

If the rung is false, the processor scans the rung in three microseconds.

Table A.I
Execution Times for Equal-to and Not-equal-to Instructions

Compare words	Execution time in microseconds for the data table sections			
	O to O	N to N B to B	D to D	F to F
0:0 – 0:1	21	25	43	47[1]
0:400 – 0:n	28	25	43	47[1]
1:n – 19:n	26	26	44	48[1]
20:n – n:n	41	45	59	67[1]

[1]Add three microseconds for an equal-to instruction.

Table A.J
Execution Times for Greater-than, Greater-than-or-equal-to, Less-than, Less-than-or-equal-to Instructions

Compare words	Execution time in microseconds for the data table sections			
	O to O	N to N B to B	D to D	F to F
0:0 – 0:1	24	28	46	47[1]
0:400 – 0:n	31	28	46	47[1]
1:n – 19:n	29	29	47	48[1]
20:n – n:n	44	48	62	67[1]

[1]Add three microseconds for a greater-than-or-equal instruction or a less-than-equal instruction.

Table A.K
Execution Times for a Limit Instruction

Compare words	Execution times in milliseconds for the data table sections			
	O≤B≤B	O≤O≤B	N≤N≤N	F≤F≤F
0:0 with 0:1 and 0:2	26	24	28	50
0:400 with 0:n and 0:(n+1)	29	31	28	50
1:n with 19:n and 19:(n+1)	30	30	30	52
20:n with n:n and n:(n+1)	49	45	49	71

Table A.L
Execution Times for Add and Subtract Instructions

Manipulate words	And the rung is	Execution times in microseconds for the data table sections			
		O, O to O	N, N to N B, B to B	D, D to D	F, F to F
0:0 and 0:1 to 0:2	true	35	39	71[2]	73[2]
0:400 and 0:n to 0:(n+1)	true	44	39	71[2]	73[2]
1:n and 19:n to 19:(n+1)	true	41	40	73[2]	74[2]
20:n and n:n to n:(n+1)	true	61	69	93[2]	103[2]

[1]If the rung is false, the processor scans the rung in three microseconds.

[2]Add two microseconds for a subtract instruction.

Table A.M
Execution Times for a Multiply Instruction

Manipulate words	And the rung is	Execution times in microseconds for the data table sections			
		O, O to O	N, N to N B, B to B	D, D to D	F, F to F
0:0 and 0:1 to 0:2	true	42	45	78	92
0:400 and 0:n to 0:(n+1)	true	50	45	78	92
1:n and 19:n to 19:(n+1)	true	47	47	79	93
20:n and n:n to n:(n+1)	true	67	75	99	121

If the rung is false, the processor scans the rung in three microseconds.

Table A.N
Execution Times for a Divide Instruction

Manipulate words	And the rung is	Execution times in microseconds for the data table sections			
		O, O to O	N, N to N B, B to B	D, D to D	F, F to F
0:0 and 0:1 to 0:2	true	98	102	134	131
0:400 and 0:n to 0:(n+1)	true	107	102	134	131
1:n and 19:n to 19:(n+1)	true	103	103	135	132
20:n and n:n to n:(n+1)	true	123	131	155	160

If the rung is false, the processor scans the rung in three microseconds.

Table A.O
Execution Times for a Square-root Instruction

Square word	And the rung is	Execution times in microseconds for the data table sections				
		O to O	B to B	D to D	N to N	F to F
0:0 to 0:1	true	84	86	108	86	135
0:400 to 0:n	true	89	86	108	86	135
1:n to 19:n	true	87	86	109	87	136
20:n to n:n	true	101	105	124	106	155

If the rung is false, the processor scans the rung in three microseconds.

Table A.P
Execution Times for a Negate Instruction

Negate word	And the rung is	Execution times in microseconds for the data table sections				
		O to N	I to N	B to N	D to N	F to F
0:0 to 0:1	true	27	26	28	38	45
0:400 to 0:n	true	30	29	29	38	45
1:n to 19:n	true	30	30	39	39	46
20:n to n:n	true	49	49	48	58	65

If the rung is false, the processor scans the rung in three microseconds.

Table A.Q
Execution Times for Logical-AND, -OR, and -XOR Instructions

Manipulate words	And the rung is	Execution times in microseconds for the data table sections		
		O, O to O	D, D to D N, N to N B, B to B	F, F to F
0:0 and 0:1 to 0:2	true	30	35	46
0:400 and 0:n to 0:(n+1)	true	40	35	46
1:n and 19:n to 19:(n+1)	true	37	36	47
20:n and n:n to n:(n+1)	true	57	65	75

If the rung is false, the processor scans the rung in three microseconds.

Table A.R
Execution Times for a File-to-file Move Instruction

Move file	And the rung is	Execution times in microseconds for the data table sections			
		O to B	O to O	N to N	F to F
0 to file 1 (ACT/WD)	true	64/26	64/26	64/27	66/33
17 to file 18	true	64/27	66/26	64/27	66/33
20 to file 21	true	83/27	79/27	84/26	86/32

If the rung is false, the processor scans the rung in three microseconds.

ACT = time for the processor to activate the file-to-file move
WD = time for the processor to move each word in the file

Table A.S
Execution Times for a Word-to-file Move Instruction

Move word	And the rung is	Execution times in microseconds for the data table sections			
		O to B	O to O	N to N	F to F
0:0 to file 1 (ACT/WD)	true	62/21	62/21	64/21	71/22
17:0 to file 18	true	65/21	65/21	66/20	70/23
20:0 to file 21	true	85/20	81/20	85/20	90/22

If the rung is false, the processor scans the rung in 26 microseconds.

ACT = time for the processor to activate the word-to-file move
WD = time for the processor to move each word

Table A.T
Execution Times for a File-move-with-mask Instruction

Move file	And the rung is	Execution times in microseconds for the data table sections			
		O, B to B	O, O to B	N, N to N	F, F to F
0 through file 1 to file 2 (ACT/WD)	true	80/36	80/36	80/37	86/48
17 through file 18 to file 19	true	82/36	82/36	80/37	86/48
20 through file 21 to file 22	true	106/36	102/36	104/37	113/49

If the rung is false, the processor scans the rung in 26 microseconds.

ACT = time for the processor to activate the move-file-with-mask
WD = time for the processor to move each word.

Table A.U
Execution Times for File-AND, -OR, and -XOR Instructions

Manipulate files	And the rung is	Execution times in microseconds for the data table sections			
		O, B to B	O, O to B	N, N to N	F, F to F
0 and 1 to file 2 (ACT/WD)	true	80/34	82/33	82/34	85/44
17 and 18 to file 19	true	82/34	82/34	82/34	85/44
20 and 21 to file 22	true	106/34	102/34	106/34	108/45

If the rung is false, the processor scans the rung in 26 microseconds.

ACT = time for the processor to activate the file instruction
WD = time for the processor to manipulate each word.

Table A.V
Execution Times for Program Control Instructions

Instruction	And the rung is	Execution times in microseconds
Jump to label (JMP)	true false	6 2
Label w/o comment (LBL) Label w/ comment (LBL)	true or false true or false	2 3
Jump to subroutine (JSR)	true false	6 2
Return (RET) No operation (NOP)	true or false true or false	5 1

Table A.W
Memory Usage for the Instructions Set

Instruction type	Abbreviation	Address	Words
relay-type input	XIC XIO	bit in I or O word < 377 bit in timer or counter word bit in file 0, word < 1024 bit in any other file/word	1 2 2 3
relay-type output	OTE OTL OTU	bit in I or O, word < 377 bit in file 0, word < 1024 bit in any other file/word	1 2 3
branch	SB BST OS OSB BND		1 1 1 1 1
timers	TON TOF RTO TOS	timer number < 64 any other timer number	1 2
counters	CTU CTD	counter number < 64 any other counter number	1 2
reset retentive timer or counter	RES	timer or counter < 64 any other timer or counter	1 2
data transfer	MOV	word in file 0, word < 1024 wod in any other file/word	3 5
	MVM	word in file 0, word < 1024 word in any other file/word	4 7
	MVS	word to and from system status	11-20
data comparison	EQU NEQ GRT GEQ LES LEQ	both sources from file 0, word < 1024 sources from any other file/word	3 5
	LIM	all sources from file 0, word < 1024 sources from any other file/word	4 7
arithmetic and logical	ADD SUB MUL DIV SQR NEG AND OR XOR	both source from file 0, word < 1024 sources from any other file/wod	4 7
	NOT	both sources from file 0, word < 1024 sources from any other file/word	3 5

Instruction type	Abbreviation	Address	Words
file data trasfer, and data comparison	MVF MMF SEQ SNE SLS SLE SGR SGE	both sources form file 0, word < 1024 souces from any other file/word	4 7
file arithmetic, and logical	ADF SBF MLF DVF SQF NGF ANF ORF XOF NTF	both sources from file 0, word < 1024 sources from any other file/word	5 9
bit shift register	BSL BSR	both sources from file 0, word < 1024 sources from any other file/word	6 10
FIFO register	FFL FFU	both sources from file 0, word < 1024 sources from any other file/word	5 7
indexed logic	XIN XIF BIN BIS BIR	bit in file 0, word < 1024 bit in any other file/word	3 4
diagnostic	DDT FBC	both sources from file 0, word < 1024 sources from any other file/word	5 9
program control	MCR JMP LBL JSR RET END NOP		1 1-2 2-3 1-2 1 1 1
block transfer	BTR BTW		3 3
message	MSG		3

Appendix B

Numbering Systems

B.0 Introduction

In general, PLC–3 family controllers store binary data (1s and 0s) in 16–bit words. You can interpret this data in a number of different ways depending on your application needs.

PLC–3 family processors can use the following number systems:

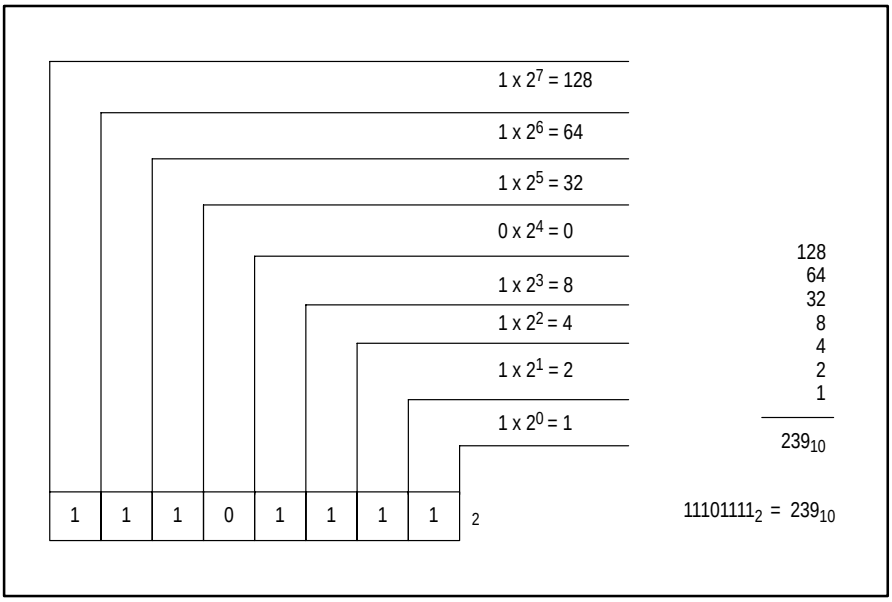
- binary
- decimal
- binary coded decimal
- hexadecimal
- octal
- integer
- floating point

We describe these numbering systems in the following sections.

B.1 Binary

The binary numbering system uses a number set that includes two digits: 0 and 1. Each digit in a binary number has a certain place value expressed as a power of two. You can compute the decimal equivalent of a binary number by multiplying each binary digit by its corresponding place value and adding these numbers together (Figure B.1).

Figure B.1
Determining the Value of a Binary Number

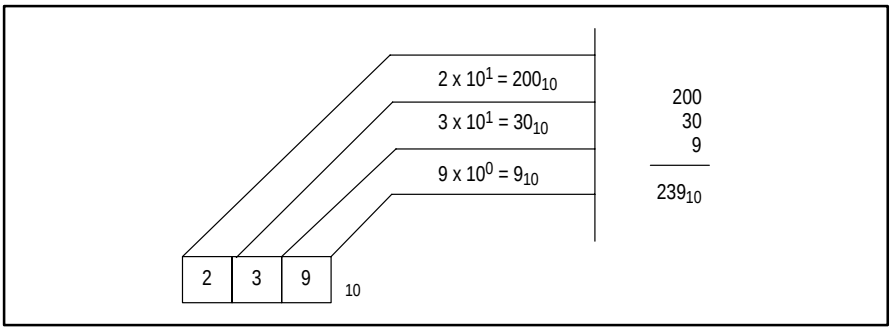


B.2 Decimal

Most of us use the decimal numbering system every day. Examples of its use include the metric system of measurement and the U.S. monetary system. The decimal numbering system uses a number set that includes ten digits: 0 through 9. The value of a decimal number depends on the digits used and the place value of each digit.

Each place value in a decimal number represents a power of ten starting with 10^0. You can compute the value of a decimal number by multiplying each digit by its corresponding place value and adding these numbers together (Figure B.2).

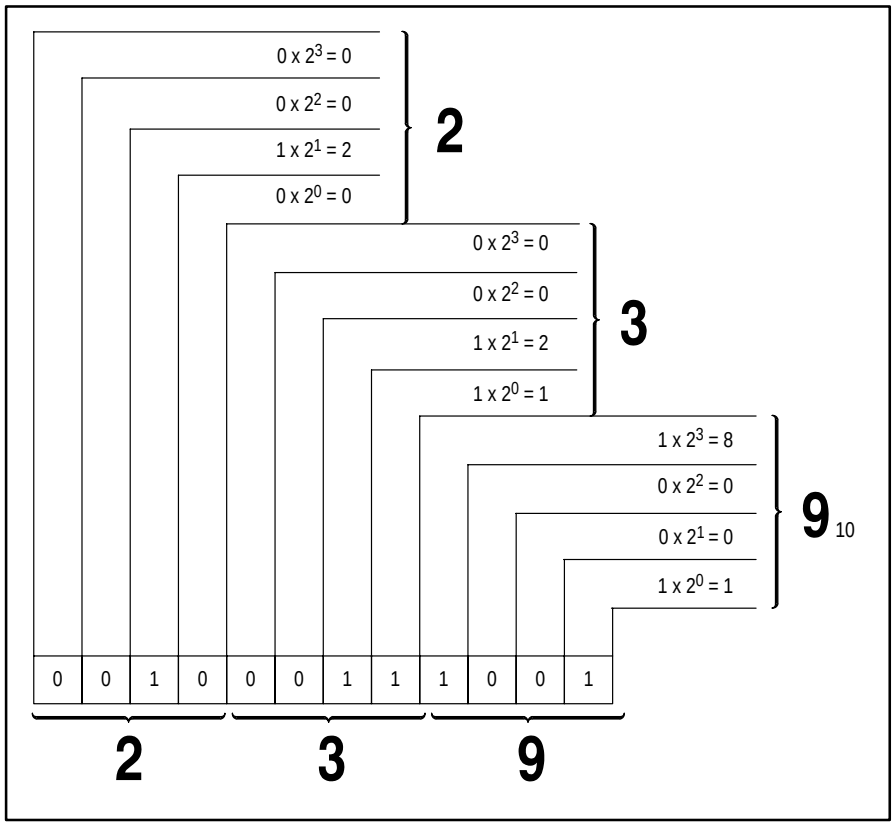
Figure B.2
Determining the Value of a Decimal Number



B.3 Binary Coded Decimal

In storing decimal numbers in memory, the processor uses the binary coded decimal (BCD) form. In BCD, each group of four binary digits (bits) represents a decimal number between 0 and 9. Thus, each 16–bit word in the decimal section represents a decimal value between 0 and 9,999 (Figure B.3).

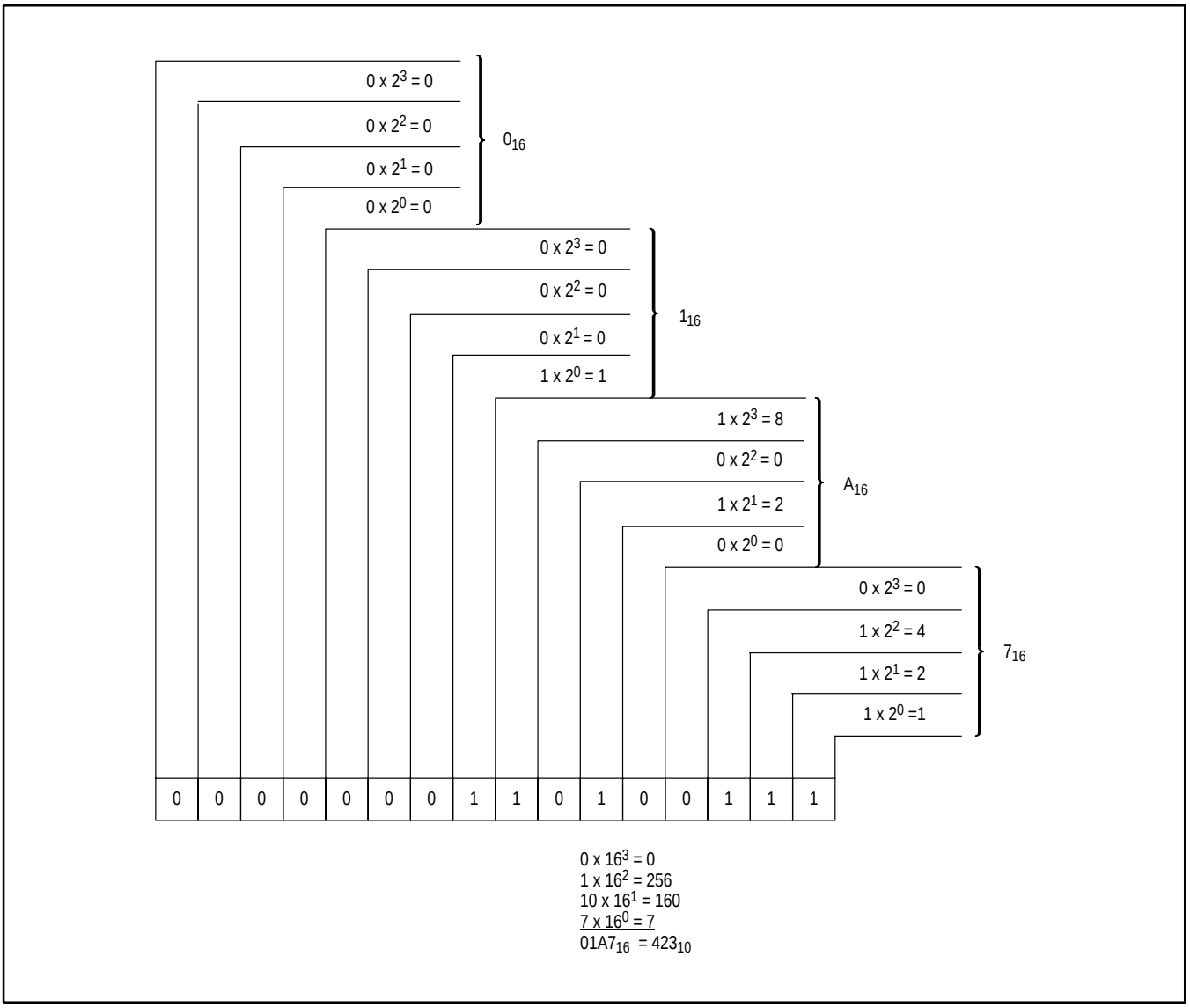
Figure B.3
Determining the Value of a Binary-Coded-Decimal Number



B.4 Hexadecimal

The hexadecimal numbering system has a number set of 16 digits: 0 through 9 and the letters A through F. The letters A through F represent the decimal numbers 10 through 15 respectively. Each place value of a hexadecimal number represents a power of sixteen. You can convert a hexadecimal number to a decimal number by multiplying the hexadecimal digit by its corresponding place value and add these values together (Figure B.4).

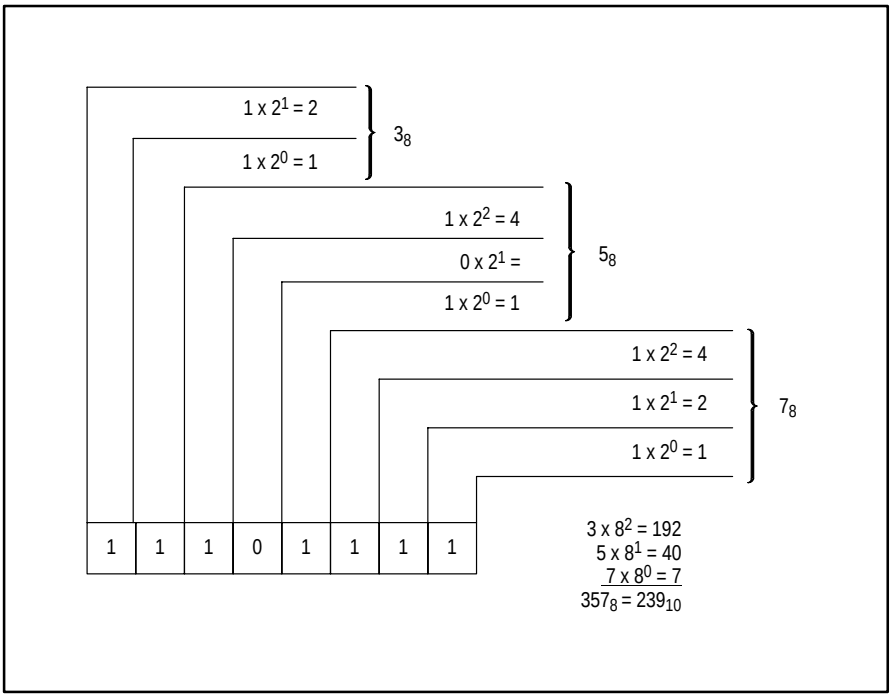
Figure B.4
Determining the Value of a Hexadecimal Number



B.5 Octal

The octal numbering system uses a number set that includes eight digits: 0 through 7. Each place value of an octal number represent a power of eight. You can compute the decimal value of an octal number by multiplying each octal digit by its place value and add these values together (Figure B.5).

Figure B.5
Determining the Value of a Octal Number



B.6 Integer

The controller uses three types of integers: signed, unsigned, and high order. These integer types are similar to the decimal numbering system with the following differences:

- storage methods
- acceptable ranges

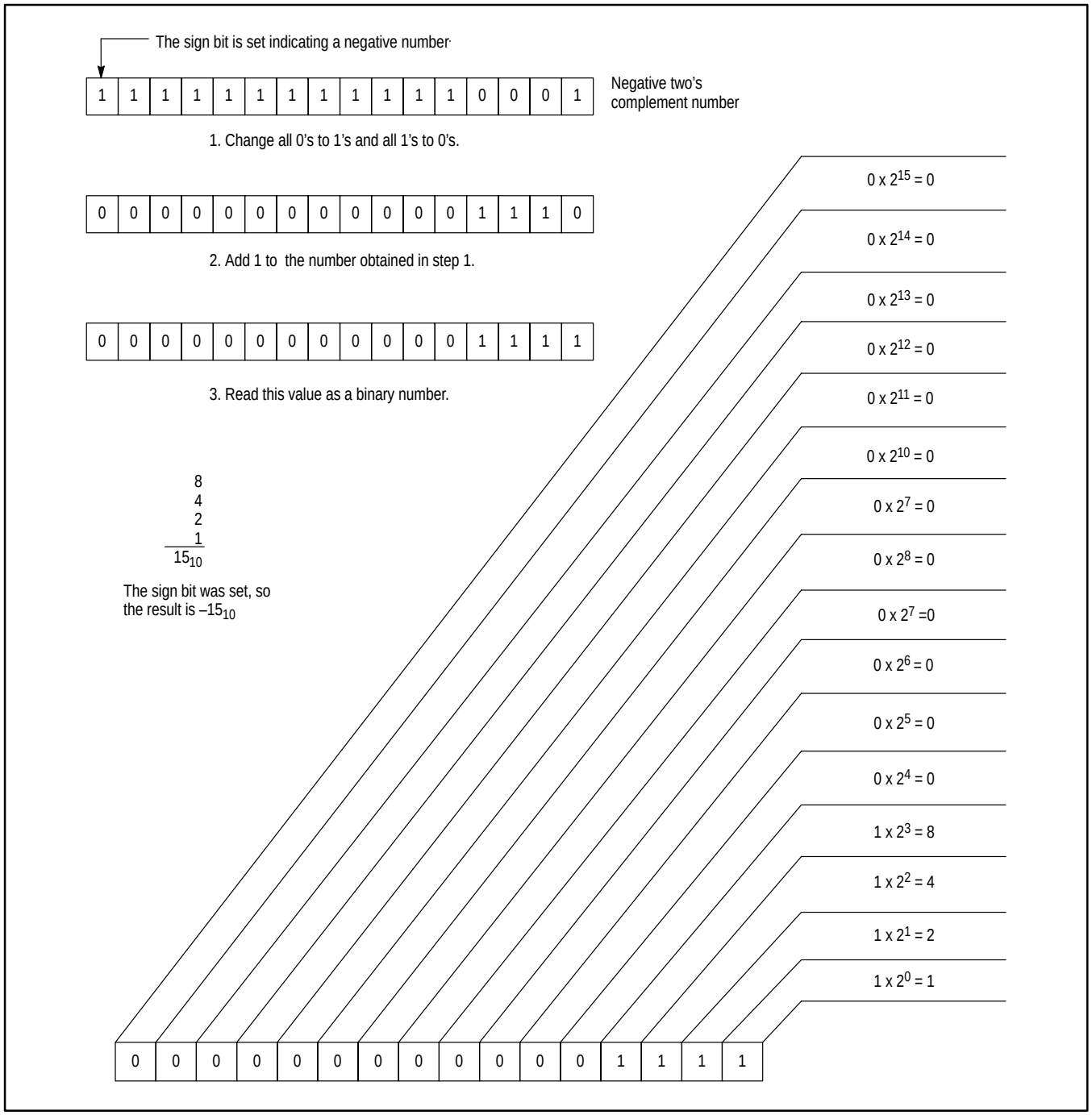
Integer type	Acceptable range	Stored in
signed	-32,768 through 32,767	two's complement in one 16-bit word
unsigned	0 through 65,535	binary one 16-bit word
high order	-2,147,483,648 through +2,147,483,647	two's complement in two 16-bit words

Numbers stored in two's complement form use the most significant bit as a sign bit:

If the sign bit is	Then the number is
set	negative
reset	positive

Read the magnitude of positive numbers stored in two's complement form in the same manner as binary numbers (Figure B.6).

Figure B.6
Two's Complement Conversion for a Negative Number



B.7 Floating Point

Floating point numbers provide eight–digit precision and can range from $\pm 2.939 \times 10^{-39}$ to $\pm 1.701 \times 10^{38}$. Each value requires two 16–bit words in memory.

B.8 Using the Conversion Tables

Table B.A is the Decimal–Hexadecimal–Octal–ASCII conversion table that converts an ASCII bit pattern to its decimal, hexadecimal, and octal equivalents. The table is divided into four columns:

Table B.A
Decimal/Hexadecimal/Octal/ASCII Conversion Table

Column 1				Column 2				Column 3				Column 4			
DEC	**HEX**	**OCT**	**ASC**	**DEC**	**HEX**	**OCT**	**ASC**	**DEC**	**HEX**	**OCT**	**ASC**	**DEC**	**HEX**	**OCT**	**ASC**
00	00	000	NUL	32	20	040	SP	64	40	100	@	96	60	140	\
01	01	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
02	02	002	STX	34	22	042	“	66	42	102	B	98	62	142	b
03	03	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
04	04	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
05	05	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
06	06	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
07	07	007	BEL	39	27	047	‘	71	47	107	G	103	67	147	g
08	08	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
09	09	011	HT	41	29	051	)	73	49	111	I	105	69	151	i
10	0A	012	LF	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	0B	013	VT	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	0C	014	FF	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	0D	015	CR	45	2D	055	–	77	4D	115	M	109	6D	155	m
14	0E	016	S0	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	0F	017	SI	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC	59	3B	073	;	91	5B	133	[	123	7B	173	{
28	1C	034	FS	60	3C	074	<	92	5C	134	\	124	7C	174	|
29	1D	035	GS	61	3D	075	=	93	5D	135	]	125	7D	175	}
30	1E	036	RS	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	US	63	3F	077	?	95	5F	137	–	127	7F	177	DEL

- Column 1 contains all the control characters.
- Column 2 contains numbers and symbols.
- Column 3 contains the capital letters. If you press the control key [CTRL] and a capital letter, the control code matches the control character in the first column. For example [CTRL] G is the control character [BEL].
- Column 4 contains lower case letters and symbols.

In addition, Table B.B gives you the binary bit patterns for hexadecimal values.

Table B.B
Binary Patterns for Hexadecimal Digits

Hexadecimal Digits	Binary Equivalent	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Appendix C

Memory Management Forms

C.0 Introduction

You can use the following memory management forms to document memory requirements for your controller.

Data Table Word Map
(1,000 Word)

PAGE _____ OF _____

ADDRESS _____ TO _____

PROJECT NAME _____ PROCESSOR _____

DESIGNER _____ SECTION _____

WORD ADDRESS	FROM ——— (30 WORDS) ——→ TO	WORD ADDRESS	REF.
☐ 000		☐ 029	
030		059	
060		089	
090		119	
120		149	
150		179	
180		209	
210		239	
240		269	
270		299	
300		329	
330		359	
360		389	
390		419	
420		449	
450		479	
480		509	
510		539	
540		569	
570		599	
600		629	
630		659	
610		689	
640		719	
720		749	
750		779	
780		809	
810		839	
840		869	
870		899	
900		929	
930		959	
960		989	
990		999	

PAGE _____ OF _____

Data Table word Assignments (100 Decimal Words or 64 Octal Words)

ADDRESS _____ TO _____

PROJECT NAME _____ PROCESSOR _____

DESIGNER _____ SECTION _____

WORD ADDR	DESCRIPTION	WORD ADDR	DESCRIPTION
0		0	
1		1	
2		2	
3		3	
4		4	
5		5	
6		6	
7		7	
8		8	
9		9	
0		0	
1		1	
2		2	
3		3	
4		4	
5		5	
6		6	
7		7	
8		8	
9		9	
0		0	
1		1	
2		2	
3		3	
4		4	
5		5	
6		6	
7		7	
8		8	
9		9	
0		0	
1		1	
2		2	
3		3	
4		4	
5		5	
6		6	
7		7	
8		8	
9		9	

NOTE: Use the shaded areas only for words numbered in decimal. Skip shaded ares for words numbered in octal (I/O sections).

Comments _____

I/O Section Word Map
(1,024)

PAGE _____ OF _____

ADDRESS _____ TO _____

PROJECT NAME _____ PROCESSOR _____

DESIGNER _____ SECTION _____

WORD ADDRESS	FROM ——— (32 WORDS) ——— TO	WORD ADDRESS	REF.
☐ 000		☐ 037	
040		077	
100		137	
140		177	
200		237	
240		277	
300		337	
340		377	
400		437	
440		477	
500		537	
540		577	
600		637	
640		677	
700		737	
740		777	
☐ 000		☐ 037	
040		077	
100		137	
140		177	
200		237	
240		277	
300		337	
340		377	
400		437	
440		477	
500		537	
540		577	
600		637	
640		677	
700		737	
740		777	

Data Table Bit Assignments

PAGE _____ OF _____

ADDRESS _____ TO _____

PROJECT NAME _____ PROCESSOR _____

DESIGNER _____ SECTION _____

WORD	BIT	DESCRIPTION	WORD	BIT	DESCRIPTION
	0 0			0 0	
	0 1			0 1	
	0 2			0 2	
	0 3			0 3	
	0 4			0 4	
	0 5			0 5	
	0 6			0 6	
	0 7			0 7	
	1 0			1 0	
	1 1			1 1	
	1 2			1 2	
	1 3			1 3	
	1 4			1 4	
	1 5			1 5	
	1 6			1 6	
	1 7			1 7	
	0 0			0 0	
	0 1			0 1	
	0 2			0 2	
	0 3			0 3	
	0 4			0 4	
	0 5			0 5	
	0 6			0 6	
	0 7			0 7	
	1 0			1 0	
	1 1			1 1	
	1 2			1 2	
	1 3			1 3	
	1 4			1 4	
	1 5			1 5	
	1 6			1 6	
	1 7			1 7	

Comments _____

Data Table Bit Assignments

PAGE _____ OF _____

ADDRESS _____ TO _____

PROJECT NAME _____ PROCESSOR _____

DESIGNER _____ SECTION _____

WORD	BIT	DESCRIPTION	WORD	BIT	DESCRIPTION
	0 0			0 0	
	0 1			0 1	
	0 2			0 2	
	0 3			0 3	
	0 4			0 4	
	0 5			0 5	
	0 6			0 6	
	0 7			0 7	
	1 0			1 0	
	1 1			1 1	
	1 2			1 2	
	1 3			1 3	
	1 4			1 4	
	1 5			1 5	
	1 6			1 6	
	1 7			1 7	
	0 0			0 0	
	0 1			0 1	
	0 2			0 2	
	0 3			0 3	
	0 4			0 4	
	0 5			0 5	
	0 6			0 6	
	0 7			0 7	
	1 0			1 0	
	1 1			1 1	
	1 2			1 2	
	1 3			1 3	
	1 4			1 4	
	1 5			1 5	
	1 6			1 6	
	1 7			1 7	

Comments _____

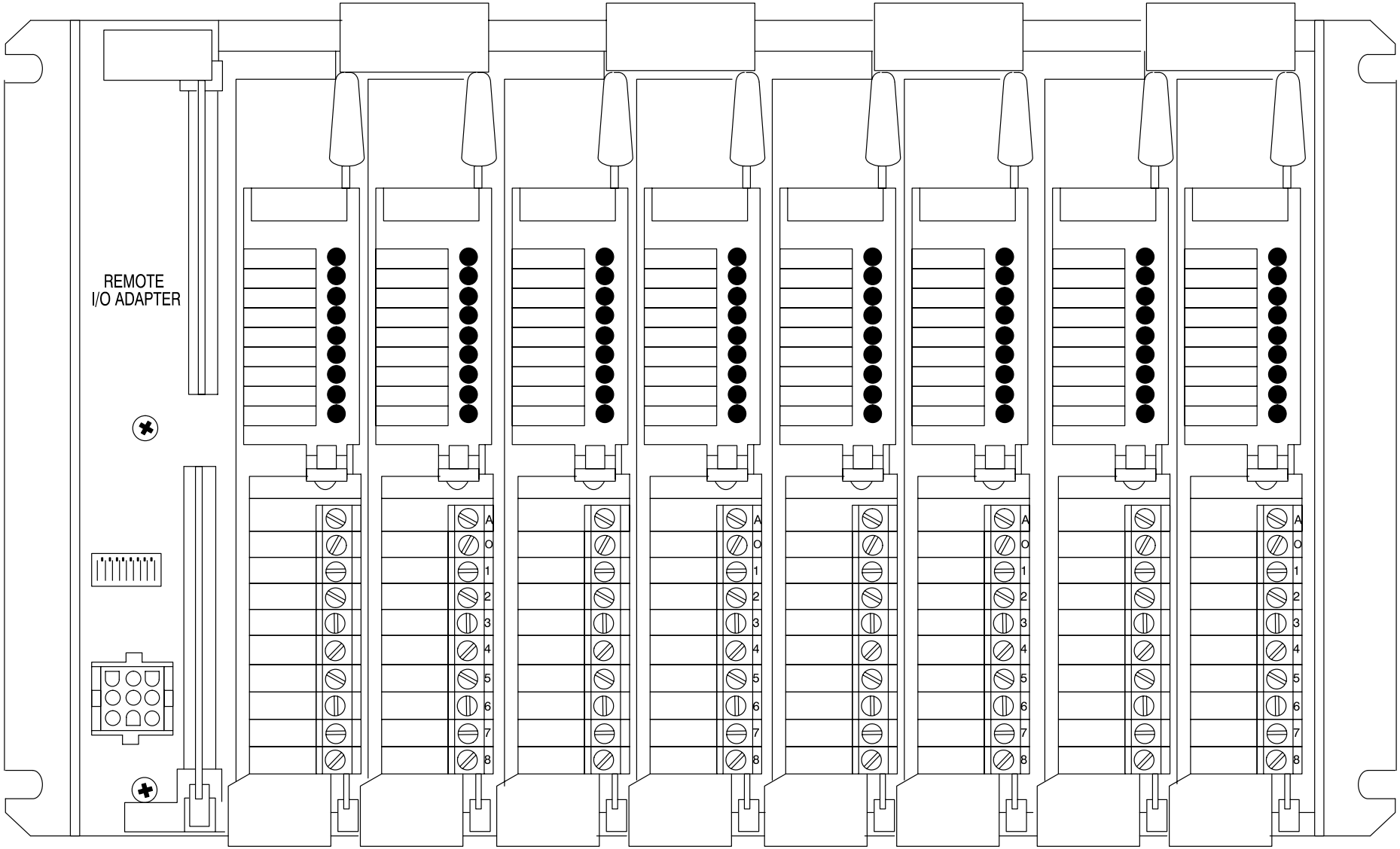
Connection Diagram Addressing for Standard-Density 1771 I/O Modules

PAGE _____ OF _____

DATE _____

DESIGNER _____

PROJECT NAME _____



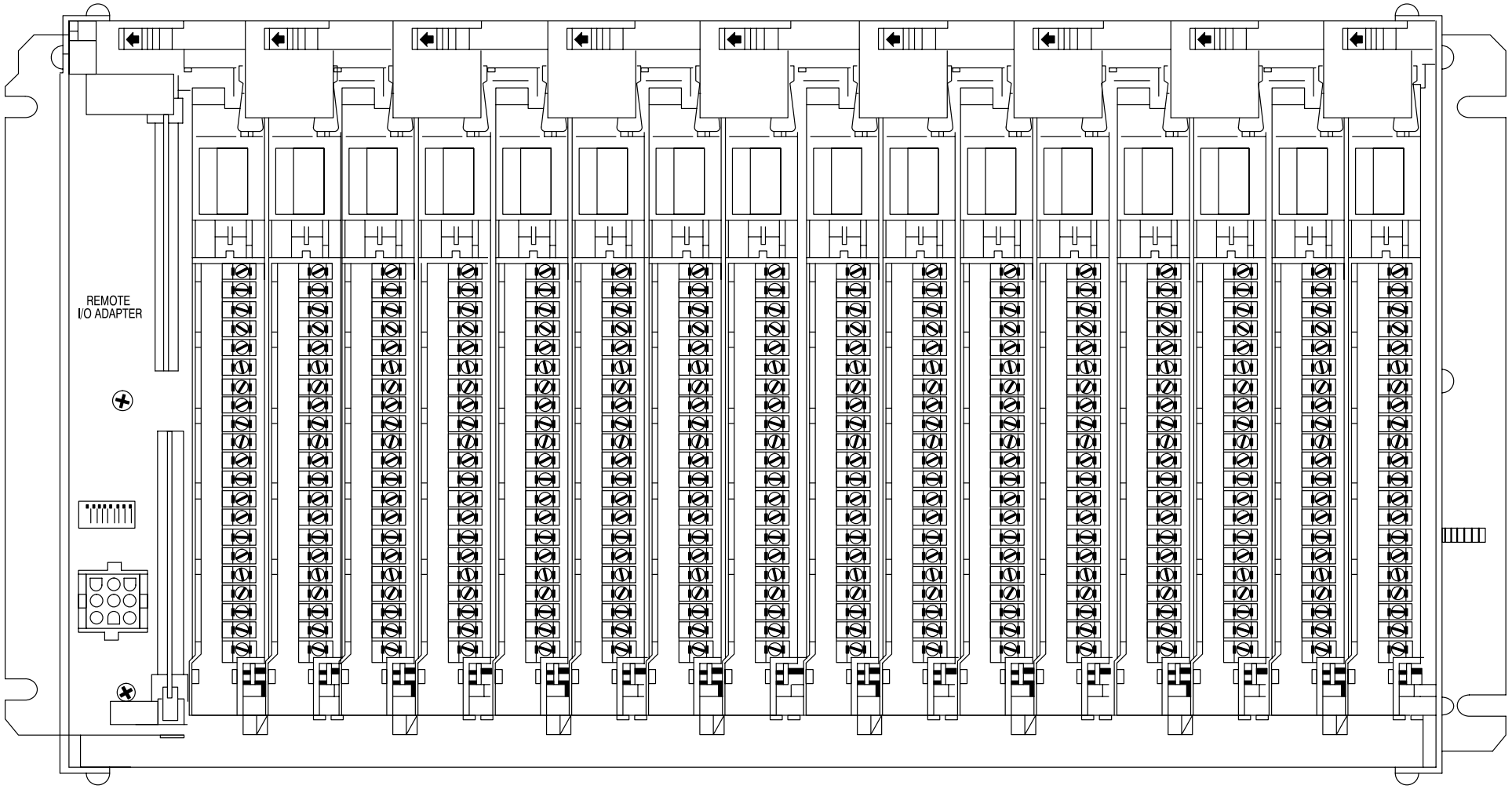
Connection Diagram Addressing for High-Density 1771 I/O Modules

PAGE _____ OF _____

DATE _____

DESIGNER _____

PROJECT NAME _____



Appendix D

Using the Instruction Set

D.0 Introduction

Tables D.A through D.H list the total instruction set available for the controller. For each instruction, we list its name, code, and compatible processor modules.

Table D.A
Relay-type and Branching Instructions

If you want to program	Use this code	Compatible with 1775-L1	L2, L3, L4
examine on	XIC	x	x
examine off	XIO	x	x
output energize	OTE	x	x
input start branch	SB	x	x
input branch start – nested	BST	x	x
output start branch	OS	x	x
output branch start – nested	OSB	x	x
branch end	BND	x	x
output latch	OTL	x	x
output unlatch	OTU	x	x

Table D.B
Timer and Counter Instructions

If you want to program	Use this code	Compatible with 1775-L1	L2, L3, L4
timer on-delay	TON	x	x
timer off-delay	TOF	x	x
retentive timer on-delay	RTO	x	x
timer one-shot	TOS	x	x
counter up	CTU	x	x
counter down	CTD	x	x
reset timer/counter	RES	x	x

Table D.C
Data Manipulation Instructions

If you want to program	Use this code	Compatible with 1775-L1	L2, L3, L4
move	MOV	x	x
move with mask	MVM	x	x
move status	MVS	x	x
equal to	EQU	x	x
not equal to	NEQ	x	x
greater than	GRT	x	x
greater than or equal to	GEQ	x	x
less than	LES	x	x
less than or equal to	LEQ	x	x
limit	LIM	x	x
add	ADD	x	x
subtract	SUB	x	x
multiply	MUL	x	x
divide	DIV	x	x
square root	SQR	x	x
negate	NEG	x	x
and	AND	x	x
or	OR	x	x
exclusive or	XOR	x	x
not	NOT	x	x

Table D.D
Data Manipulation Instructions for Files

If you want to program	Use this code	Compatible with 1775-L1	L2, L3, L4
file move	MVF	x	x
file move with mask	MMF	x	x
search equal	SEQ	x	x
search not equal	SNE	x	x
search less than	SLS		x
search less than or equal to	SLE		x
search greater than	SGR		x
search greater than or equal to	SGE		x
file add	ADF		x
file subtract	SBF		x
file multiply	MLF		x
file divide	DVF		x
file square root	SQF		x
file negate	NGF		x
file and	ANF		x
file or	ORF		x
file exclusive or	XOF		x
file not	NTF		x

Table D.E
Shift Register and Indexed Logic Instructions

If you want to program	Use this code	Compatible with 1775-L1	L2, L3, L4
bit shift left	BSL	x	x
bit shift right	BSR	x	x
examine indexed bit on	XIN		x
examine indexed bit off	XIF		x
indexed bit on	BIN		x
latch (set) indexed bit	BIS		x
unlatch (reset) indexed bit	BIR		x

Table D.F
FIFO Register and Diagnostic Instructions

If you want to program	Use this code	Compatible with 1775-L1	L2, L3, L4
FIFO load	FFL		x
FIFO unload	FFU		x
diagnostic detect	DDT	x	x
file bit compare	FBC	x	x

Table D.G
Program Control Instructions

If you want to program	Use this code	Compatible with 1775-L1	L2, L3, L4
master control reset	MCR	x	x
jump to label	JMP	x	x
label	LBL	x	x
jump to subroutine	JSR	x	x
return	RET	x	x
end	END	x	x
no operation	NOP	x	x

Table D.H
Block-transfer and Message Instructions

If you want to program	Use this code	Compatible with 1775-L1	L2, L3, L4
block transfer read	BTR	x	x
block transfer write	BTW	x	x
message	MSG	x	x

Numbers

A

B

C

D

E

F

ALLEN-BRADLEY
A ROCKWELL INTERNATIONAL COMPANY

As a subsidiary of Rockwell International, one of the world's largest technology companies — Allen-Bradley meets today's challenges of industrial automation with over 85 years of practical plant-floor experience. More than 13,000 employees throughout the world design, manufacture and apply a wide range of control and automation products and supporting services to help our customers continuously improve quality, productivity and time to market. These products and services not only control individual machines but integrate the manufacturing process, while providing access to vital plant floor data that can be used to support decision-making throughout the enterprise.

With offices in major cities worldwide

WORLD HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (414) 382-2000
Telex: 43 11 016
FAX: (414) 382-4444

EUROPE/MIDDLE EAST/AFRICA HEADQUARTERS
Allen-Bradley Europa B.V.
Amsterdamseweg 15
1422 AC Uithoorn
The Netherlands
Tel: (31) 2975/60611
Telex: (844) 18042
FAX: (31) 2975/60222

ASIA/PACIFIC HEADQUARTERS
Allen-Bradley (Hong Kong) Limited
Room 1006, Block B, Sea View Estate
28 Watson Road
Hong Kong
Tel: (852) 887-4788
Telex: (780) 64347
FAX: (852) 510-9436

CANADA HEADQUARTERS
Allen-Bradley Canada Limited
135 Dundas Street
Cambridge, Ontario N1R 5X1
Canada
Tel: (519) 623-1810
FAX: (519) 623-8930

LATIN AMERICA HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (414) 382-2000
Telex: 43 11 016
FAX: (414) 382-2400

Publication 1775-6.4.1 – October, 1992
Supersedes Publication 1775-6.4.1 – October, 1987

PN 955103-58