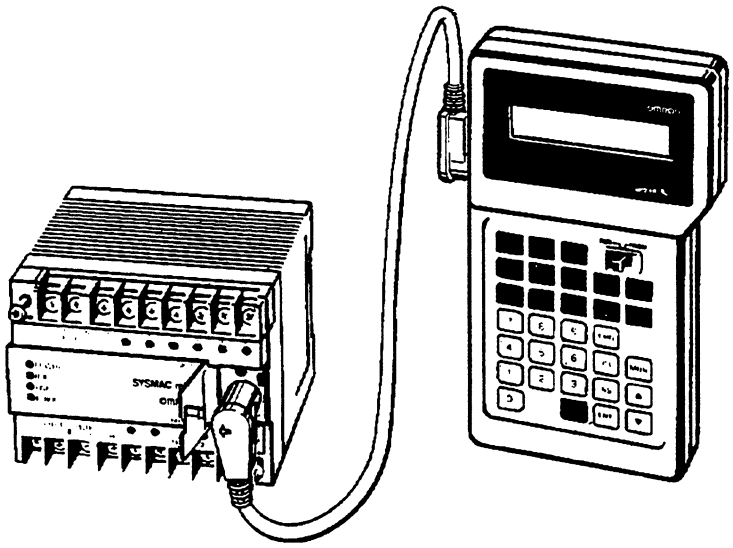
SYSMAC Mini Programmable Controllers
SP10, SP16, SP20
Operation Manual

Revised October 1992



Notice:

OMRON products are manufactured for use according to proper procedures by a qualified operator and only for the purposes described in this manual.

The following conventions are used to indicate and classify warnings in this manual. Always heed the information provided with them.

DANGER! Indicates information that, if not heeded, could result in loss of life or serious injury.

Caution Indicates information that, if not heeded, could result in minor injury or damage to the product.

OMRON Product References

All OMRON products are capitalized in this manual. The word "Unit" is also capitalized when it refers to an OMRON product, regardless of whether or not it appears in the proper name of the product.

The abbreviation "Ch," which appears in some displays and on some OMRON products, means "word" and is abbreviated "Wd" in documentation.

The abbreviation "PC" means Programmable Controller and is not used as an abbreviation for anything else.

Visual Aids

The following headings appear in the left column of the manual to help you locate different types of information.

Note Indicates information of particular interest for efficient and convenient operation of the product.

1, 2, 3... Indicates lists of one sort or another, such as procedures, precautions, etc.

TABLE OF CONTENTS

About this Manual:

This manual describes the installation and operation of the SYSMAC mini Programmable Controllers and includes the sections described below. The SYSMAC mini PCs include the SP10, SP16, and the SP20 and are called SP-series PCs in this manual. Please read this manual completely and be sure you understand the information provided before attempting to install and operation any of the SP-series PCs.

Section 1 Introduction explains the background and some of the terms used in ladder-diagram programming. It also provides an overview of the process of programming and operating a PC and explains basic terminology used with OMRON PCs. Descriptions of the features of the SP-series PCs and Units that comprise SP-series systems are also provided.

Section 2 Installation provides details on the installation environment and the wiring of the PC. The dimensions of all components are also presented.

Section 3 Programming describes information necessary for programming SP-series PCs. The first five subsections provide enough information to enable you to write, input, and execute a basic ladder-diagram program. The remainder of this section provides more advanced programming information, with *3–7* describing individually each instruction in the SP-series instruction set.

Section 4 Operation provides further information on operating SP-series PCs via the Programming Console, such as monitoring, data modification, and Memory Card operations.

Section 5 Troubleshooting provides information on error indications. Information in this section is also necessary when debugging a program.

The appendices provide tables of standard OMRON products available for the SP-series PCs, specifications, reference tables of instructions and Programming Console operations, and error and arithmetic flag operation. Also provided are several programming and data area assignment sheets that can be copied out of the manual and used in developing programs.

SECTION 1
Introduction

This section will introduce you to Programmable Controllers in general and specifically to the SP-series PCs and the various Units available for use with them. It also describes the configurations possible with the SP-series PCs and how to connect these configurations. Detailed wiring and installation procedures are provided in *Section 2 Installation*.

1–1 Features

Miniature High-performance The SP-series PCs are extremely compact yet have a programming capacity of about 100 instructions in the SP10 or about 240 instructions in the SP16 and SP20. The SP10 is equipped with 34 different instructions and the SP16 and SP20 are equipped with 38 instructions. With real programming capability in such a small package, these compact PCs are ideal for mounting in a control box or in the device being controlled.

High-speed Processing The minimum instruction execution time is as short as 0.2 microseconds. The input delay is only 500 microseconds.

Low Maintenance The user program is automatically transferred from RAM to EEPROM, eliminating the need to back up memory, which can be rewritten up to 5,000 times.

Input Signal Filter To prevent errors due to chattering or external noises on input signals, the input circuits are provided with filter timers that can be set to 0, 1, 5, or 10 ms.

Efficient Distributed Control Up to four SP-series PCs can be connected with a Link Adapter. A total of 128 points can be linked between the PCs, which means that up to 32 points can be processed by one SP-series PCs. Each PC still operates according to its own program.

Easy-to-use Analog Timers One analog timer is provided with the SP10 and two analog timers are provided with the SP16 and SP20. The set time of these analog timers can be changed even while the PC is operating, with adjustment screws located inside the front cover.

Reversible Drum Counter A reversible drum counter can be programmed for various counter present value ranges.

Step Instructions Up to five steps (four processes) of instructions can be created, making it easy to program start-stop control.

Shift Register A 16-bit shift register can be used to control various operations easily.

Arithmetic/Logical Instructions Addition, subtraction, ANDs, and ORs can be performed on16-bit data.

Differentiated Instructions Up to 16 rising edge/falling edge differentiated instructions can be programmed.

1–2 PC Basics

A PC (Programmable Controller) is basically a CPU (Central Processing Unit) containing a program and connected to input and output (I/O) devices. The program controls the PC so that when an input signal from an input device turns ON, the appropriate response is made. The response normally involves turning ON an output signal to some sort of output device. The input devices could be photoelectric sensors, pushbuttons on control panels, limit switches, or any other device that can produce a signal that can be input into the PC. The output devices could be solenoids, switches activating indicator lamps, relays turning on motors, or any other devices that can be activated by signals output from the PC.

For example, a sensor detecting a passing product turns ON an input to the PC. The PC responds by turning ON an output that activates a pusher that pushes the product onto another conveyor for further processing. Another sensor, positioned higher than the first, turns ON a different input to indicate that the product is too tall. The PC responds by turning on another pusher positioned before the pusher mentioned above to push the too-tall product into a rejection box.

Although this example involves only two inputs and two outputs, it is typical of the type of control operation that PCs can achieve. Actually even this example is much more complex than it may at first appear because of the timing that would be required, i.e., "How does the PC know when to activate each pusher?" Much more complicated operations, however, are also possible. The problem is how to get the desired control signals from available inputs at appropriate times.

To achieve proper control, the SP-series PCs use a form of PC logic called ladder-diagram programming. The next few sections will explain ladder-diagram programming and will prepare you for programming and operating the SP-series PCs.

Relay Circuits: The Roots of PC Logic

PCs historically originate in relay-based control systems. Although the integrated circuits and internal logic of the PC have taken the place of the discrete relays, timers, counters, and other such devices, actual PC operation proceeds as if those discrete devices were still in place. PC control, however, also provides computer capabilities and accuracy to achieve a great deal more flexibility and reliability than is possible with relays.

The symbols and other control concepts used to describe PC operation also come from relay-based control and form the basis of the ladder-diagram programming method. Most of the terms used to describe these symbols and concepts, however, have come from computer terminology.

Relay vs. PC Terminology

The terminology used throughout this manual is somewhat different from relay terminology, but the concepts are the same. The following table shows the relationship between relay terms and the terms used for OMRON PCs.

Relay term	PC equivalent
contact	Input or condition
Coil	Output or work bit
NO relay	Normally open (NO) condition
NC relay	Normally closed (NC) condition

The terms used for PC will be described in detail later.

1–2–1 PC Terminology

The following terms are crucial to understanding PC operation and are thus explained here. Definitions are also provided in the *Glossary* at the back of the manual.

Inputs and Outputs

A device connected to the PC that sends a signal to the PC is called an **input device**; the signal it sends is called an **input signal**. A signal enters the PC through terminals or through pins on a connector on a Unit. The place where a signal enters the PC is called an **input point**. This input point is allocated a location in memory that reflects its status, i.e., either ON or OFF. This memory location is called an **input bit**. The CPU, in its normal processing cycle, monitors the status of all input points and turns ON or OFF corresponding input bits accordingly.

There are also **output bits** in memory that are allocated to **output points** on Units through which **output signals** are sent to **output devices**, i.e., an output bit is turned ON to send a signal to an output device through an output point. The CPU periodically turns output points ON or OFF according to the status of the output bits.

These terms are used when describing different aspects of PC operation. When programming, one is concerned with what information is held in memory, and so I/O bits are referred to. When talking about the Units that connect the PC to the controlled system and the places on these Units where signals enter and leave the PC, I/O points are referred to. When wiring these I/O points, the physical counterparts of the I/O points, either terminals or connector pins, are referred to. When talking about the signals that enter or leave the PC, one refers to input signals and output signals, or sometimes just inputs and outputs. It all depends on what aspect of PC operation is being discussed.

Controlled System and Control System

The Control System includes the PC and all I/O devices it uses to control an external system. A sensor that provides information to achieve control is an input device that is clearly part of the Control System. The controlled system is the external system that is being controlled by the PC program through these I/O devices. I/O devices can sometimes be considered part of the controlled system, e.g., a motor used to drive a conveyor belt.

1–2–2 Overview of PC Operation

The following are the basic steps involved in programming and operating the SP-series PCs. Assuming you have already purchased one or more of these PCs, you must have a reasonable idea of the required information for steps one and two, which are discussed briefly below. The rest of the steps are described later in this manual.

1, 2, 3..

1. Determine what the controlled system must do, in what order, and at what times.
2. Determine what size of system is required, i.e.,will a single CPU suffice or will a Link Adapter be required to join multiple CPUs.
3. On paper, assign all input and output devices to I/O points on the CPUs and determine which I/O bits will be allocated to each. (*3–2 Memory Areas*)
4. Using relay ladder symbols, write a program that represents the sequence of required operations and their inter-relationships. Also, be sure to program appropriate responses for all possible emergency situations. (*3–4 Basic Programming*, *3–6 Advanced Programming*, and *3–7 Instruction Set)*
5. Input the program and all required data into the PC. (*3–5 Inputting the Program*)
6. Debug the program, first to eliminate any syntax errors, and then to find execution errors.(*3–8 Debugging*)
7. Wire the PC to the controlled system. (*Section 2 Installation*)
8. Test the program in an actual control situation and carry out fine tuning as required. (*Section 4 Operation)*
9. Record two copies of the finished program on masters and store them safely in different locations.(*3–5–8 Program Transfer*)

Control System Design

Designing the Control System is the first step in automating any process. A PC can be programmed and operated only after the overall Control System is understood. Designing the Control System requires, first of all, a thorough understanding of the devices that are to be controlled. The first step in de-

signing a Control System is thus determining the requirements of the controlled system.

Once the entire Control System has been designed, the task of programming, debugging, and operation as described in the remaining sections of this manual can begin.

Input/Output Requirements

The first thing that must be assessed is the number of input and output points that the controlled system will require. This is done by identifying each device that is to send an input signal to the PC or which is to receive an output signal from the PC.

Sequence, Timing, and Relationships

Next, determine the sequence in which control operations are to occur and the relative timing of the operations. Identify the physical relationships between the I/O devices as well as the kinds of responses that should occur between them.

For instance, a photoelectric switch might be functionally tied to a motor by way of a counter within the PC. When the PC receives an input from a start switch, it could start the motor. The PC could then stop the motor when the counter has received a specified number of input signals from the photoelectric switch.

Each of the related tasks must be similarly determined, from the beginning of the control operation to the end.

Note Programs and Peripheral Devices are not compatible between the SYSMAC SP-series PCs and C-series PCs.

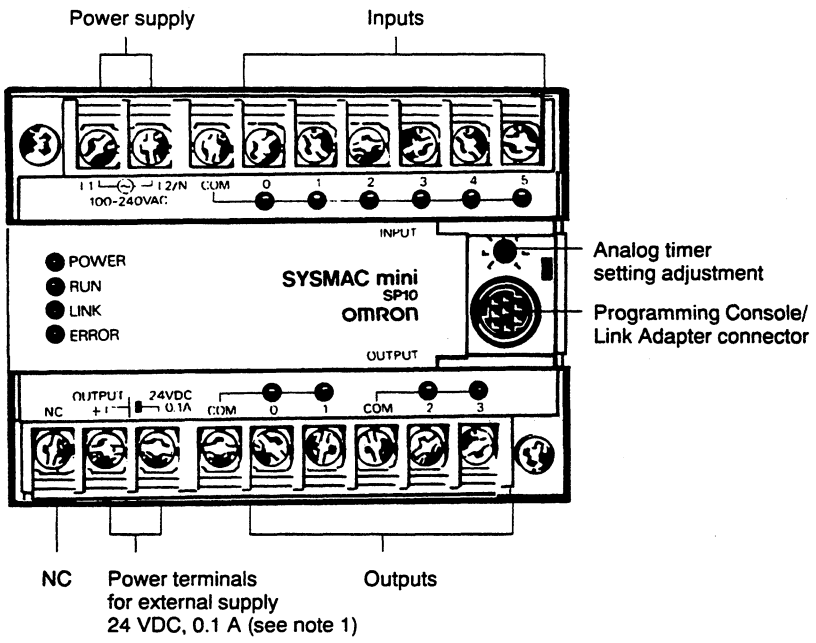
1–3 Units

This section presents the names and functions of the various components of the CPU, Programming Console, and Link Adapter.

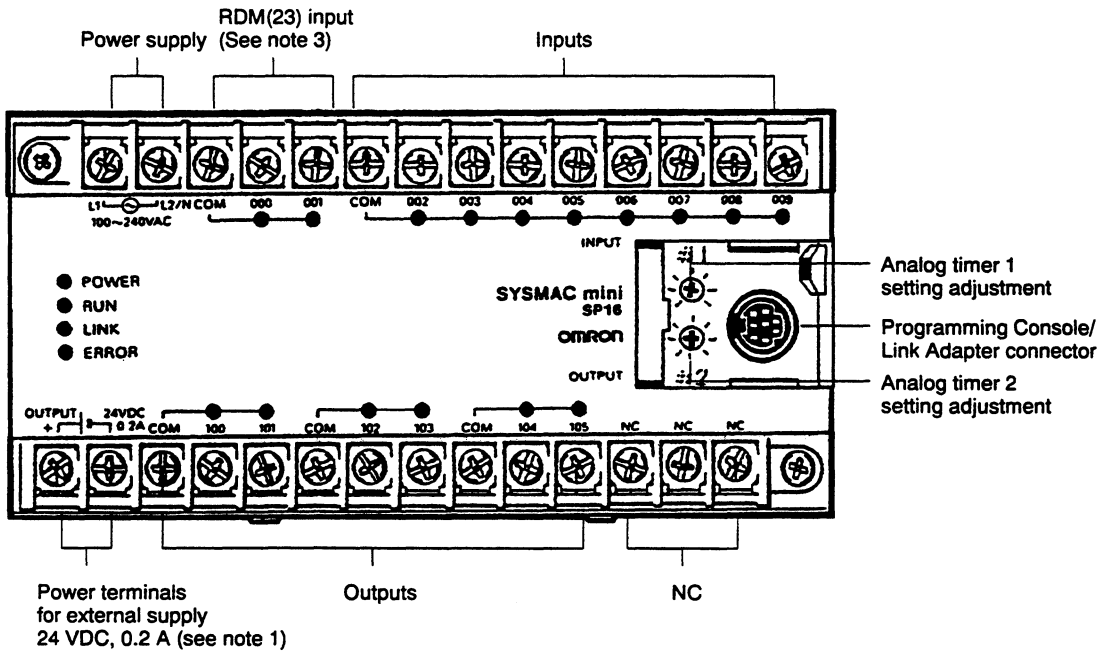
1–3–1 CPU

The SP-series PCs are shown below. Four models are available for each: two powered by a 100- to 240-VAC power supply and the other two powered by a 24-VDC power supply. Refer to *Appendix A Standard Models* for details.

SP10



SP16 and SP20 The SP20 is essentially the same as the SP16. The SP16 is shown below.



Note
1. The power terminals for external supply are provided for the 100 to 240 VAC model (SP__ -__ -A) only.
2. Connect nothing to the NC terminal.
3. Input 000 is the counter input and 001 is the hard reset input for the REVERSIBLE DRUM COUNTER, RDM(23). When RDM(23) isn't being used, these terminals can be used as normal input points but the input signal must be below 1 kHz.

Indicators

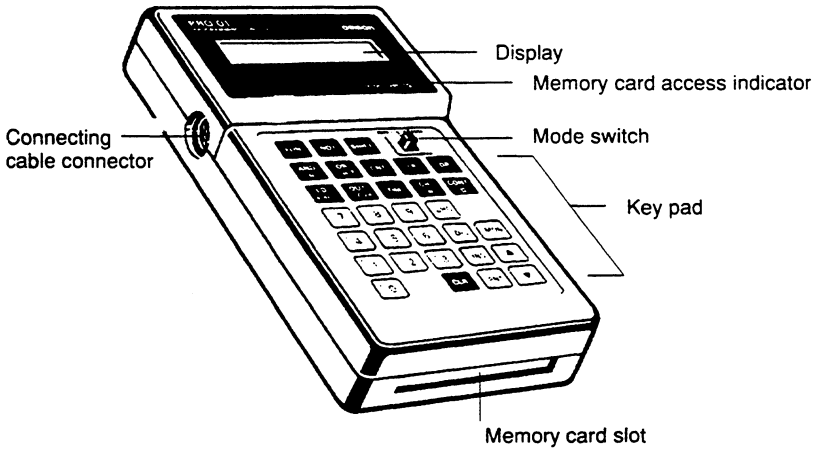
The PC has four indicators on the front panel: POWER, RUN, LINK, and ERROR. The functions of the indicators are presented as follows.

POWER (green): Lit while power is supplied.
RUN (green): Lit when the PC is in RUN mode and operating normally.

LINK(green): Lit when the PC Link is operating normally.
ERROR(red): Lights when self-diagnosis detects an error. The PC will stop operating.

1–3–2 Programming Console

The Programming Console is shown below.

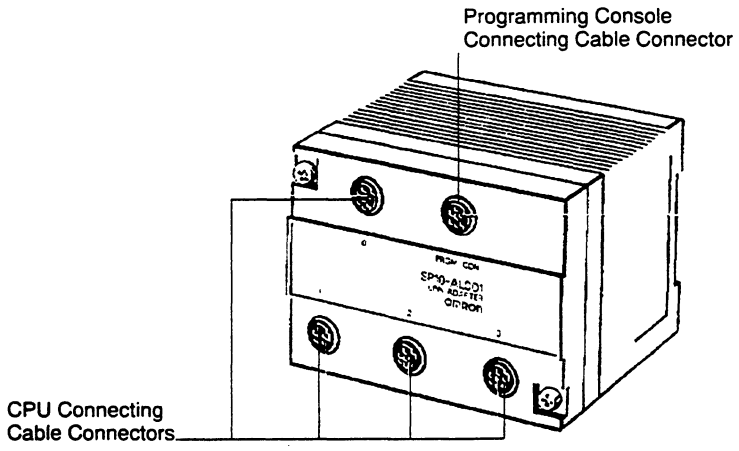


The Programming Console is used to write and transfer programs to the PC. It is also used to monitor operation and modify data. The Programming Console can be connected directly to the PC for single PCs. It can also be connected via a Link Adapter when PCs are connected in a PC Link configuration to access each PC individually without reconnection.

Note PCs connected to a Link Adapter cannot be directly connected to a Programming Console. The Programming Console is connected to the Link Adapter.

1–3–3 Link Adapter

The Link Adapter is shown below. The Link Adapter is used to link up to four CPUs so that data can be transferred between the CPUs and so that all of the CPUs can be accessed from the Programming Console from a single connection point. For details refer to *1-4 PC Configuration*.



1–3–4 Memory Cards

The Programming Console provides the ability to backup programs. The Memory Card slot located at the base of the keyboard allows programs to be transferred directly to and from the Programming Console. Each Card has a built-in battery to preserve data.

Only one model of Memory Card, HMC-ES141, may be used. Each Memory Card has 16 Kbytes of S-RAM. One Memory Card can hold up to 26 SP10 programs or up to 18 SP16 or SP20 programs.

A battery is built-in to the Memory Card to allow the data to be retained. The battery must be replaced within five years to ensure data is not lost. To remove the battery, insert a sharp object, like a pen tip, into the hole at the bottom right of the card. The new battery must be inserted within one minute of removing the old one.

Memory Cards have a write-protect switch. When the switch is ON, writing operations to the memory card will not be possible.

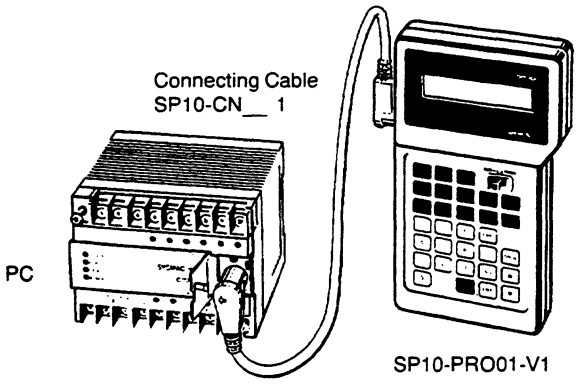
Caution While the Memory Card is being accessed, the M/C ON LED on the Programming Console will be lit. If the Memory Card is removed out from the Programming Console while the LED is ON, the data contained in memory on the Card may be damaged.

1–4 PC Configuration

The SP-series PCs can be configured to control a control system of 10 to 80 I/O points. An SP10 provides 10 I/O points (6 input and 4 output points), an SP16 provides 16 I/O points (10 input and 6 output points), and an SP20 provides 20 I/O points (12 input and 8 output points). A maximum of 4 SP-series PCs can be linked together via a Link Adapter, making a maximum of 80 I/O points with 4 SP20s.

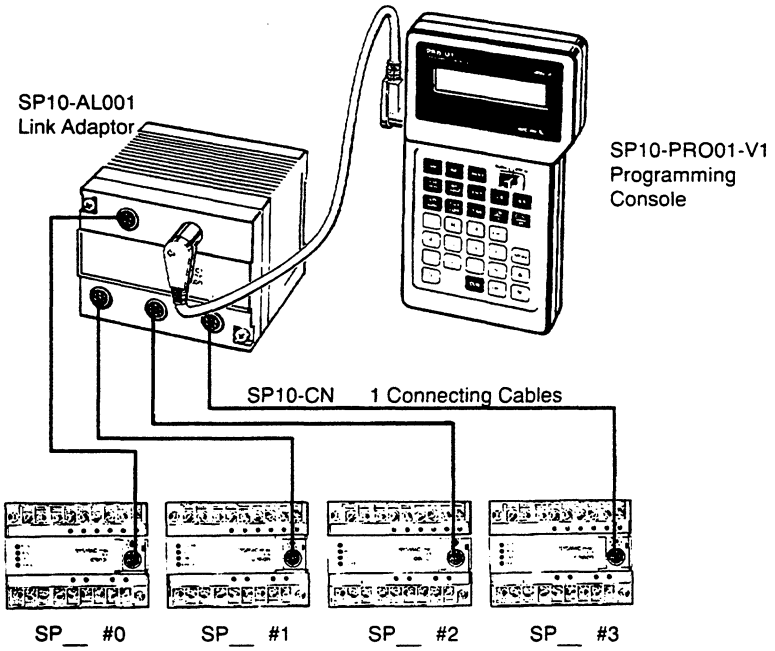
1–4–1 Basic Configuration

When only one SP-series PC is used, the number of I/O points available is 10 with the SP10, 16 with the SP16, or 20 with the SP20. Here, the Programming Console is connected directly to the CPU for programming and operation.



1–4–2 Expanded System Configuration: PC Links

Up to four PCs can be linked with a Link Adapter to increase the number of I/O points to 80 maximum. Although each PC still operates on its own program, no special programming is needed to transfer data between the PCs via LR bits. Up to 128 bits of data (32 bits per PC) can be shared between the PCs through their LR areas. Refer to *3–2–5 LR Area* for details on LR bits.



Note

1. When two or more PCs are linked, apply power to all of the PCs at once or to PC #0 last.
2. When using a Link Adapter, one PC must be connected to connector number 0 on the Link Adapter.
3. Once CPUs are connected to the Link Adapter and turned ON, unit numbers are automatically assigned to the CPUs by the Link Adapter. Do not change the point of connection of any CPU after a unit number has been assigned to it. If CPUs are connected to a connector for a different unit number, unpredictable errors will occur during operation.

Unit Numbers

When four PCs are linked, each are given a number from 0 through 3 depending on which connector on the Link Adapter is used. PC #0 controls LR data transfers between the PCs. The Programming Console is connected to the Link Adapter and can program and monitor any of the four PCs.

SECTION 2
Installation

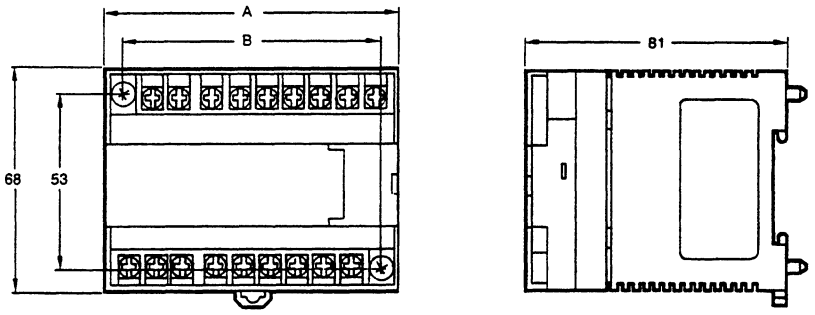
This section provides information on mounting and wiring the CPUs and on I/O specifications. Basic unit connections are described in *1–4 PC Configuration*. Detailed specifications are provided in *Appendix B Specifications*.

2–1 Dimensions

This section gives mounting dimensions. All dimensions are in millimeters.

CPUs

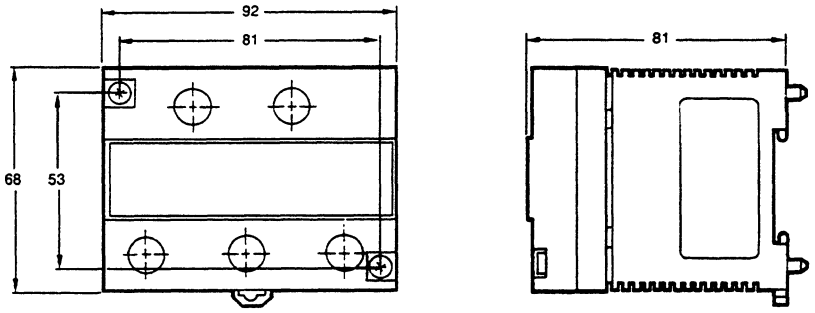
SP10-D_-_, SP16-D_-_, SP20-D_-_



PC model	Dimension A	Dimension B
SP10-D_-_	92	81
SP16-D_-_	135	124
SP20-D_-_	160	149

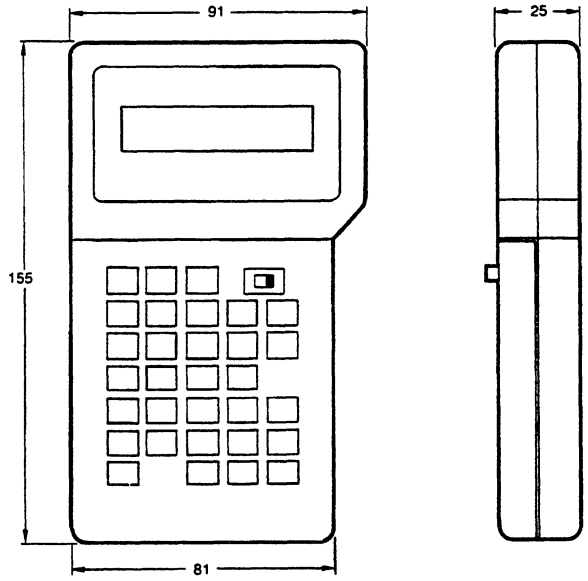
Link Adapter

SP10-AL001

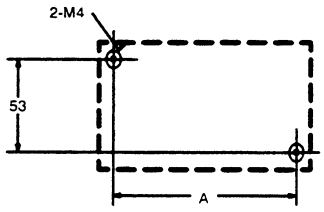


Programming Console

SP10-PRO01-V1



Surface Mounting Dimensions



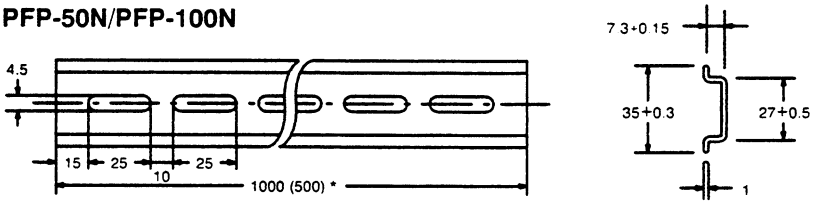
PC model	Dimension A
SP10-D_-_	81
SP16-D_-_	124
SP20-D_-_	149

Mounting Track

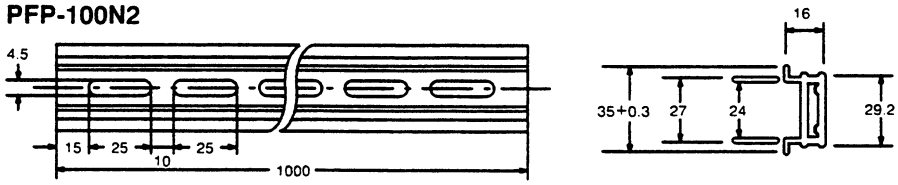
The SP-series PCs can be mounted onto DIN Tracks.

Model No.	Length (L)
PFP-50N	50 cm
PFP-100N	1 m
PFP-100N2	1 m

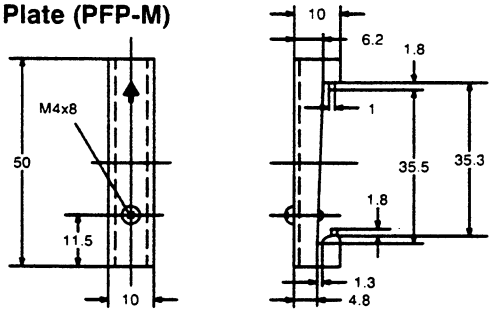
PFP-50N/PFP-100N



PFP-100N2



End Plate (PFP-M)



2–2 Installation

2–2–1 Installation Environment

Although the SP-series Programmable Controllers are highly reliable and durable, a number of factors should be considered when installing them. Do not expose an SP-series PC to the following conditions:

- An ambient temperature that falls below 0°C or exceeds 55°C for the CPU, or that falls below 0°C or exceeds 45°C for the Programming Console.

- Abrupt changes in temperature that cause condensation.
- A relative humidity less than 10% or greater than 90%.
- Corrosive or flammable gas.
- Dust, salt, or iron particles.
- Direct vibration or shock.
- Direct sunlight.
- Splashes of water, oil, or chemicals.

2–2–2 Cooling

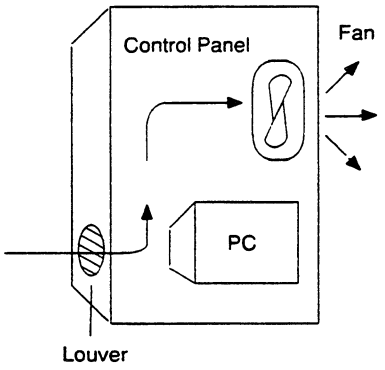
There are two points to consider to ensure that the PC does not overheat. The first is the clearance between the CPUs and control panel surround them, and the second is the installation of a cooling fan.

Clearance

The CPUs need to have sufficient room between them to allow for I/O wiring, and additional room to ensure that the wiring does not hamper cooling. The CPU's must be mounted close enough so that the length of the Connecting Cable between any CPU and the Link Adapter does not exceed 4 meters.

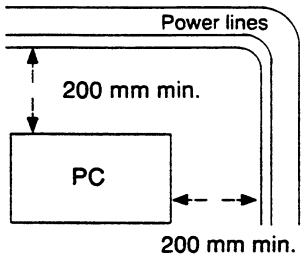
Cooling Fan

Ensure adequate ventilation is provided for the PCs. A cooling fan is not always necessary, but may be needed if the PC is mounted in a warm or enclosed area or over a source of heat. Although it is best to avoid installing the PC in a warm area, use a cooling fan or an air conditioner, as shown in the following illustration, to maintain the ambient temperature within specifications.



2–2–3 Preventing Noise

In order to prevent noise from interfering with the operation of the PC, use AWG 14 twisted-pair cables (cross-sectional area of at least 2 mm^2). Do not mount the PC in a control panel in which high-power equipment is installed and make sure the point of installation is at least 200 mm away from power cables, as shown in the following diagram. Ground the panel to which the PC is mounted.



Whenever possible, use wiring conduit to hold the I/O wiring. Standard wiring conduit should be used, and it should be long enough to completely contain the I/O wiring and keep it separated from other cables.

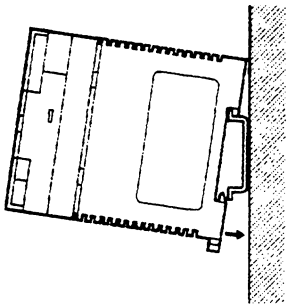
2–2–4 Mounting Requirements

The system consists of one to four CPUs and, if more than one CPU is used, a Link Adapter. The Units may be mounted horizontally or vertically, as desired. Do not mount a Unit on its side. The Unit should be mounted with the printing on the front panel oriented as it would normally be read. The PC can be mounted using DIN Track or mounted directly to any sturdy support meeting the environmental specifications listed in *Appendix B Specifications.*

Track Mounting

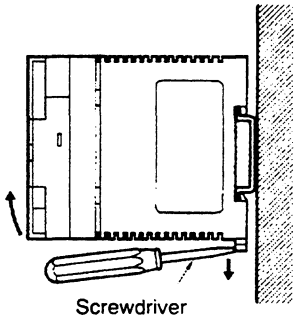
The PC may be mounted using DIN Track if desired. Use DIN Track 35 mm wide. There is a groove on the back of the Unit that is used to attach it to the DIN Track. When mounting to DIN Track, be sure to remove the mounting screws.

Mounting



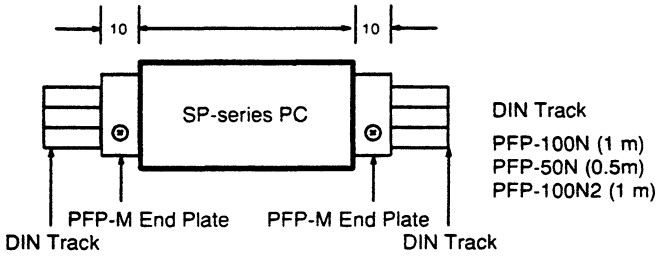
Engage the top hook to the track and push the PC in until the bottom hook locks onto the track.

Removal



Push down the bottom hook with a screwdriver and push the PC upward.

Attach an End Plate to the left and right sides of the Unit to hold it in place.



Note Remove the mounting screws when mounting on a track.

2–3 Wiring

Caution Do not wire the terminal marked "NC."

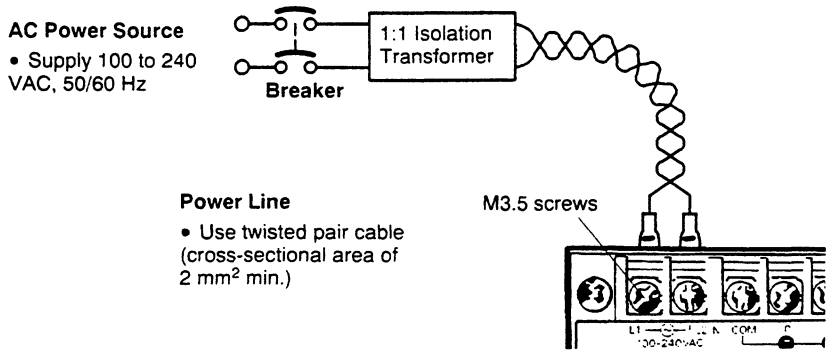
2–3–1 Power Supply

Use independent power sources for the inputs, the output loads, and the PC. Voltage fluctuations caused by current surges to motors may affect operation

of the PC. When using more than one PC, use a separate power supply for each PC for two reasons, first to prevent voltage drops caused by surge currents and secondly, to prevent the breaker from malfunctioning.

The following diagrams show the proper way to connect the power source to the PC. Refer to *Appendix B Specifications* for detailed specifications.

AC Connections

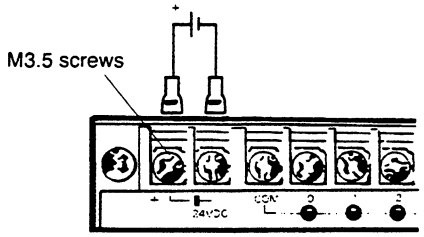


To reduce noise interference from the power lines, use twisted pair cables. Noise can also be significantly reduced by connecting a 1-to-1 isolation transformer.

Note Do not short the positive and negative lines.

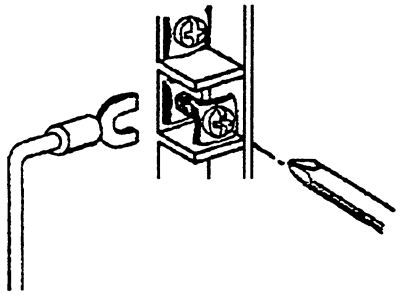
DC Connections

Supply 24 VDC and keep voltage fluctuations within the specified range.



2–3–2 I/O Connections

Connect the I/O devices to the I/O terminals using wire with a cross-sectional area of 1.04 to 2.63 mm². The terminals have screws with M3.5 heads and self-rising pressure plates. Connect the lead wires to the terminals as shown below. Tighten the screws with a torque of 8 kg-cm maximum.

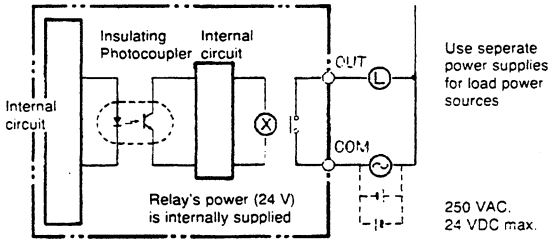


If you wish to attach solderless type terminals to the ends of the lead wires, use terminals having the following dimensions.

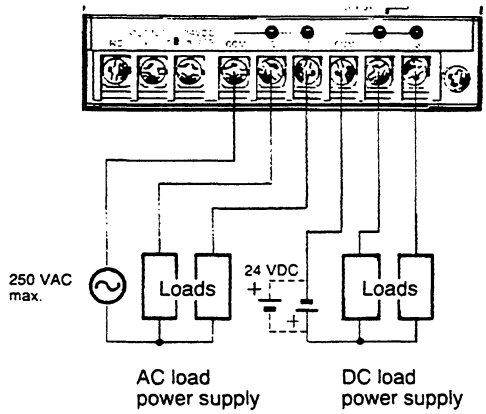
Output Circuits

Refer to *Appendix B Specifications* for detailed specifications.

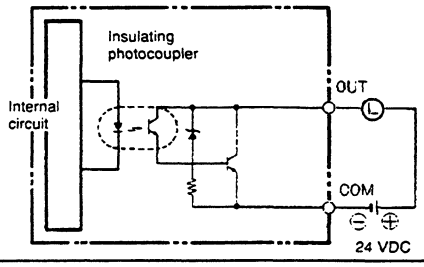
Relay Contact Outputs

Maximum switching capacity	Resistive loads	250 VAC, 2 A (cosf=1), 24 VDC, 2A/pt
	Inductive loads	250 VAC, 0.5 A (cosf=0.4)/pt
Minimum switching capacity	5 VDC, 100 mA	
Circuit configuration		

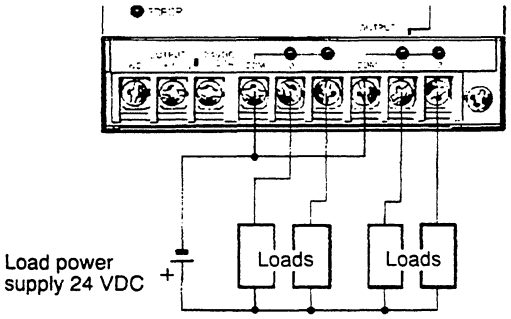
The following example uses an SP10 CPU.



Transistor Outputs

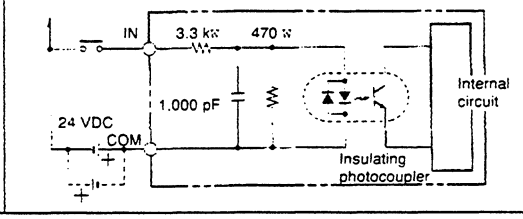
Maximum switching capacity	24 VDC $^{+10\%}/_{-15\%}$, 0.3 A
Leakage current	0.1 mA max.
Residual voltage	1 V max.
Circuit configuration	

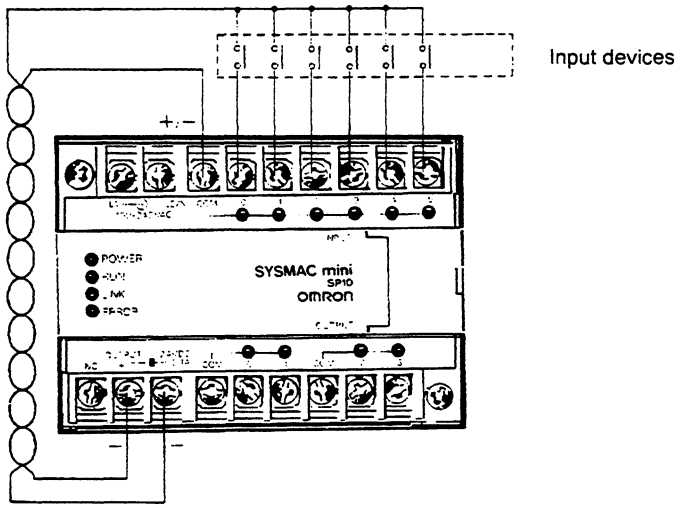
The following example uses an SP10 CPU.



Input Circuits

Either positive or negative poles of the power supply can be connected to the common (COM) terminals, enabling connection of both PNP (negative common) and NPN (positive common) inputs.

Input voltage	24 VDC $^{+10\%}/_{-15\%}$
Input current	7 mA Typ. (24 VDC)
Circuit configuration	



The power source of the SP10 for external supply is rated at 0.1 A, 24 VDC max., and the power source of the SP16 and SP20 is rated at 0.2 A, 24 VDC max. The input circuit consumes about 7 mA (typ. at 24 VDC) per input point.

Transistor outputs with a current consumption up to 0.2 A can be used with the SP10, and a current consumption up to 0.32 A with the SP16 and SP20. Relay contact outputs require a current of 0.013 A each, so when all of the relays are ON, the external power supply capacity is 0.1 A for the SP10 and 0.2 A for the SP16 and SP20. When using the SP10 as the power source for input devices such as sensors, etc., make sure that the power consumption of the devices does not exceed the ratings of the PC.

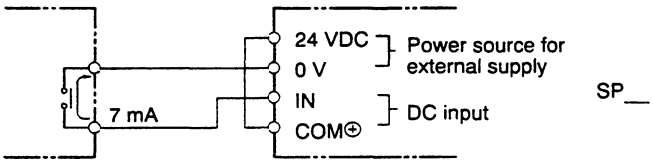
DC Input Examples

The following diagrams show the correct way to wire the terminals on the CPU. When wiring, work carefully to ensure that all terminals are wired correctly. If an input device is connected to an output point, damage may result. Check all I/O devices to ensure they meet the specifications (refer to *Appendix B* Specifications).

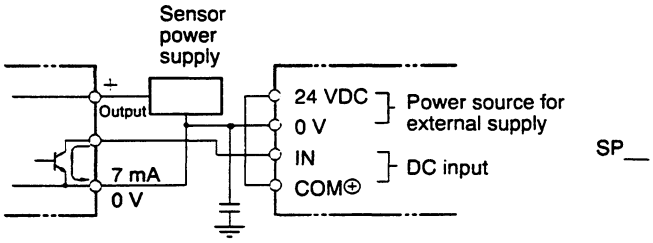
The DC inputs in the following diagrams are NPN (positive common). Reverse the polarity if PNP (negative common) is used.

Use the CPU's 24 VDC power supply output to supply power to inputs. If the maximum output current of 0.3 A is not sufficient, use a separate DC power supply.

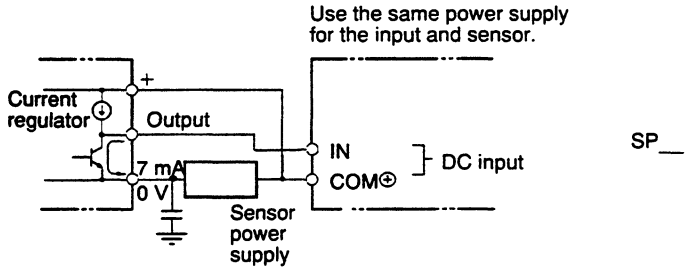
DC Input Devices



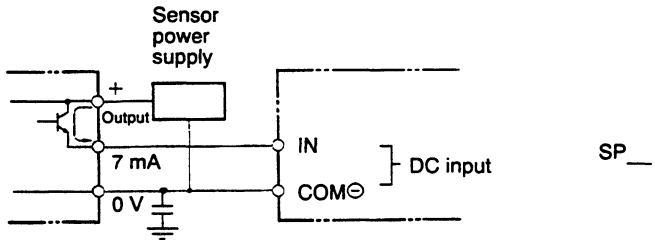
NPN Open-collector Outputs



NPN Current Outputs



PNP Current Outputs



Note 1. When using the DC model (SP__ -D_ -D), do not input the signal through a NC contact (which makes the PC operate when the externally input signal turns OFF). Proper operation for power interruptions will not be possible if NC contacts are used in conjunction with counter, shift, or keeping (latching) instructions.

2. When using a capacitive proximity sensor, ground the 0 V terminal of the power supply with a 0.01 μF, 630 V capacitor to ensure stable operation.

2–3–3 Precautions

Unit Sticker

A sticker is provided on the upper face of the CPU to prevent foreign objects, such as wire clippings, from entering the CPU. Leave this protective sticker on until the CPU is ready for operation. The sticker must be removed before operation to enable proper cooling.

Contact Outputs

High inductance on for contact outputs will reduce relay life. Keep inductance low and use an arc suppressor (such as a diode for DC loads). This is particularly important with inductive DC loads.

Vibration

Relay operation may be adversely affected if the relay is located near contactors, valves, motors, or other devices that produce vibration.

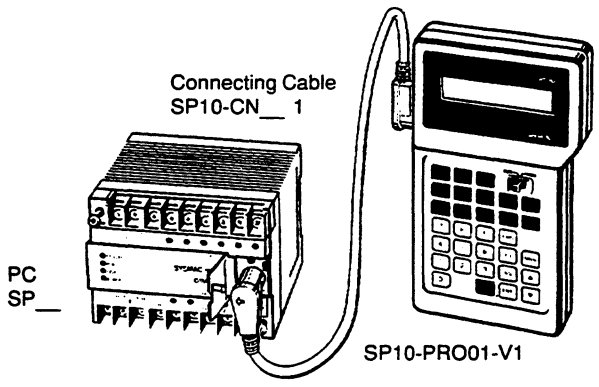
Protective Circuits

Omron recommends the use of arc suppressors to increase contact life and alleviate the affects of noise. Arc suppressors, however, will delay release time somewhat and, if used incorrectly, they can inhibit proper operation. The most common arc suppressors for AC are capacitor-resistor circuits and varistor circuits; for DC: capacitor-resistor circuits, diode circuits, and varistor circuit. Do not use a capacitor without a resistor as the charging current flow to the capacitor when current is turned ON can cause the contacts to fuse.

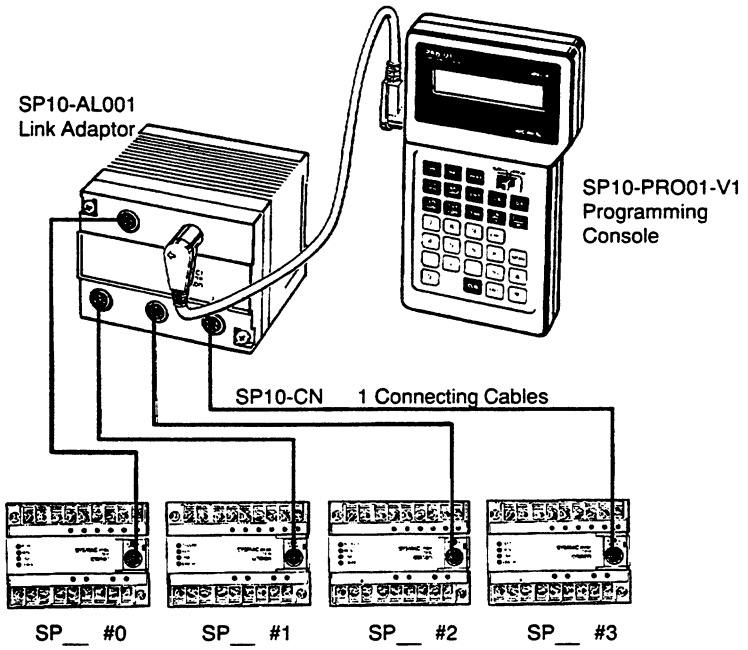
2–4 Programming Console

Open the connector cover of the PC, align the notch on the connector, and press the connector into place.

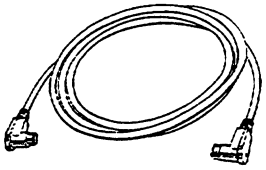
Connection to a CPU



Connection to a Link Adaptor



Connecting Cable Use one of the following Connecting Cables to connect the Programming Console.

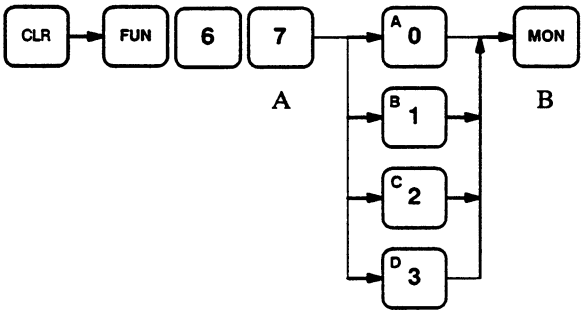


SP10-CN221 (2 m)
SP10-CN421 (4 m)

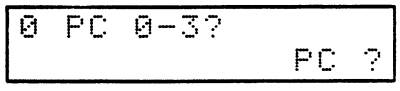
Note The sum of the cable lengths between Unit #0 and the Link Adapter and between the Link Adapter and the Programming Console must be 4.2 m maximum.

2–4–1 Designating the PC

Any of the PCs connected in a PC Link may be accessed through the Link Adapter using the Programming Console. Use the following key sequence to specify the number of the desired PC. The PC can be designated in either PROGRAM or RUN mode.



A



The PC's operation or operation mode is not affected by changing the PC designation. When the mode switch of the Programming Console and the

operation mode of the PC being monitored are identical, the following message is displayed. The number in the top left corner indicates the number of the PC being monitored, in this case PC #1.

B
```
1-000
```

When the mode switch of the Programming Console and the operation mode of the PC being monitored are not identical, the following message is displayed.

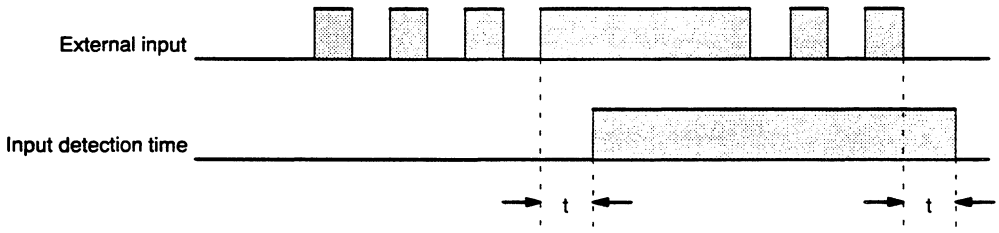
B
```
<RUN>
MODE SET ERR
```

In this example, the message indicates that the Programming Console is set to PRGM (program) mode, and that PC #1 is set to RUN mode. To clear the error and reset the corresponding alarm, turn the Programming Console mode to RUN and then change it back to PRGM mode. PC #1 will change to PRGM mode.

Note If there is a communication error, the display will read "COMM ERR".

2–4–2 Input Filters

To prevent the PC from malfunctioning due to the chattering (bouncing) of the input device signals or induced noise, the input signals are received via a filter. The filter may be adjusted so that input pulses of a duration less than a minimum specified duration of the filter are ignored. The minimum duration before the detection of an input signal may be set to 0, 1, 5, or 10 ms. The following diagram illustrates the use of a filter.



The input detection time, t, for the various possible settings is given in the following table. The "key" column shows which key is pressed to input each setting in the key sequence below.

Key	Setting	Actual detection time
0	0 ms	t = 150 μs
1	1 ms	t = 1 to 1.5 ms
2	5 ms	t = 5 to 5.5 ms
3	10 ms	t = 10 to 10.5 ms

During the period t to t + 0.5 ms, the positive and negative transitions of the input signal may or may not be detected.

Filter Value Settings

The filter values are set using the Programming Console. The input circuits are grouped into three groups. The circuits included in each group depend on the PC, as shown in the table below. A different filter value can be set for

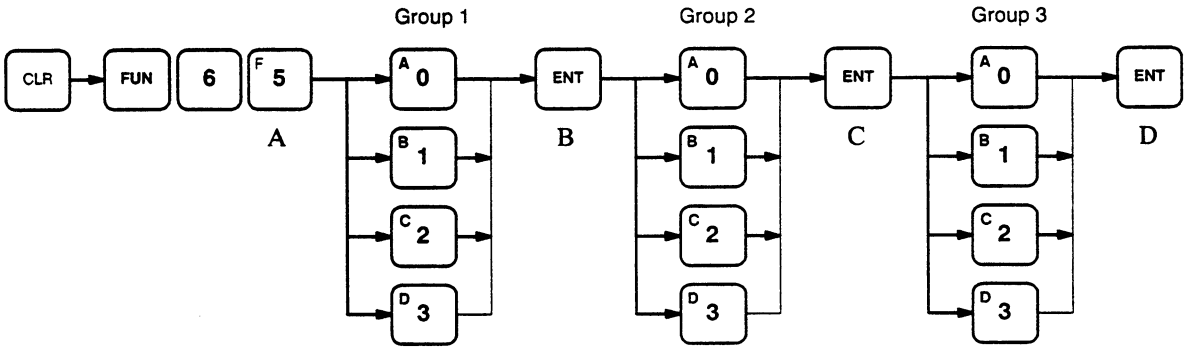
each group. The filter values can be set in PROGRAM mode only and must be set before operating the PC. The filter values are set simultaneously in the PC and in the Programming Console.

PC model	Group 1 inputs	Group 2 inputs	Group 3 inputs
SP10-D_-_	0 to 2	3 to 5	None
SP16-D_-_	0 and 1	2 to 5	6 to 9
SP20-D_-_	0 and 1	2 to 9	10 and 11

Always set the filter values after transferring the program and before starting operation. Set the filter value to 5 or 10 ms when the PC is installed in environments subject to noise, or when input devices that may cause chattering are connected to the PC. If the filter value is set to 0 or 1 ms, be sure that the input wiring is carefully installed to prevent interference.

Key Sequence

Input 0 to specify 0 ms, 1 for 1 ms, 2 for 5 ms, and 3 for 10 ms.



The following diagrams illustrate the Programming Console displays at the respective positions marked in the key sequence diagram.

A
```
0 FILTER VAL SET
GROUP1 SET NO.?
```

B
```
0 FILTER VAL SET
GROUP2 SET NO.?
```

C
```
0 FILTER VAL SET
GROUP3 SET NO.?
```

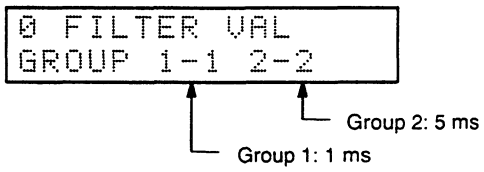
D
```
0 FILTER VAL  OK
```

Set the filter values of groups 1, 2, and 3 at the same time. After entering the filter values, read them on the Programming Console for confirmation. Use the following key sequence. Reading is possible in either RUN or PROGRAM mode.

Key Sequence

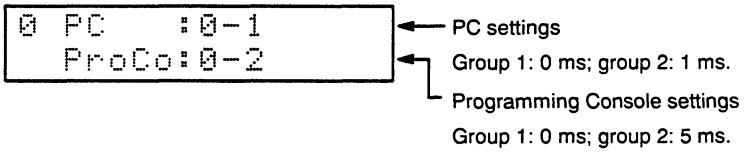
The Programming Consoles will display the information in the following formats.

SP10-PRO01

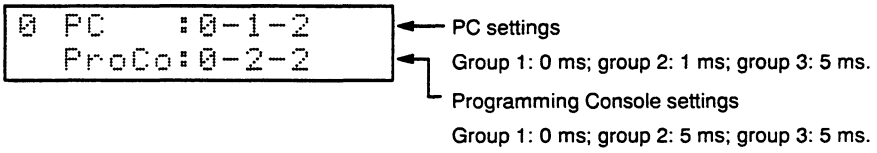


SP10-PRO01-V1

The display will show the settings for groups 1 and 2 when the programming console is connected to an SP10.



The display will show the settings for groups 1, 2, and 3 when the programming console is connected to an SP16 or SP20.



SECTION 3
Programming

This section takes you all the way through the programming procedure from understanding memory area allocation to debugging and executing the program. *Section 4 Operation* will then provide procedures for monitoring PC operation and manipulating data after you have written, input, and debugged the program.

3–1 Introduction

There are several basic steps involved in writing a program. Sheets that can be copied to aid in programming are provided in *Appendix F I/O Assignment Sheets* and *Appendix G Program Coding Sheet*.

1, 2, 3..

1. Obtain a list of all I/O devices and the I/O points that have been assigned to them and prepare a table that shows the I/O bit allocated to each I/O device.
2. Determine what words are available for work bits and prepare a table in which you can allocate these as you use them.
3. Also prepare tables of TC numbers so that you can allocate these as you use them. Remember, the function of a TC number can be defined only once within the program. (TC numbers are described in *3–7–13 Timers and Counters*.)
4. Draw the ladder diagram.
5. Input the program into the Programming Console.
6. Check the program for syntax errors and correct these.
7. Transfer the program from the Programming Console to the CPU and execute the program to check for execution errors and correct these.
8. After the entire Control System has been installed and is ready for use, execute the program and fine tune it if required.

3–2 Memory Areas

Details, including the name, acronym, range, and function of each area are summarized in the following table. All but the last area are data areas. Data and memory areas are normally referred to by their acronyms. Bits not listed in the following table cannot be used.

Area	PC	No. of bits	Word addresses	Bit addresses	Function
Input bits	SP10	6	00	0000 to 0005	Input external signals to the PC. These bits can be used as many times as required in the program.
	SP16	10	00	0000 to 0009	
	SP20	12	00	0000 to 0011	
Output bits	SP10	4	01	0100 to 0103	Each of these bits can be used in only one instruction controlling its status, but can be used as many times as required in other instructions. If the status of the same output bit is controlled by more than one instruction, only the status determined by the last instruction will be output.
	SP16	6	01	0100 to 0105	
	SP20	8	01	0100 to 0107	
Work bits	SP10	36	00	0008 to 0015	These bits are used within the program to aid programming.
			01	0104 to 0115	
			02	0200 to 0215	
	SP16	208	00	0010 to 0015	
			01	0106 to 0115	
			02	0200 to 0215	
			10 to 20	1000 to 2015	
	SP20	204	00	0012 to 0015	
			01	0108 to 0115	
			02	0200 to 0215	
			10 to 20	1000 to 2015	

Area	PC	No. of bits	Word addresses	Bit addresses	Function
Dedicated bits	SP10	20	03	0300 to 0315	These bits are assigned specific functions. For details, refer to the table in *3–2–4 Dedicated Bits*.
			04	0408 to 0411	
	SP16, SP20	69	03	0300 to 0315	
			04	0408 to 0411	
			05	0515	
			07	0700 to 0715	
			08	0800 to 0815	
			09	0900 to 0915	
Data Retention (DR)	All	256 max.	DR 00 to DR 15	DR 0000 to DR 1515	These bits retain their ON/OFF state even during power interruptions. The number of DR bits decreases if more link bits are designated.
Link Relay (LR)	All	128 max.	LR 00 to LR 07	LR 0000 to LR 0715	Used to exchange data with other SP-series PCs through a Link Adapter. To access the LR area, the LR area must be defined via the Programming Console.
Timer/Counter (TC)	All	16	TIM/CNT 00 to 15		Used to define timers and counters and to access Completion Flags, PV, and SV for them. TC 14 is used by the HIGH-SPEED TIMER instruction (TIMH), and TC 15 is used by the ANALOG TIMER instruction.

3–2–1 Data Area Structure

When designating a data area, the acronym for the area is always required for the DR, TC, and LR areas.

An actual data within any data area but the TC area is designated by its address. The address designates the bit or word within the area where the desired data is located. The TC area consists of TC numbers, each of which is used for a specific timer or counter defined in the program. Refer to *3–2–8 TC (Timer/Counter) Area* for more details on TC numbers.

The rest of the data area consists of words, each of which consists of 16 bits numbered 00 through 15 from right to left. words 000 and 001 are shown below with bit numbers. Here, the content of each word is shown as all zeros. Bit 00 is called the rightmost bit; bit 15, the leftmost bit.

The term least significant bit is often used for rightmost bit; the term most significant bit, for leftmost bit. These terms are not used in this manual because a single data word is often split into two or more parts, with each part used for different parameters or operands. When this is done, the rightmost bits of a word may actually become the most significant bits, i.e., the leftmost bits in another word, when combined with other bits to form a new word.

Bit number	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Word 000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Word 001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

To designate data by word, all that is necessary is the acronym (if required) and the two-digit word address. To designate data by bit, the word address is combined with the bit number as a single four-digit address. The following table show examples of this. The two rightmost digits of a bit designation must indicate a bit between 00 and 15, i.e., the rightmost digit must be 5 or less the next digit to the left, either 0 or 1.

The same TC number can be used to designate either the present value (PV) of the timer or counter, or a bit that functions as the Completion flag for the timer or counter.

Area	Word designation	Bit designation
I/O, work, and dedicated bits	00	0015 (leftmost bit in word 00)
TC	TC 03 (designates PV)	TC 03 (designates Completion Flag)
LR	LR 07	LR 0000
DR	DR 15	DR 0513

Data Structure

Word data input as decimal values is stored in binary-coded decimal (BCD); word data entered as hexadecimal is stored in binary form. Each four bits of a word represents one digit, either a hexadecimal or decimal digit, numerically equivalent to the value of the binary bits. One word of data thus contains four digits, which are numbered from right to left. These digit numbers and the corresponding bit numbers for one word are shown below.

Digit number	3				2				1				0			
Bit number	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Contents	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

When referring to the entire word, the digit numbered 0 is called the rightmost digit; the one numbered 3, the leftmost digit.

When inputting data into data areas, it must be input in the proper form for the intended purpose. This is no problem when designating individual bits, which are merely turned ON (equivalent to a binary value of 1) or OFF (a binary value of 0). When inputting word data, however, it is important to input it either as decimal or as hexadecimal, depending on what is called for by the instruction it is to be used for. *3–7 Instruction Set* specifies when a particular form of data is required for an instruction.

Converting Different Forms of Data

Binary and hexadecimal can be easily converted back and forth because each four bits of a binary number is numerically equivalent to one digit of a hexadecimal number. The binary number 0101111101011111 is converted to hexadecimal by considering each set of four bits in order from the right. Binary 1111 is hexadecimal F; binary 0101 is hexadecimal 5. The hexadecimal equivalent would thus be 5F5F, or 24,415 in decimal ($16^3 \times 5 + 16^2 \times 15 + 16 \times 5 + 15$).

Decimal and BCD are easily converted back and forth. In this case, each BCD digit (i.e., each group of four BCD bits) is numerically equivalent of the corresponding decimal digit. The BCD bits 0101011101010111 are converted to decimal by considering each four bits from the right. Binary 0101 is decimal 5; binary 0111 is decimal 7. The decimal equivalent would thus be 5,757. Note that this is not the same numeric value as the hexadecimal equivalent of 0101011101010111, which would be 5,757 hexadecimal, or 22,359 in decimal ($16^3 \times 5 + 16^2 \times 7 + 16 \times 5 + 7$).

Because the numeric equivalent of each four BCD binary bits must be numerically equivalent to a decimal value, any four bit combination numerically greater then 9 cannot be used, e.g., 1011 is not allowed because it is numerically equivalent to 11, which cannot be expressed as a single digit in decimal notation. The binary bits 1011 are of course allowed in hexadecimal are a equivalent to the hexadecimal digit C.

Decimal Points

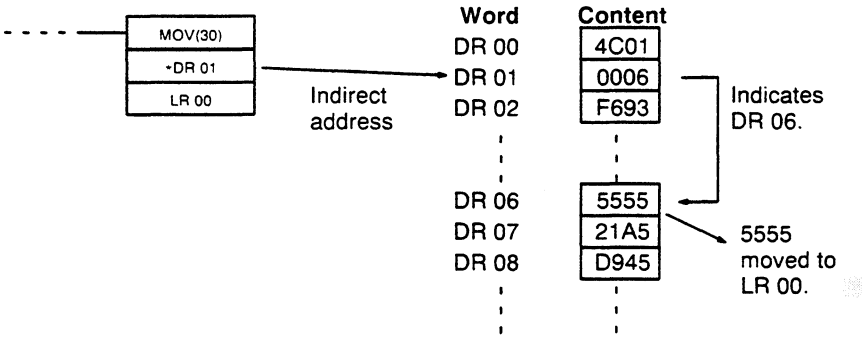
Decimal points are used in timers only. The least significant digit represents tenths of a second. All arithmetic instructions operate on integers only.

Indirect Addressing

Normally, when the content of a data area word is specified for an instruction, the instruction is performed directly on the content of that word. For example,

suppose CMP(32) (COMPARE), with word 05 as the first operand and DR 10 as the second operand, is used in the program. When this instruction is executed, the content of word 05 is compared with that of DR 10.

It is also possible, however, to use indirect DR addresses as operands for instructions. If *DR 01 is specified as the data for a programming instruction, the asterisk in front of DR indicates that it is an indirect address that specifies another DR word which contains the actual operand data. If, in this case, the content of DR 01 is 06, then *DR 01 indicates DR 06 as the word that contains the desired data, and the content of DR 06 is used as the operand in the instruction. The following example shows this type of indirect addressing with the MOVE instruction (MOV(30)).



3–2–2 I/O Bits

Input bits are used to read the status of input terminals, i.e., input bits are used as operands in the program to control program execution. Output bits are used to control the status of output terminals, i.e., various conditions in the program are used to determine the status of output bits through the OUTPUT and other instructions. The relationship of the I/O bits and terminals in the SP10 is shown below. The relationship between I/O bits and terminals in the SP16 and SP20 follows the same pattern.

Inputs			Outputs		
Word	**Bit**	**Terminal**	**Word**	**Bit**	**Terminal**
00	0000	0	01	0100	0
	0001	1		0101	1
	0002	2		0102	2
	0003	3		0103	3
	0004	4		---	
	0005	5			

After the program is executed, the status of outputs determined by the program is actually output from the output bits to the output terminals. Also, the current status of all inputs is read from the input terminals to the input bits.

Caution Do not use normally closed input signals for SP-series PCs with DC power supplies. Doing so can cause counters and shift registers to reset and bits programmed with the KEEP instruction to invert when power is interrupted, resulting in errors in program execution.

3–2–3 Work Bits

Work words and bits can be used in programming as required to control other bits. The work bits listed in the following table well as bits in the DR and LR

areas can be used as work bits if they are not used for other purposes. The actual application of work bits is described in *3–6–6 Work Bits (Internal Relays)*. In the SP10, bits 0006 and 0007 cannot be used for work bits or for any other purpose.

SP10		SP16		SP20	
Word	**Bits**	**Word**	**Bits**	**Word**	**Bits**
00	0008 to 0015	00	0010 to 0015	00	0012 to 0015
01	0104 to 0115	01	0106 to 0115	01	0108 to 0115
02	0200 to 0215	02	0200 to 0215	02	0200 to 0215
---		10 to 20	1000 to 2015	10 to 20	1000 to 2015

3–2–4 Dedicated Bits

The dedicated bit area contains flags and control bits used for monitoring system operation, accessing clock pulses, and signalling errors. In the SP10, word addresses range from 03 through 04; bit addresses, from 0300 through 0411. In the SP16 and SP20, word addresses range from 03 through 09; bit addresses, from 0300 through 0915. Bits in the dedicated bit area that are not assigned functions cannot be used for work bits or for any other purpose.

The following table lists the functions of flags and control bits in the dedicated bit area. Most of these bits are described in more detail following the table.

Unless otherwise stated, flags are OFF until the specified condition arises, when they are turned ON. Bits 0311 through 0315 are turned OFF when the END is executed at the end of each program scan, and thus cannot be monitored on the Programming Console. Other bits are OFF until set by the user.

Information in the following table applies to the SP10, SP16, and SP20.

Word	Bit	Function	
03	0300	PC #0	Turns ON when a PC link error occurs.
	0301	PC #1	
	0302	PC #2	
	0303	PC #3	
	0304	PC #0	Turns ON when PC link is normal or in RUN mode.
	0305	PC #1	
	0306	PC #2	
	0307	PC #3	
	0308	1.0-second Clock Pulse	
	0309	0.1-second Clock Pulse	
	0310	0.01-second Clock Pulse	
	0311	Error (ER) Flag	
	0312	Carry (CY) Flag	
	0313	Less Than (LE) Flag	
	0314	Equals (EQ) Flag	
	0315	Greater Than (GR) Flag	
04	0400 to 0407	Cannot be used	
	0408	Always ON Flag	
	0409	Always OFF Flag	
	0410	First Scan Flag	
	0411	Step Flag	
	0412 to 0415	Cannot be used	

Information in the following table applies to the SP16 and SP20 only.

Word	Bit	Function
05	0500 to 0514	Cannot be used
	0515	DR Data Transfer Enable Bit
06	0600 to 0615	Cannot be used
07	0700 to 0707	Maximum Scan Time Area
	0708 to 0715	Current Scan Time Area
08	0800 to 0815	ATM1 Set Value Area
09	0900 to 0915	ATM2 Set Value Area

Descriptions: SP10, SP16, and SP20

Error Flag

Bit 0311 turns ON when data for an arithmetic operation or indirectly addressed data is not in BCD. It also turns ON when a specified operand exceeds the data area, e.g., when an operand requires two words and the last word in a data area is designated.

Arithmetic Flags

The following flags are used in arithmetic calculation , and comparison instructions. These flags are all reset when END is executed, and therefore cannot be monitored from a programming device.

Carry Flag, CY	Bit 0312 turns ON when a carry occurs as a result of arithmetic operation.
Less Than Flag, LE	Bit 0313 turns ON when the result of a comparison operation between two operands shows the first to be less than the second.
Equals Flag, EQ	Bit 0314 turns ON when the result of a comparison shows two operands to be equal or when the result of an arithmetic operation is zero.
Greater Than Flag, GR	Bit 0315 turns ON when the result of a comparison operation between two operands shows the first to be greater than the second.

For relations between arithmetic flags and instructions, refer to *Appendix E*.

Always ON/OFF Flags

Bit 0408 is always ON and bit 0409 is always OFF. These bits can be programmed to control external indicating devices such as an LED to monitor the PC's operating status. They can also be used in programming when an instruction is to be executed every scan.

First Scan Flag

Bit 0410 turns ON when program execution starts and turns OFF after one scan.

Step Flag

Bit 0411 turns ON for one scan when step execution is started by the STEP instruction.

Descriptions: SP16 and SP20 Only

DR Data Transfer Enable Bit

Turn bit 0515 ON to transfer DR data from EEPROM to RAM when power is applied to the PC. This bit will be ON after the "DR Area Transfer" operation has been performed. The status of bit 0515 is retained in a power interruption, i.e., DR data will be transferred from EEPROM to RAM when the power is turned ON if bit 0515 is ON when power is interrupted. If you want to retain the DR data as it was just before a power interruption, turn bit 0515 OFF with the "Force Set/Reset" operation. Bit 0515 is turned OFF in the "Data Clear" operation.

Maximum Scan Time Area Bits 0700 to 0707 contain the maximum scan time since start-up in 2-digit BCD (0.0 to 9.9 ms). The maximum scan time is reset when the PC begins operation.

Current Scan Time Area Bits 0708 to 0715 contain the current scan time in 2-digit BCD (0.0 to 9.9 ms).

Note The present and maximum scan time can be read out from the Programming Console with the SP16 or SP20. Refer to page 109 for details.

ATM1 Set Value Area Word 08 contains the set value in BCD for analog timer 1 as set with the adjustment screw on the front of the CPU (SP16 and SP20 only).

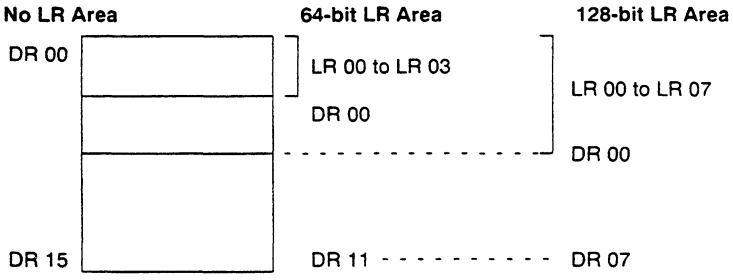
ATM12Set Value Area Word 09 contains the set value in BCD for analog timer 2 as set with the adjustment screw on the front of the CPU (SP16 and SP20 only).

3–2–5 LR Area

The LR area is used to pass data back and forth between PCs linked through a Link Adapter. To use the LR area, part of the DR area must be allocated as the LR area. Once this is done, each PC is allocated write bits in the LR area that it can write to so that the other PCs can read the data. All PCs in the link will thus write to certain LR words and read from the words written by the other PCs to transfer data back and forth. Each PC must write data only to the write area allocated to it: never to the write areas of other PCs. Allocation examples are given later in this section.

Each of the PCs connected to the same Link Adapter must be allocated the same size of LR area and the Programming Console must also be set to the same value. If the data areas are not identical in size, data will be lost.

Data Link Function LR areas must be allocated to enable data transfers through the Link Adapter. When LR areas are allocated, the size of the DR area is reduced as shown below. Segments of 64 or 128 bits allocated from the DR area are used for the LR area.



LR Allocation Allocating 0, 64 or 126 bits for the LR area corresponds to the following number of words for the transfer of data to the other PCs.

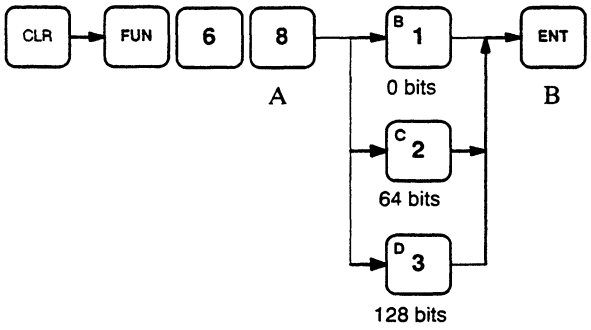
- 0 bits: LR area is not used.
- 64 bits: One write word (16 bits) for each CPU. DR area is reduced to DR 00 through DR 11.
- 128 bits: Two write words (32 bits) for each CPU. DR area is reduced to DR 00 through DR 07.

Caution If the size of the LR area is changed after programming operations have been started or the program code accesses illegal addresses, program trans-

fer cannot be performed and the message "????" will be displayed on the Programming Console.

LR Allocation Procedure

The size of the LR area is designated using the following key sequence. This operation must be performed in PROGRAM mode. This procedure allocates memory to each of units 0 through 3 simultaneously.



The following diagrams illustrate the Programming Console displays at the corresponding positions noted in the previous Key Sequence diagram.

A
```
0   LR NUMBER ?
  1.0 2.64 3.128
```

B
```
0   LR NUMBER OK
  1.0 2.64 3.128
```

LR Allocation Read

To check the size of the LR Area that has been allocated, use the following key sequence. This procedure can be performed in either RUN or PROGRAM mode.

When using the SP10-PRO01 Programming Console, the result is a display similar to the one shown below.

```
0 LR SIZE
          128
```

└— Indicates the LR area is set to 128 bits.

When using the SP10-PRO01-V1 Programming Console, the result is a display similar to the one shown below.

```
0 PC   :128
  ProCo:64
```

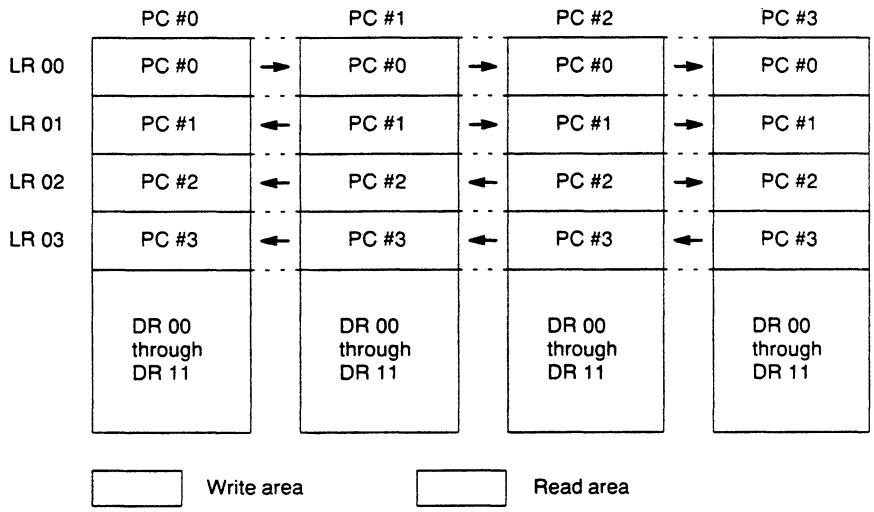
← Indicates the LR area is currently set to 128 bits in the PC.

└— Indicates the LR area is currently set to 64 bits in the Programming Console.

LR Area Allocation - 64 Bits

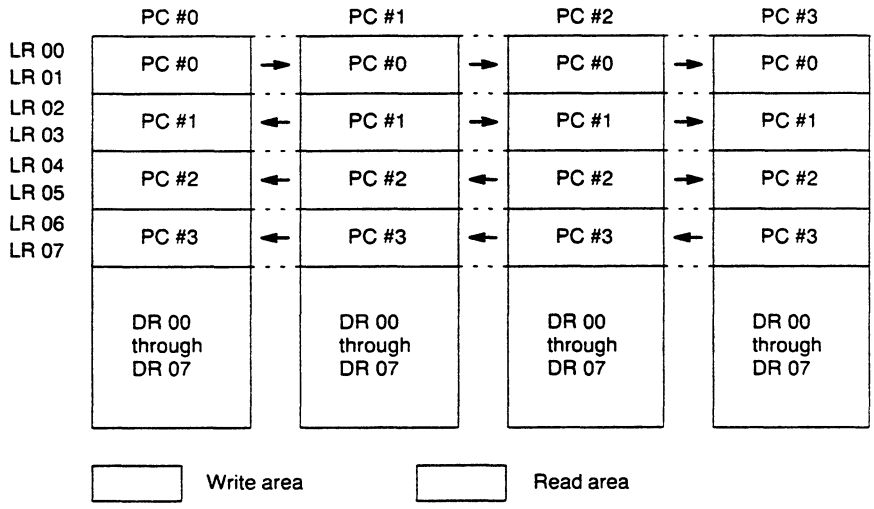
When the LR area is 64 bits, each PC is allocated one word (16 bits) of write area for its own use. Data transfer is illustrated in the following diagram.

When data is written to the write area of a PC, it is transferred to the same words in the LR areas of the other PCs linked through the Link Adapter.



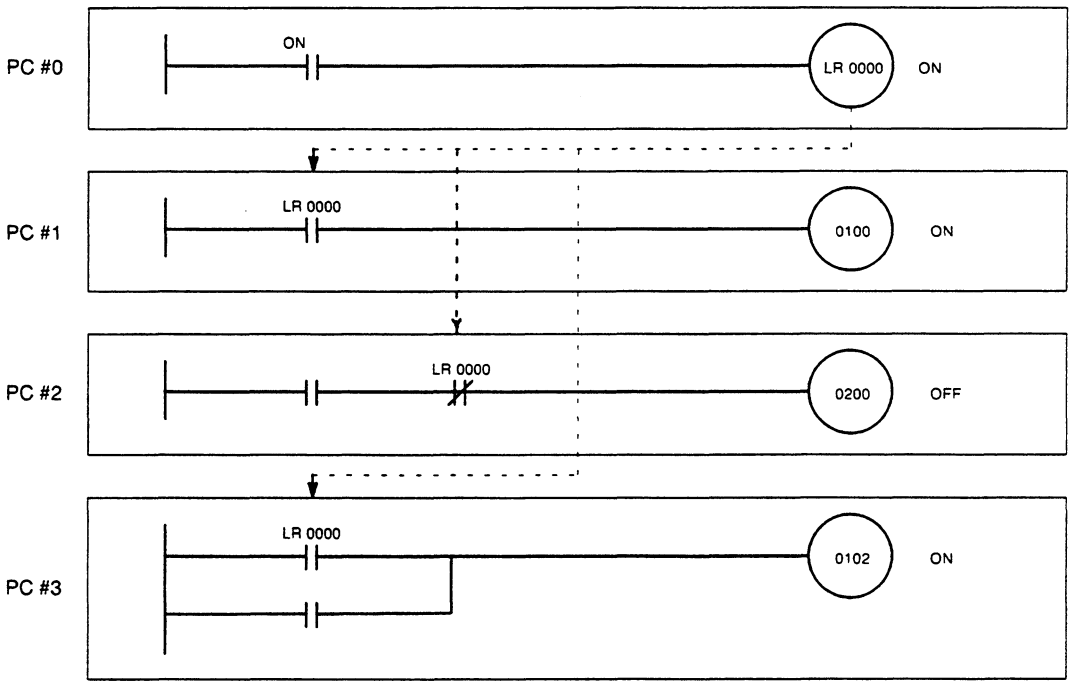
LR Area Allocation - 128 Bits

When the LR area is 128 bits in size, each PC is allocated two words (32 bits) of write area for its own use.



Data Link Communication Example

The following ladder diagrams illustrate an example of communications between linked PCs.



When LR 0000 of PC #0 is turned ON, LR 0000 of PC #1, #2 and #3 are also turned ON (OFF). LR 00 is the write area of PC #0, i.e., LR 00 of PC #1, #2 and #3 are used to read data written by PC #0.

3–2–6 DR Area

The DR area is used for data storage and manipulation. All data that is to be preserved for power interruptions, must be placed in this area. The size of the DR area depends on the size designated for the LR area (see *3–2–5 LR Area* for details).

3–2–7 TC (Timer/Counter) Area

The TC area is used to create and program timers and counters and holds the Completion Flags, set values (SV), and present values (PV) for all timers and counters. All of these are accessed through TC numbers ranging from TC 00 through TC 15. Each TC number is defined as either a timer or counter using one of the following instructions: TIM, TIMM(20), TIMH(21), ATIM(22), ATM1(25), ATM2(26), CNT, RDM(23), or CNTH(24). No prefix is required when using a TC number as a definer in a timer or counter instruction.

Once a TC number has been defined using one of these instructions, it cannot be redefined elsewhere in the program either using the same or a different instruction. If the same TC number is defined in more than one of these instructions or in the same instruction twice, an error will be generated. There are no restrictions on the order in which TC numbers can be used. TC num-

bers TC 11 through TC 15 (just TC 14 and TC 15 for the SP10) are assigned to specific instructions, as shown in the table below.

TC number	Instruction	Applicable PCs
TC 11	ANALOG TIMER 1, ATM1(25)	SP16, SP20
TC 12	ANALOG TIMER 2, ATM2(26)	SP16, SP20
TC 13	HIGH-SPEED COUNTER, CNTH(24)	SP16, SP20
TC 14	HIGH-SPEED TIMER, TIMH(21)	SP10, SP16, SP20
TC 15	ANALOG TIMER, ATIM(22)	SP10, SP16, SP20

Once defined, a TC number can be designated as an operand in one or more of certain instructions other than those listed above and can be used as many times as necessary in ladder instructions. When defined as a timer, a TC number designated as an operand takes a TIM prefix. The TIM prefix is used regardless of the timer instruction that was used to define the timer. Once defined as a counter, the TC number designated as an operand takes a CNT prefix. The CNT is also used regardless of the counter instruction that was used to define the counter.

TC numbers can be designated for operands that require bit data or for operands that require word data. When designated as an operand that requires bit data, the TC number accesses the Completion Flag of the timer or counter. When designated as an operand that requires word data, the TC number accesses a memory location that holds the PV of the timer or counter.

The TC area retains the SVs of both timers and counters during power interruptions. The PVs of timers are reset when PC operation is begun and when reset in interlocked program sections. Refer to *3–7–10 Interlock and Interlock Clear - IL(02) and ILC(03)* for details on timer and counter operation in interlocked program sections. The PVs of counters are not reset at these times.

Note that in programming "TIM 0" is used to designate three things: the Timer instruction defined with TC number 00, the Completion Flag for this timer, and the PV of this timer. The meaning in context should be clear, i.e., the first is always an instruction, the second is always a bit, and the third is always a word. The same is true of all other TC numbers prefixed with TIM or CNT.

3–3 The Programming Console

The Programming Console is used to program, monitor, and maintain the PCs. All programming is first input into the Programming Console and then transferred to the CPUs for execution or Memory Cards for storage.

The Programming Console keys are divided into several sections for ease in operation. The gray keys are used in combination with the white numeric keys to designate instructions, operands, and Programming Console functions. The yellow keys are used to designate Programming Console operations. The red Clear Key is used to clear the display and cancel Programming Console operations. Key functions are described in detail in the next section.

3–3–1 The Keyboard

Key		Function
FUN	Function Key	Designates instructions via function codes or designates Programming Console functions.
NOT	NOT Key	Pressed after the Load, AND, or OR Key to designate a normally closed condition with the LOAD, AND, or OR instructions.
SHIFT	Shift Key	Designates the upper function on keys that have two functions. Used with the CH/* Key, the Bit/Constant Key, or Numeric Keys 0 through 5.
AND	AND Key	Inputs an AND instruction.

Key		Function
OR	OR Key	Inputs an OR instruction.
LD	Load Key	Inputs a LOAD instruction when pressed alone or an OR LOAD or AND LOAD instruction when pressed after the OR or AND Key.
OUT	Output Key	Inputs an OUTPUT instruction when pressed alone or an OUTPUT NOT instruction if pressed before the NOT key.
TIM	Timer Key	Inputs a TIMER instruction.
CNT	Counter Key	Inputs a COUNTER instruction.
LR	Link Bit Key	Indicates an LR (link) bit.
DR	Data Bit Key	Indicates a DR (data) bit.
CH / *	Word/Indirect Address Key	Indicates an indirect DR address when pressed without the Shift Key and designates a word address when pressed after the Shift Key.
CONT / #	Bit/Constant Key	Indicates a bit or a constant depending on whether the Shift Key is used.
CHG	Change Key	Pressed to change the content of a memory address.
DEL	Delete Key	Pressed to delete an instruction in combination with the Up Key.
INS	Insert Key	Pressed to insert an instruction in combination with the Down Key.
CLR	Clear Key	Normally cancels operations and resets the Programming Console.
ENT	Enter Key	Inputs instructions, set values, and other data.
▲	Up Key	Pressed when reading programs to scroll the program memory address or pressed to delete instructions (see Delete Key).
▼	Down Key	Pressed when reading programs to scroll the program memory address or pressed to insert instructions (see Insert Key).
MON	Monitor Key	Pressed to monitor bit status or word content.
A 0 to F 5 6 to 9	Numeric keys	Input numeric values, addresses, and other data. The Shift key is pressed before the 0 through 5 Keys to input hexadecimal numerals A through F.

3–3–2 PC Modes

There are two PC operating modes that are set from the Programming Console: RUN and PROGRAM.

RUN mode is used for normal program execution once the program has been input. In RUN mode, input terminal status is read into the PC and output terminals are updated according to program execution results.

PROGRAM mode is used for programming operations to input and debug the program when setting up the control system and for data access and manipulation once a control system is running. The program is not executed in PROGRAM mode.

Startup Mode

When the PC is turned on with the Programming Console attached, the mode switch on the Programming Console will determine the initial operating mode.

If the Programming Console is not attached, the PC will always start in RUN mode and the program will be executed immediately.

If the Programming Console is attached after the PC is already turned on, the current mode will continue regardless of the setting of the Programming Console mode switch.

Caution Always confirm that the Programming Console is in PROGRAM mode when turning on the PC with a Programming Console connected unless another mode is desired for a specific purpose. If the Programming Console is in RUN mode when PC power is turned on, any program in Program Memory will be executed, possibly causing a PC-controlled system to begin operation. If the START input on the CPU Power Supply Unit is ON and there is no device connected to the CPU, ensure that commencing operation is safe and appropriate before turning on the PC.

DANGER! Do not leave the Programming Console connected to the PC by an extension cable when in RUN mode. Noise detected via the extension cable can enter the PC, affecting the program and thus the controlled system.

3–4 Basic Programming

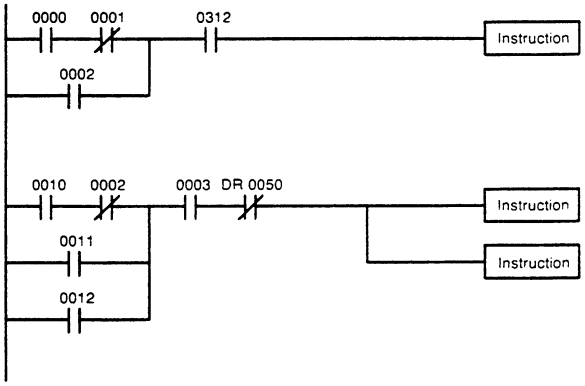
3–4–1 Terminology

There are basically two types of instructions used in ladder-diagram programming: ladder instructions that correspond to the conditions on the ladder diagram and right-hand instructions that are used on the right side of the ladder diagram and are controlled by the ladder instructions. Ladder instructions are used in instruction form only when converting a program to mnemonic code.

Most instructions have at least one or more operands associated with them. Operands indicate or provide the data on which an instruction is to be performed. These are sometimes input as the actual numeric values, but are usually the addresses of words or bits that contain the data to be used. For instance, a MOVE instruction that has word 00 designated as the source operand will move the contents of word 00 to some other location. The other location is also designated as an operand. A bit whose address is designated as an operand is called an operand bit; a word whose address is designated as an operand is called an operand word. If the actual value is entered as a constant, it is preceded by # to indicate that it is not an address.

Basic Ladder Diagram

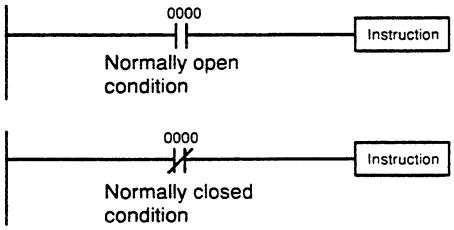
A ladder diagram consists of one line running down the left side with lines branching off to the right. The line on the left is called the bus bar; the branching lines, instruction lines or rungs. (Sometimes a right bus bar is also drawn.) Along the instruction lines are placed conditions that lead to other instructions on the right side. The logical combinations of these conditions on the ladder determine when and how the right-hand instructions are executed. A simple ladder diagram is shown below.



As shown in the diagram above, instruction lines can branch apart and they can join back together. The vertical pairs of lines are called conditions. Conditions without diagonal lines through them are called normally open conditions and correspond to a LOAD, AND, or OR instruction. The conditions with diagonal lines through them are called normally closed conditions and correspond to a LOAD NOT, AND NOT, or OR NOT instruction. The number above each condition indicates the operand bit for the condition. It is the status of the bit associated with each condition that determines the execution condition for following instructions. The way the operation of each of the instructions corresponds to a condition is described below. Before we consider these, however, there are some basic terms that must be explained.

Normally Open and Normally Closed Conditions

Each condition in a ladder diagram is either ON or OFF depending on the status of the operand bit that has been assigned to it. A normally open condition is ON if the operand bit is ON; OFF if the operand bit is OFF. A normally closed condition is ON if the operand bit is OFF; OFF if the operand bit is ON. Generally speaking, you use a normally open condition when you want something to happen when a bit is ON, and a normally closed condition when you want something to happen when a bit is OFF.



Execution Conditions

In ladder diagram programming, the logical combination of ON and OFF conditions before an instruction determines the compound condition under which the instruction is executed. This condition, which is either ON or OFF, is called the execution condition for the instruction. All instructions other than LOAD instructions have execution conditions.

Operand Bits

The operands designated for any of the ladder instructions can be any I/O, work, DR, or dedicated bit. This means that the conditions in a ladder diagram can be determined by I/O status, flag status, status contained in work bits, timer/counter status, etc.

Logic Blocks

The way that conditions correspond to what instructions is determined by the relationship between the conditions within the instruction lines that connect them. Any group of conditions that go together to create a logic result is called a logic block. Although ladder diagrams can be written without actually

analyzing individual logic blocks, understanding logic blocks is necessary for efficient programming and is essential when programs are to be input in mnemonic code.

3–4–2 Mnemonic Code

The ladder diagram cannot be directly input into the PC via a Programming Console. To input from a Programming Console, it is necessary to convert the ladder diagram to mnemonic code. The mnemonic code provides exactly the same information as the ladder diagram, but in a form that can be typed directly into the PC. Actually you can program directly in mnemonic code, although it in not recommended for beginners or for complex programs. Also, the program is stored in memory in mnemonic form.

Because of the importance of mnemonic code, we will introduce and describe the mnemonic code along with the ladder diagram.

Program Memory Structure

The program is input into addresses in Program Memory. Addresses in Program Memory are slightly different to those in other memory areas because each address does not necessarily hold the same amount of data. Rather, each address holds one instruction and all of the definers and operands (described in more detail later) required for that instruction. Because some instructions require one word, while others require up to five words, Program Memory addresses can be from one to five words long.

Program Memory addresses start at 000 and run until the capacity of Program Memory has been exhausted (144 words). The first word at each address defines the instruction. Any definers used by the instruction are also contained in the first word. Also, if an instruction requires only a single bit operand (with no definer), the bit operand is also programmed on the same line as the instruction. The rest of the words required by an instruction contain the operands that specify what data is to be used. When converting to mnemonic code, all but ladder diagram instructions are written in the same form, one word to a line, just as they appear in the ladder diagram symbols. An example of mnemonic code is shown below. The instructions used in it are described later in the manual.

Address	Instruction	Operands	
000	LD	DR	0001
001	AND		0001
002	OR		0002
003	LD NOT		0100
004	AND		0101
005	AND LD		0102
006	MOV(30)		
			00
		DR	00
007	CMP(32)		
		#	0100
		DR	00

The address and instruction columns of the mnemonic code table are filled in for the instruction word only. For all other lines, the left two columns are left blank. If the instruction requires no definer or bit operand, the operand column is left blank for first line. It is a good idea to cross through any blank data column spaces (for all instruction words that do not require data) so that the data column can be quickly scanned to see if any addresses have been left out.

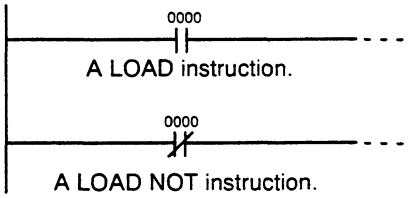
When programming, addresses are automatically displayed and do not have to be input unless for some reason a different location is desired for the instruction. When converting to mnemonic code, it is best to start at Program Memory address 000 unless there is a specific reason for starting elsewhere.

3–4–3 Ladder Instructions

Ladder instructions are those instructions that correspond to the conditions on the ladder diagram. Ladder instructions, either independently or in combination with the logic block instructions described next, form the execution conditions upon which the execution of all other instructions are based.

LOAD and LOAD NOT

The first condition that starts any logic block within a ladder diagram corresponds to a LOAD or LOAD NOT instruction. Each of these instructions requires one line of mnemonic code. "Instruction" is used as a dummy instruction in the following examples and could be any of the right-hand instructions described later in this manual.

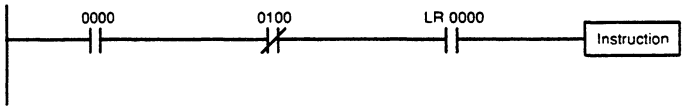


Address	Instruction	Operands
000	LD	0000
001	Instruction	
002	LD NOT	0000
003	Instruction	

When this is the only condition on the instruction line, the execution condition for the instruction at the right is ON when the condition is ON. For the LOAD instruction (i.e., a normally open condition), the execution condition would be ON when bit 0000 was ON; for the LOAD NOT instruction (i.e., a normally closed condition), it would be ON when bit 0000 was OFF.

AND and AND NOT

When two or more conditions lie in series on the same instruction line, the first one corresponds to a LOAD or LOAD NOT instruction; and the rest of the conditions, to AND or AND NOT instructions. The following example shows three conditions which correspond in order from the left to a LOAD, an AND NOT, and an AND instruction. Again, each of these instructions requires one line of mnemonic code.



Address	Instruction	Operands
000	LD	0000
001	AND NOT	0100
002	AND	LR 0000
003	Instruction	

The instruction would have an ON execution condition only when all three conditions are ON, i.e., when bit 0000 was ON, bit 0100 was OFF, and LR 0000 was ON.

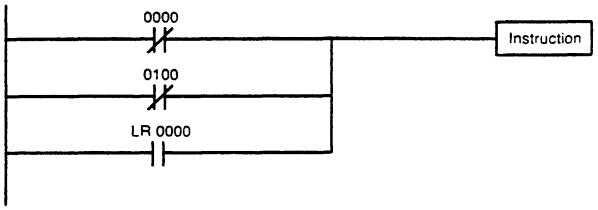
AND instructions in series can be considered individually, with each taking the logical AND of the execution condition (i.e., the total of all conditions up to that point) and the status of the AND instruction's operand bit. If both of these are ON, an ON execution condition will be produced for the next instruction. If either is OFF, the result will also be OFF. The execution condition for the first AND instruction in a series is the first condition on the instruction line.

Each AND NOT instruction in a series would take the logical AND between its execution condition and the inverse of its operand bit.

OR and OR NOT

When two or more conditions lie on separate instruction lines running in parallel and then joining together, the first condition corresponds to a LOAD or

LOAD NOT instruction; the rest of the conditions correspond to OR or OR NOT instructions. The following example shows three conditions which correspond in order from the top to a LOAD NOT, an OR NOT, and an OR instruction. Again, each of these instructions requires one line of mnemonic code.



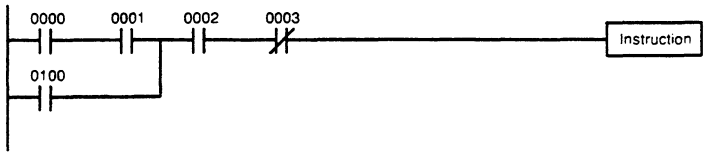
Address	Instruction	Operands
000	LD	0000
001	OR NOT	0100
002	OR	LR 0000
003	Instruction	

The instruction would have an ON execution condition when any one of the three conditions was ON, i.e., when bit 0000 was OFF, when bit 0100 was OFF, or when LR 0000 was ON.

OR and OR NOT instructions can be considered individually, each taking the logical OR between its execution condition and the status of the OR instruction's operand bit. If either one of these were ON, an ON execution condition would be produced for the next instruction.

Combining AND and OR Instructions

When AND and OR instructions are combined in more complicated diagrams, they can sometimes be considered individually, with each instruction performing a logic operation on the execution condition and the status of the operand bit. The following is one example. Study this example until you are convinced that the mnemonic code follows the same logic flow as the ladder diagram.



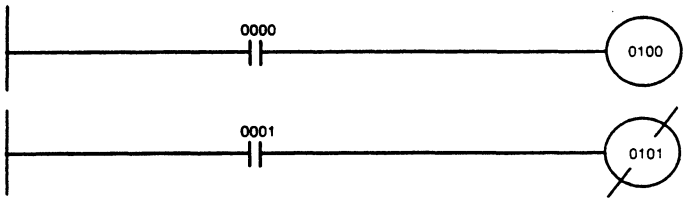
Address	Instruction	Operands
000	LD	0000
001	AND	0001
002	OR	0100
003	AND	0002
004	AND NOT	0003
005	Instruction	

Here, an AND is taken between the status of bit 0000 and that of bit 0001 to determine the execution condition for an OR with the status of bit 0100. The result of this operation determines the execution condition for an AND with the status of bit 0002, which in turn determines the execution condition for an AND with the inverse (i.e., and AND NOT) of the status of bit 0003.

In more complicated diagrams, however, it is necessary to consider logic blocks before an execution condition can be determined for the final instruction, and that's where AND LOAD and OR LOAD instructions are used. Before we consider more complicated diagrams, however, we'll look at the instructions required to complete a simple "input-output" program.

3–4–4 OUTPUT and OUTPUT NOT

The simplest way to output the results of combining execution conditions is to output it directly with the OUTPUT and OUTPUT NOT. These instructions are used to control the status of the designated operand bit according to the execution condition. With the OUTPUT instruction, the operand bit will be turned ON as long as the execution condition is ON and will be turned OFF as long as the execution condition is OFF. With the OUTPUT NOT instruction, the operand bit will be turned ON as long as the execution condition is OFF and turned OFF as long as the execution condition is ON. These appear as shown below. In mnemonic code, each of these instructions requires one line.



Address	Instruction	Operands
000	LD	0000
001	OUT	0100

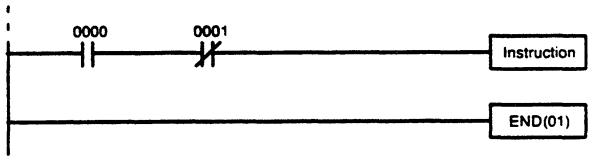
Address	Instruction	Operands
000	LD	0001
001	OUT NOT	0101

In the above examples, bit 0100 will be ON as long as bit 0000 is ON and bit 0101 will be OFF as long as bit 0001 is ON. Here, bit 0000 and bit 0001 are input bits and bit 0100 and bit 0101 are output bits, i.e., the signals coming in through inputs 0 and 1 are controlling outputs 0 and 1, respectively.

The length of time that a bit is ON or OFF can be controlled by combining the OUTPUT or OUTPUT NOT instruction with Timer instructions. Refer to Examples under *3–7–14 Timer - TIM* for details.

3–4–5 The END Instruction

The last instruction required to complete a simple program is the END instruction. When the CPU scans the program, it executes all instruction up to the first END instruction before returning to the beginning of the program and beginning execution again. Although an END instruction can be placed at any point in a program, which is sometimes done when debugging, no instructions past the first END instruction will be executed until it is removed. The number following the END instruction in the mnemonic code is its function code, which is used when inputting most instructions into the PC. These are described later. The END instruction requires no operands and no conditions can be placed on the same instruction line with it.



Program execution ends here.

Address	Instruction	Operands
000	LD	0000
001	AND NOT	0001
002	Instruction	
003	END(01)	---

If there is no END instruction anywhere in the program, the program will not be executed at all.

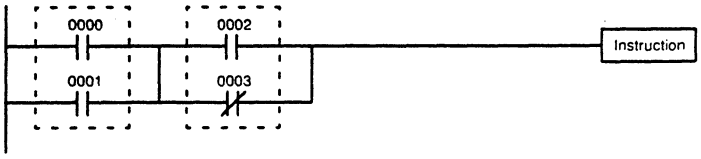
Now you have all of the instructions required to write simple input-output programs. Before we finish with ladder diagram basics and go on to inputting the program into the PC, let's look at logic block instructions (AND LOAD and OR LOAD), which are sometimes necessary even with simple diagrams.

3–4–6 Logic Block Instructions

Logic block instructions do not correspond to specific conditions on the ladder diagram; rather, they describe relationships between logic blocks. The AND LOAD instruction logically ANDs the execution conditions produced by two logic blocks. The OR LOAD instruction logically ORs the execution conditions produced by two logic blocks.

AND LOAD

Although simple in appearance, the diagram below requires an AND LOAD instruction.



Address	Instruction	Operands
000	LD	0000
001	OR	0001
002	LD	0002
003	OR NOT	0003
004	AND LD	---

The two logic blocks are indicated by dotted lines. Studying this example shows that an ON execution condition will be produced when: either of the conditions in the left logic block is ON (i.e., when either bit 0000 or bit 0001 is ON), and when either of the conditions in the right logic block is ON (i.e., when either bit 0002 is ON or bit 0003 is OFF).

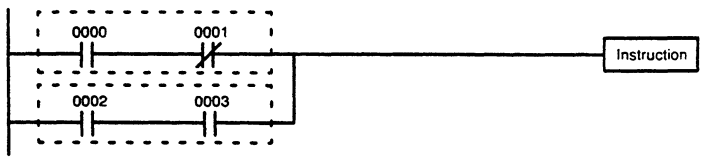
The above ladder diagram cannot be converted to mnemonic code using AND and OR instructions alone. If an AND between bit 0002 and the results of an OR between bit 0000 and bit 0001 is attempted, the OR NOT between bit 0002 and bit 0003 is lost and the OR NOT ends up being an OR NOT between just bit 0003 and the result of an AND between bit 0002 and the first OR. What we need is a way to do the OR (NOT)'s independently and then combine the results.

To do this, we can use the LOAD or LOAD NOT instruction in the middle of an instruction line. When LOAD or LOAD NOT is executed in this way, the current execution condition is saved in special buffers and the logic process is begun over. To combine the results of the current execution condition with that of a previous "unused" execution condition, an AND LOAD or an OR LOAD instruction is used. Here "LOAD" refers to loading the last unused execution condition. An unused execution condition is produced by using the LOAD or LOAD NOT instruction for any but the first condition on an instruction line.

Analyzing the above ladder diagram in terms of mnemonic instructions, the condition for bit 0000 is a LOAD instruction and the condition below it is an OR instruction between the status of bit 0000 and that of bit 0001. The condition at bit 0002 is another LOAD instruction and the condition below is an OR NOT instruction, i.e., an OR between the status of bit 0002 and the inverse of the status of bit 0003. To arrive at the execution condition for the instruction at the right, the logical AND of the execution conditions resulting from these two blocks would have to be taken. AND LOAD does this. The mnemonic code for the ladder diagram is shown in the previous table. The AND LOAD instruction requires no operands of its own, because it operates on previously determined execution conditions. Here too, dashes are used to indicate that no operand needs to be designated or input.

OR LOAD

The following diagram requires an OR LOAD instruction between the top logic block and the bottom logic block. An ON execution condition would be produced for the instruction at the right either when bit 0000 is ON and bit 0001 is OFF or when bit 0002 and bit 0003 are both ON. The operation of the mnemonic code for the OR LOAD instruction is exactly the same as those for a AND LOAD instruction except that the current execution condition is ORed with the last unused execution condition.



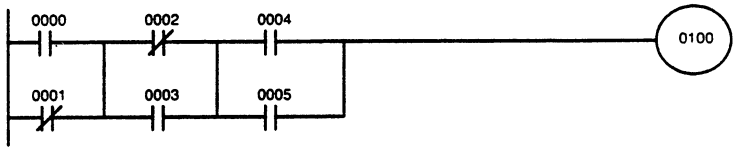
Address	Instruction	Operands
000	LD	0000
001	AND NOT	0001
002	LD	0002
003	AND	0003
004	OR LD	---

Naturally, some diagrams will require both AND LOAD and OR LOAD instructions.

Logic Block Instructions in Series

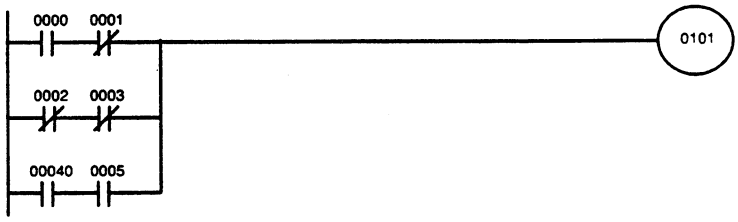
To code diagrams with logic block instructions in series, the diagram must be divided into logic blocks. Each block is coded as normal using a LOAD instruction to code the first condition, and then AND LOAD or OR LOAD is used to logically combine the blocks. First input the first two logic blocks and then the logic block instruction to combine the results. Then input each additional logic block along with the logic block instruction required to combine it with the previous result. Examples are given next.

The following diagram requires AND LOAD to be converted to mnemonic code because three pairs of parallel conditions lie in series.



Address	Instruction	Operands
000	LD	0000
001	OR NOT	0001
002	LD NOT	0002
003	OR	0003
004	AND LD	—
005	LD	0004
006	OR	0005
007	AND LD	—
008	OUT	0100

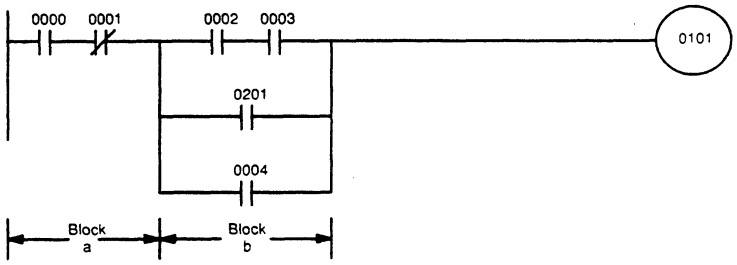
The following diagram requires OR LOAD instructions to be converted to mnemonic code because three pairs of conditions in series lie in parallel to each other. The first of each pair of conditions is converted to LOAD with the assigned bit operand and then ANDed with the other condition. The first two blocks are coded first, followed by OR LOAD, the last block, and another OR LOAD.



Address	Instruction	Operands
000	LD	0000
001	AND NOT	0001
002	LD NOT	0002
003	AND NOT	0003
004	OR LD	—
005	LD	0004
006	AND	0005
007	OR LD	—
008	OUT	0101

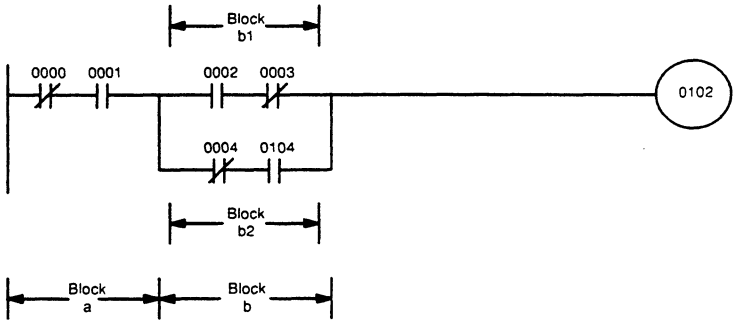
Combining AND LOAD and OR LOAD

AND LOAD and OR LOAD can naturally be used in the same section of program. The following diagram contains only two logic blocks as shown. It is not necessary to further separate block b components, because it can coded directly using only AND and OR.



Address	Instruction	Operands
000	LD	0000
001	AND NOT	0001
002	LD	0002
003	AND	0003
004	OR	0201
005	OR	0004
006	AND LD	—
007	OUT	0101

Although the following diagram is similar to the one above, block b in the diagram below cannot be coded without separating it into two blocks combined with OR LOAD. Here the three logic blocks are coded first followed by the two logic block instructions required to combine them. When coding the logic block instructions together at the end of the logic blocks they are combining, they must, as shown below, be coded in reverse order, i.e., the logic block instruction for the last two blocks is coded first, followed by the one to combine the execution condition resulting from the first logic block instruction and the execution condition of the logic block third from the end, and on back to the first logic block that is being combined.



Address	Instruction	Operands
000	LD NOT	0000
001	AND	0001
002	LD	0002
003	AND NOT	0003
004	LD NOT	0004
005	AND	0104
006	OR LD	—
007	AND LD	—
008	OUT	0102

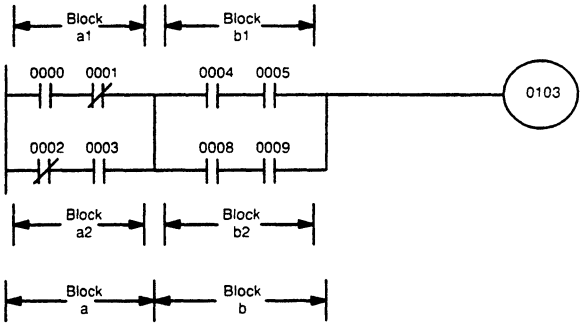
Complicated Diagrams

When determining what logic block instructions will be required to code a diagram, it is sometimes necessary to break the diagram into large blocks and then continue breaking the large blocks down until logic blocks that can be coded without logic block instructions have been formed. These blocks are then coded, combining the small blocks first, and then combining the larger blocks. Either AND LOAD or OR LOAD is used to combine the blocks, i.e., AND LOAD or OR LOAD always combines the last two execution conditions that existed, regardless of whether the execution conditions resulted from a single condition, from logic blocks, or from previous logic block instructions.

When working with complicated diagrams, blocks will ultimately be coded starting at the top left and moving down before moving across. This will generally mean that, when there might be a choice, OR LOAD will be coded before AND LOAD.

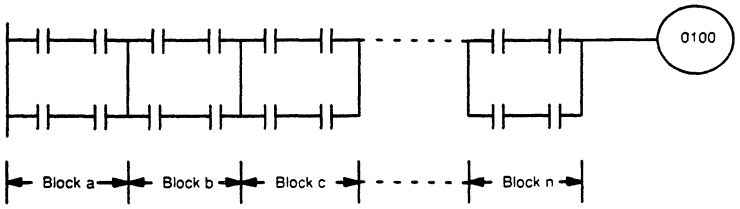
The following diagram must be broken down into two blocks and each of these then broken into two blocks before it can be coded. As shown below, blocks a and b require an AND LOAD. Before AND LOAD can be used, how-

ever, OR LOAD must be used to combine the top and bottom blocks on both sides, i.e., to combine a1 and a2; b1 and b2.

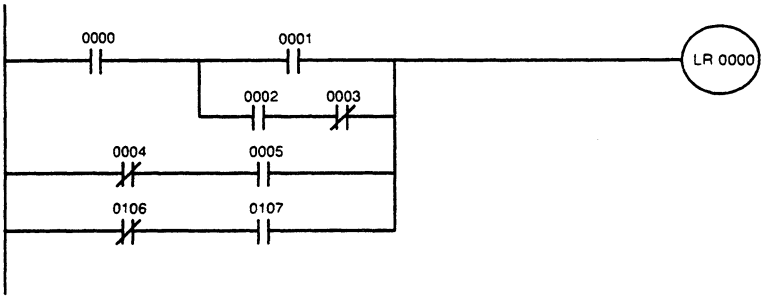


Address	Instruction	Operands	
000	LD	0000	
001	AND NOT	0001	
002	LD NOT	0002	
003	AND	0003	
004	OR LD	—	Blocks a1 and a2
005	LD	0004	
006	AND	0005	
007	LD	0008	
008	AND	0009	
009	OR LD	—	Blocks b1 and b2
010	AND LD	—	Blocks a and b
011	OUT	0103	

The following type of diagram can be coded easily if each block is coded in order: first top to bottom and then left to right. In the following diagram, blocks a and b would be combined using AND LOAD as shown above, and then block c would be coded and a second AND LOAD would be used to combined it with the execution condition from the first AND LOAD. Then block d would be coded, a third AND LOAD would be used to combine the execution condition from block d with the execution condition from the second AND LOAD, and so on through to block n.

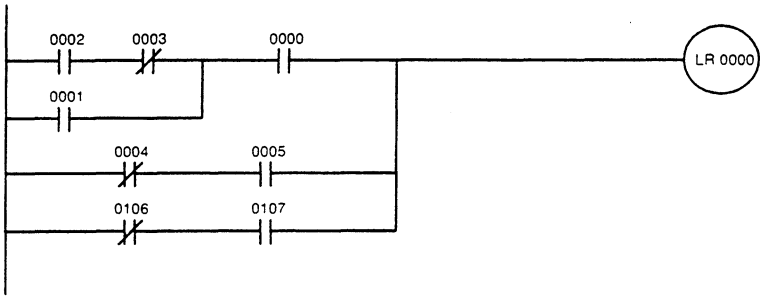


The following diagram requires an OR LOAD followed by an AND LOAD to code the top of the three blocks, and then two more OR LOADs to complete the mnemonic code.



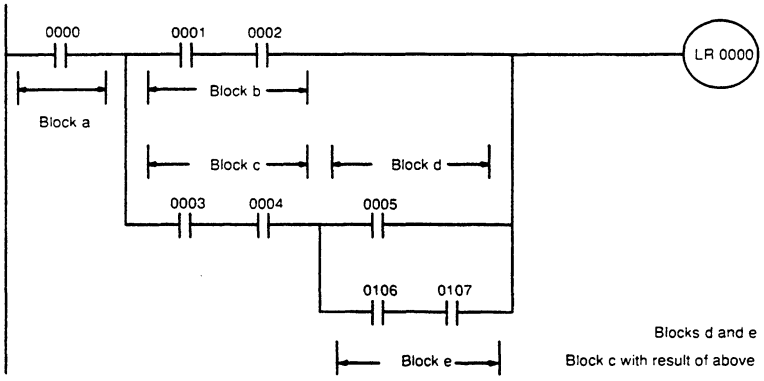
Address	Instruction	Operands
000	LD	0000
001	LD	0001
002	LD	0002
003	AND NOT	0003
004	OR LD	--
005	AND LD	--
006	LD NOT	0004
007	AND	0005
008	OR LD	--
009	LD NOT	0106
010	AND	0107
011	OR LD	--
012	OUT	LR 0000

Although the program will execute as written, this diagram could be drawn as shown below to eliminate the need for the first OR LOAD and the AND LOAD, simplifying the program and saving memory space.



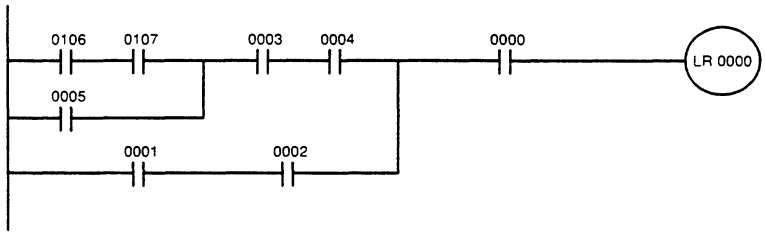
Address	Instruction	Operands
000	LD	0002
001	AND NOT	0003
002	OR	0001
003	AND	0000
004	LD NOT	0004
005	AND	0005
006	OR LD	--
007	LD NOT	0106
008	AND	0107
009	OR LD	--
010	OUT	LR 0000

The following diagram requires five blocks, which here are coded in order before using OR LOAD and AND LOAD to combine them starting from the last two blocks and working backward. The OR LOAD at program address 008 combines blocks d and e, the following AND LOAD combines the resulting execution condition with that of block c, etc.



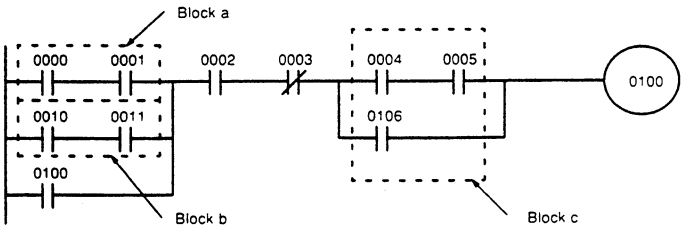
Address	Instruction	Operands
000	LD	0000
001	LD	0001
002	AND	0002
003	LD	0003
004	AND	0004
005	LD	0005
006	LD	0106
007	AND	0107
008	OR LD	--
009	AND LD	--
010	OR LD	--
011	AND LD	--
012	OUT	LR 0000

Again, this diagram can be redrawn as follows to simplify program structure and coding and to save memory space.

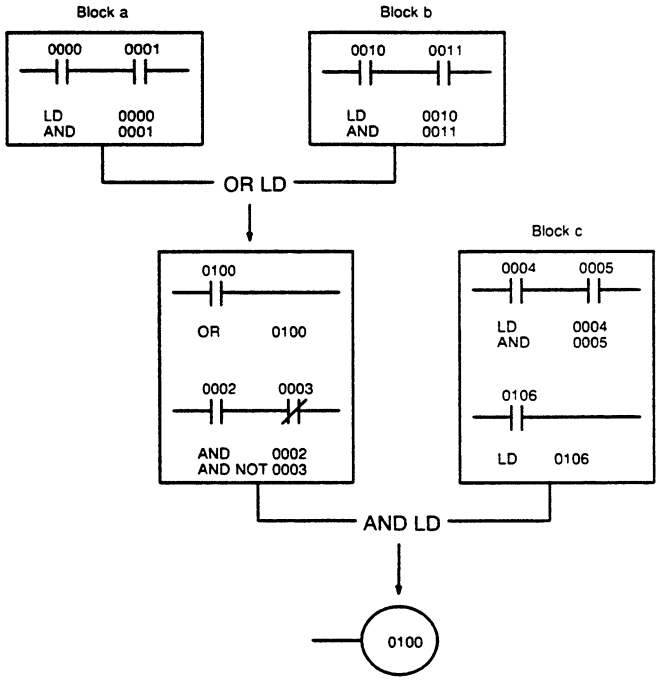


Address	Instruction	Operands
000	LD	0106
001	AND	0107
002	OR	0005
003	AND	0003
004	AND	0004
005	LD	0001
006	AND	0002
007	OR LD	--
008	AND	0000
009	OUT	LR 0000

The next and final example may at first appear very complicated but can be coded using only two logic block instructions. The diagram appears as follows:



The first logic block instruction is used to combine the execution conditions resulting from blocks a and b, and the second one is to combine the execution condition of block c with the execution condition resulting from the normally closed condition assigned bit 0003. The rest of the diagram can be coded with OR, AND, and AND NOT instructions. The logical flow for this and the resulting code are shown below.

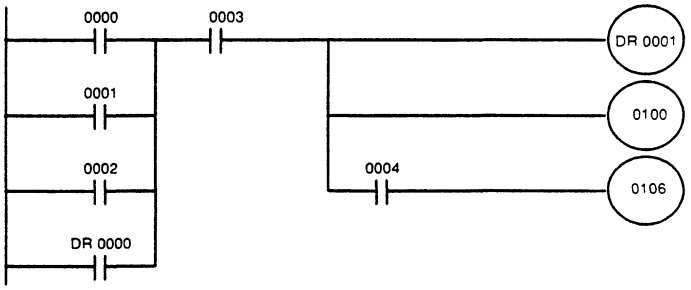


Address	Instruction	Operands
000	LD	0000
001	AND	0001
002	LD	0010
003	AND	0011
004	OR LD	--
005	OR	0100
006	AND	0002
007	AND NOT	0003
008	LD	0004
009	AND	0005
010	OR	0106
011	AND LD	--
012	OUT	0100

3–4–7 Coding Multiple Right-hand Instructions

If there is more than one right-hand instruction executed with the same execution condition, they are coded consecutively following the last condition on

the instruction line. In the following example, the last instruction line contains one more condition that corresponds to an AND with bit 0004.



Address	Instruction	Operands
000	LD	0000
001	OR	0001
002	OR	0002
003	OR	DR 0000
004	AND	0003
005	OUT	DR 0001
006	OUT	0100
007	AND	0004
008	OUT	0106

3–5 Inputting the Program

Once a program is written in mnemonic code, it can be input directly into the PC from a Programming Console. Mnemonic code is keyed into Program Memory addresses from the Programming Console. Checking the program involves a syntax check to see that the program has been written according to syntax rules. Once syntax errors are corrected, a trial execution can begin and, finally, correction under actual operating conditions can be made.

The operations required to input a program are explained below. Operations to modify programs that already exist in memory are also provided in this section, as well as the procedure to obtain the current scan time.

Before starting to input a program, check to see whether there is a program already loaded. If there is a program already loaded that you do not need, clear it first using the program memory clear key sequence, then input the new program. If you need the previous program, be sure to check it with the program check key sequence and correct it as required.

3–5–1 Initial Programming Console Operation

When operating the Programming Console for the first time, use the following procedure:

1, 2, 3..

1. Connect the Programming Console to the PC or to the Link Adapter. Make sure that the Programming Console is securely connected; improper connection may inhibit operation.
2. Set the mode selector of the Programming Console to PRGM (PROGRAM) mode.
3. Turn on the PC.
4. The backlight on the display of the Programming Console will light, and "<PROGRAM> PASSWORD!" will be displayed.
5. Press CLR and then MON (the password). "<PROGRAM> BZ" will be displayed.
6. If more then one PC is connected via a Link Adapter, designate the PC.
7. Clear memory.

Each of these operations from entering the password on is described in detail in the following subsections. All operations should be done in PROGRAM mode unless otherwise noted.

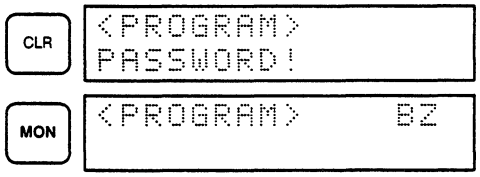
Password

To gain access to the PC's programming functions, you must first enter the password. The password prevents unauthorized access to the program.

The PC prompts you for a password when PC power is turned on or, if PC power is already on, after the Programming Console has been connected to

the PC. To gain access to the system when the "Password!" message appears, press CLR and then MON. Then press CLR to clear the display.

If the Programming Console is connected to the PC when PC power is already on, the first display below will indicate the mode the PC was in before the Programming Console was connected. **Ensure that the PC is in PROGRAM mode before you enter the password.** When the password is entered, the PC will shift to the mode set on the mode switch, causing PC operation to begin if the mode is set to RUN. The mode can be changed to RUN with the mode switch after entering the password.



Indicates the mode set by the mode selector switch.

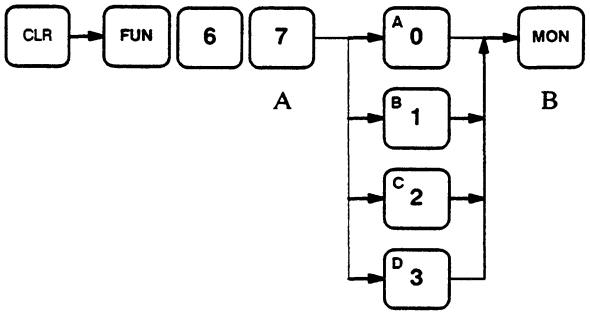
Buzzer

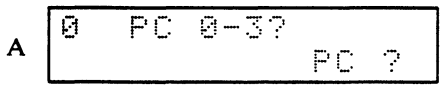
Immediately after the password is input or anytime immediately after the mode has been changed, SHIFT and then the 1 key can be pressed to turn on and off the buzzer that sounds when Programming Console keys are pressed. If BZ is displayed in the upper right corner, the buzzer is operative. If BZ is not displayed, the buzzer is not operative.

This buzzer also will also sound whenever an error occurs during PC operation. Buzzer operation for errors is not affected by the above setting.

3–5–2 Designating the PC

When more than one PC is connected via a Link Adapter and the Programming Console is connected to the Link Adapter, you must designate the PC you are accessing. Use the following key sequence to specify the number of the desired PC. The PC can be designated in either PROGRAM or RUN mode.





The PC's operation or operation mode is not affected by changing the PC designation. When the mode switch of the Programming Console and the operation mode of the PC being monitored are identical, the following message is displayed. The number in the top left corner indicates the number of the PC being monitored, in this case PC #1.

B
```
1-000
```

When the mode switch of the Programming Console and the operation mode of the PC being monitored are not identical, a message like the following one will be displayed.

B
```
<RUN>
MODE SET ERR
```

In this example, the message indicates that the Programming Console is set to PRGM (program) mode, and that PC #1 is set to RUN mode. To clear the error and reset the corresponding alarm, turn the Programming Console mode to RUN and then change it back to PRGM mode. PC #1 will change to PRGM mode.

Note If there is a communication error, the display will read "COMM ERR".

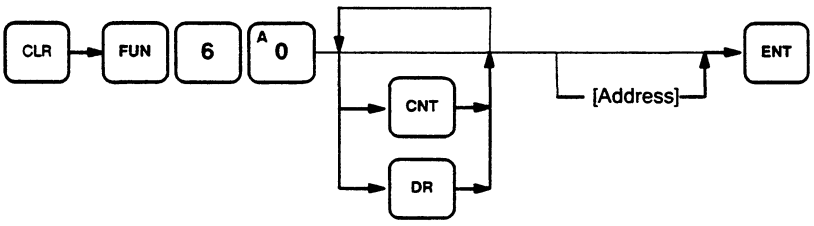
3–5–3 Clearing Memory

Using the Memory Clear operation it is possible to clear part or all of the Program Memory, work bits, and the DR and TC areas. Unless otherwise specified, the clear operation will clear all of the above memory areas, as well as the contents of the Programming Console's memory.

Before beginning to program for the first time or when installing a new program, clear all memory areas. Before clearing memory, check to see if a program is already loaded that you need. If you need the program, clear only the memory areas that you do not need, and be sure to check the existing program with the program check key sequence before using it. The check sequence is provided later in this section. To clear all memory areas, press CLR until all zeros are displayed, and then input the keystrokes given in the top line of the following key sequence. The branch lines shown in the sequence are used only when performing a partial memory clear, which is described below. When program memory has been cleared NOP(00) instructions (00) are written to the entire area of memory. These instructions perform nothing.

Memory can be cleared in PROGRAM mode only.

Key Sequence



All Clear

The following procedure is used to clear memory completely.

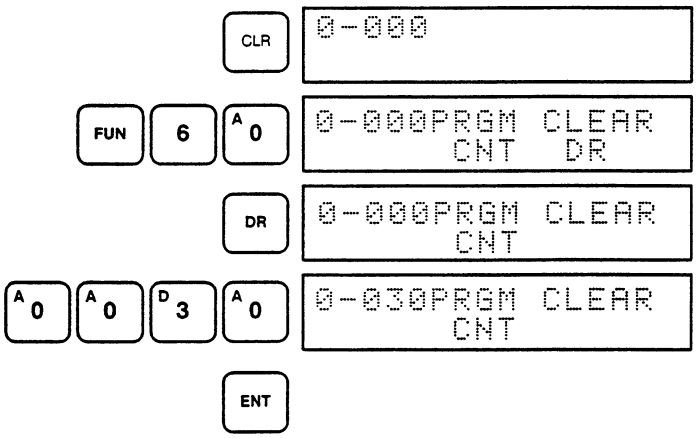
Partial Clear

It is possible to retain the data in specified areas or part of the Program Memory. To retain the data in the TC and/or DR areas, press the appropriate

key after entering the function number 60. If not specified for retention, both areas will be cleared. CNT is used for the entire TC area. The display will show those areas that will be cleared.

It is also possible to retain a portion of the Program Memory from the first memory address to a specified address. After designating the data areas, DR and/or CNT, to be retained, specify the first Program Memory address to be cleared. For example, to leave addresses 000 to 029 untouched, but to clear addresses from 030 to the end of Program Memory, input 030.

As an example, to leave the DR area untouched and retain Program Memory addresses 000 through 029, input as follows:



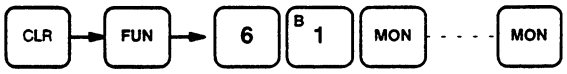
3–5–4 Clearing Error Messages

Before inputting a new program, any error messages recorded in memory should be cleared. It is assumed here that the causes of any of the errors for which error messages appear have already been taken care of. If the buzzer sounds when an attempt is made to clear an error message, eliminate the cause of the error, and then clear the error message (refer to *Section 5 Troubleshooting*).

To display any recorded error messages, press CLR, FUN, 6, 1, and then MON. The first message will appear. Pressing MON again will clear the present message and display the next error message. Continue pressing MON until all messages have been cleared. The ERROR indicator will go OFF when all messages have been cleared.

Although error messages can be accessed in any mode, they can be cleared only in PROGRAM mode.

Key Sequence



3–5–5 Setting and Reading from Program Memory Address

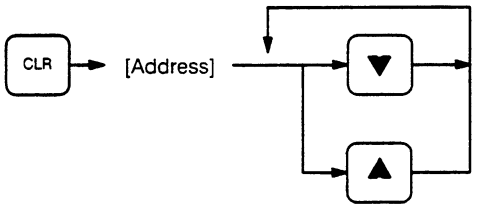
When inputting a program for the first time, it is generally written to Program Memory starting from address 000. Because this address appears when the display is cleared, it is not necessary to specify it.

When inputting a program starting from other than 000 or to read or modify a program that already exists in memory, the desired address must be designated. To designate an address, press CLR and then input the desired address.

Once the address is entered, press the up or down key and the desired contents will be displayed. The up and down keys can then be used to scroll through Program Memory. Each time one of these keys is pressed, the next or previous word in Program Memory will be displayed.

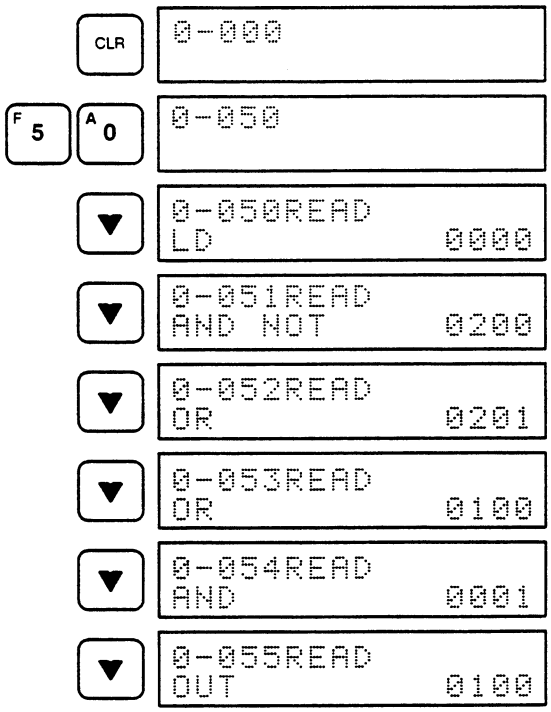
If Program Memory is read in RUN mode, the ON/OFF status of any displayed bit will also be shown.

Key Sequence



Example

If the following mnemonic code has already been input into Program Memory, the key inputs below would produce the displays shown.



Address	Instruction	Operands
050	LD	0000
051	AND NOT	0200
052	OR	0201
053	OR	0100
054	AND	0001
055	OUT	0100

3–5–6 Entering or Editing Programs

Programs can be entered or edited only in PROGRAM mode.

The same procedure is used to either input a program for the first time or to change a program that already exists. In either case, the current contents of Program Memory is overwritten.

To input a program, just follow the mnemonic code that was produced from the ladder diagram, ensuring that the proper address is set before starting. Once the proper address is displayed, input the first instruction word, and input any operands required, pressing ENT after each operand is typed into the Programming Console, i.e., ENT is pressed at the end of each line of the mnemonic code. When ENT is pressed, the designated instruction will be entered and the next display will appear. If the instruction requires two or more words, the next display will indicate the next operand required and provide a default value for it. If the instruction requires only one word, the next address will be displayed. Continue inputting each line of the mnemonic code until the entire program has been entered.

When inputting numeric values for operands, it is not necessary to input leading zeros. Leading zeros are required only when inputting function codes (see below). When designating operands, be sure to designate the data area for all DR and LR addresses by pressing the corresponding data area key, and to designate each constant by pressing CONT/#. CONT/# is not required for counter or timer SVs (see below).TC numbers as bit operands (i.e., completion flags) are designated by pressing either TIM or CNT before the address, depending on whether the TC number has been used to define a timer or a counter. To designate an indirect DR address, press CH/* before DR.

Inputting SV for Counters and Timers

The SV (set value) for a timer or counter is generally entered as a constant, although inputting the address of a word that holds the SV is also possible. When inputting an SV as a constant, CONT/# is not required; just input the numeric value and press ENT. To designate a word, press CLR and then input the word address as described above.

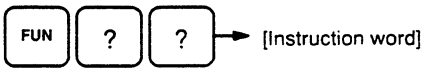
Designating Instructions

The most basic instructions are input using the Programming Console keys provided for them. All other instructions are entered using function codes. These function codes are always written after the instruction's mnemonic. If no function code is given, there should be a Programming Console key for that instruction.

To input an instruction using a function code, set the address, press FUN, input the function code, input any bit operands or definers required on the instruction line, and then press ENT.

Caution Enter function codes with care.

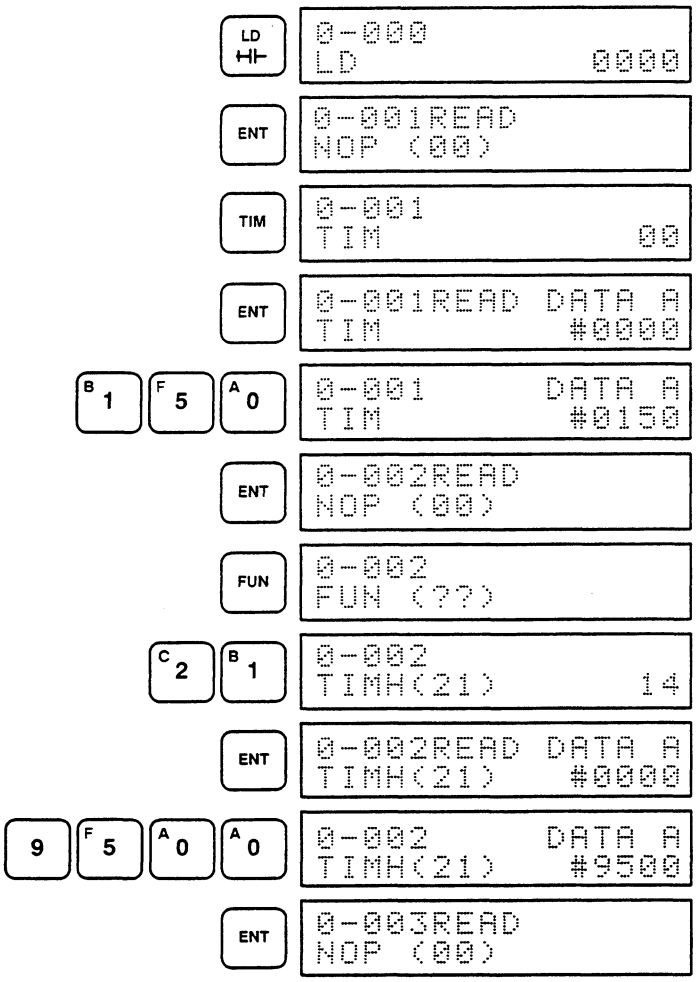
Key Sequence



Example

The following program can be entered using the key inputs shown below. Displays will appear as indicated.

Address	Instruction	Operands
000	LD	0000
001	TIM	00
		# 0150
002	TIMH(21)	01
		# 9500



Error Messages

The following error messages may appear when inputting a program. Correct the error as indicated and continue with the input operation.

Error Message	Error Type	Possible Cause/Correction
PRGM OVER	Program too large	Program size exceeds the capacity. (The last address is not a NOP instruction, so the program cannot be written.) Clear any data after the END instruction or shorten the program.
ADR OVER	Address too large	Program exceeds program memory's last address. Set the address again.
I/O No. ERR	Operand error	An illegal value has been entered for an operand. Reconfirm the allowable operand area for each instruction, and correct the data.

3–5–7 Checking the Program

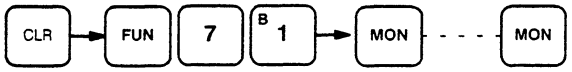
Once a program has been entered, it should be checked for syntax to be sure that no programming rules have been violated. This check should also be performed if the program has been changed in any way that might create a syntax error.

To check the program, input the key sequence shown below. When MON is entered, the program check will start. If an error is discovered, the check will stop and a display indicating the error and the error's address will appear.

Press MON to continue the check. If an error is not found, the program will be checked through to the first END(01). When the check has reached the first END, "PRGM CHK END(01)" will be displayed. If an error occurs, read the address which contains the error, and correct the program. Be sure to check corrected code by re-running the check function. CLR can be pressed to cancel the check after it has been started.

Note A syntax check can be performed on a program only in PROGRAM mode.

Key Sequence



Error message	Name	Meaning
PRGM CHK END (01)	Program check end	The check has been completed to the END instruction with no (more) errors having been found in the program.
(*program address*) or ????	-	An error has been detected in the program at the displayed address. Correct the code. If the LR area setting has been changed, "????" will be displayed. Change the LR area setting ensuring the region is identical to that specified when the program was created.
NO END INST	No END instruction	An END instruction cannot be found in the program. Input the END instruction at the end of the program.

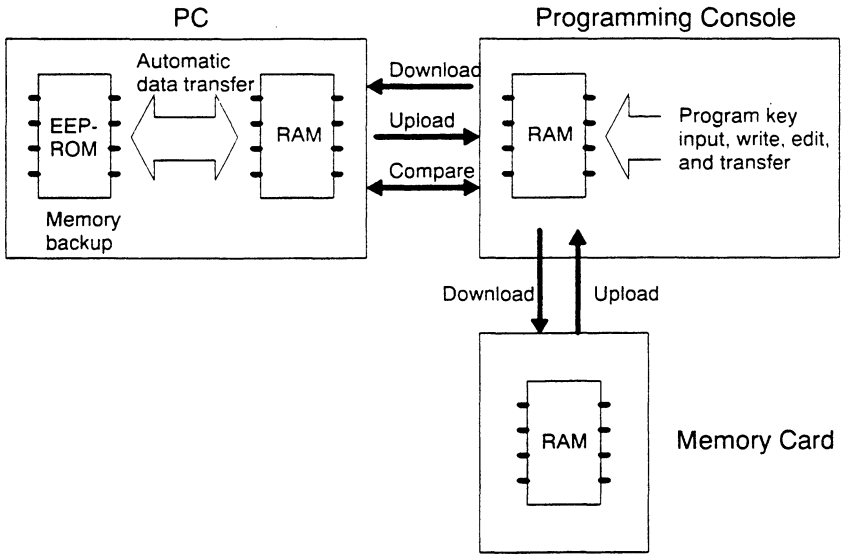
3–5–8 Program Transfer

After all errors are removed from the program, the program may be transferred from the Programming Console to the PC.

To be executed, the program has to be transferred to the PC from RAM in the Programming Console. Whenever a program is written to RAM in the PC, the program is automatically transferred to the EEPROM in the PC.

The program and/or data may also be stored in Memory Cards via the Programming Console. This provides a backup facility for programs and the later use of them to form the outline of new programs.

Data transfers are always referred to from the point of view of the Programming Console, i.e., downloading is always away from the Programming Console; uploading is always toward the Programming Console.

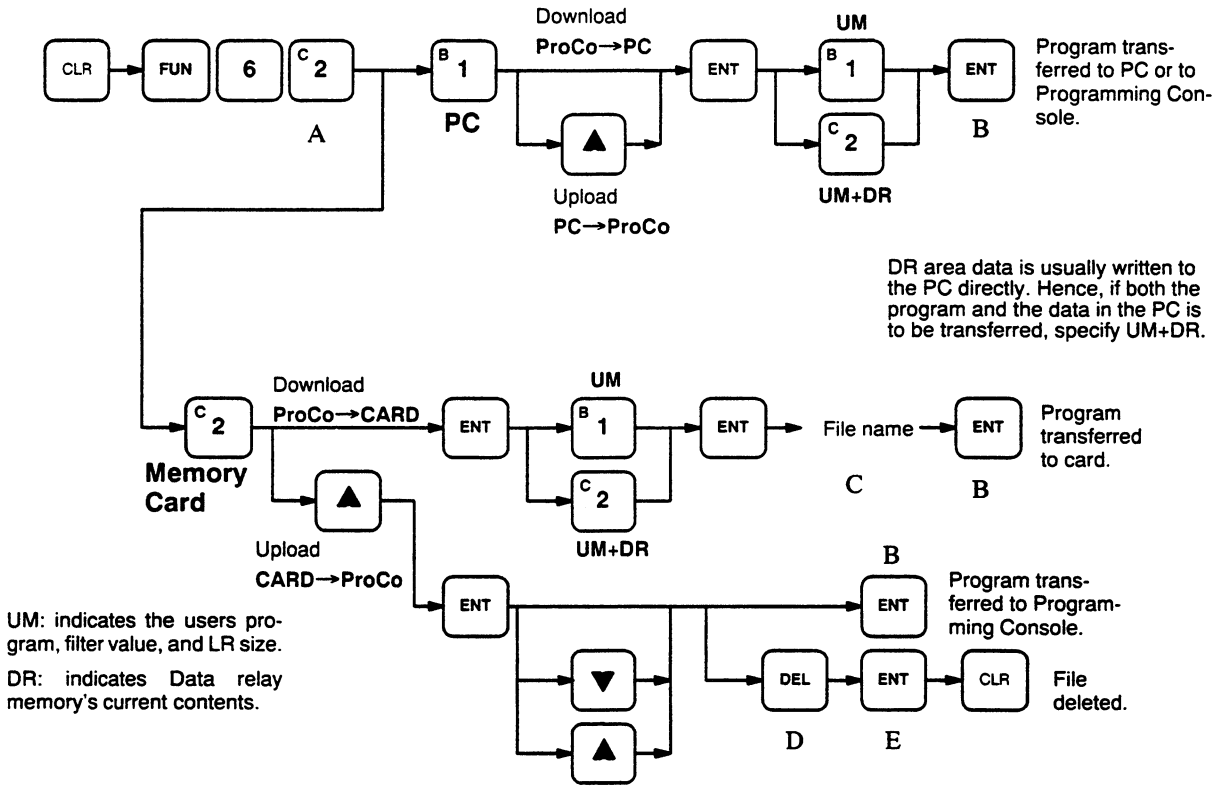


Note
1. New Memory Cards must be initialized before data can be stored. Be sure to format Memory Cards before use.
2. Placing a 27th program on the memory card will inhibit rewriting operations; do not store more than 26 programs per card.
3. If the size of the LR area is changed after programming operations have been started or the program code accesses illegal addresses, program transfer cannot be performed and the message "????" will be displayed on the Programming Console.

Transfer Procedure

The key sequence for transferring data between the PC and the Programming Console or the Programming Console and the Memory Card is given below. By selecting 1 or 2, after entering the function number, CPU or Memory Card transfer is selected.The up arrow or down arrow keys can be used to toggle between uploading and downloading.

Key Sequence



The Programming Consoles displays at different stages of the keying sequence are shown below.

A
```
0 PRGM TRANSFER?
  1.PC  2.CARD
```
When 1 or 2 is selected as the response to this prompt, the display will indicate the selection by placing a flashing cursor over the corresponding number.

B
```
0 PRGM TRANSFER
   ProCo-PC
```
The display will indicate the direction of the transfer by use of an arrow. During transfer, a cursor will flash over the arrow.

B
```
0 PRGM TRANSFER
END ProCo→PC
```
This display indicates the transfer is complete, in this example from the Programming Console to the PC.

C
```
0 PRGM TRANSFER?
 NAME =
```
When writing to a Memory Card, a file name must be assigned to the program to allow identification. The file name can consist of up to a maximum of 8 characters, the allowable characters 0 through 9 and A through F.

D
```
0 PRGM DELETE?
U--      1000
```
Indicates that a file containing only the user program called "1000" will be deleted.

E
```
0 PRGM   DELETE
END      1000
```
Indicates that the file called "1000" has been deleted.

Caution Files that have been deleted cannot be recovered. Be sure that you have designated the correct file before pressing the ENT Key.

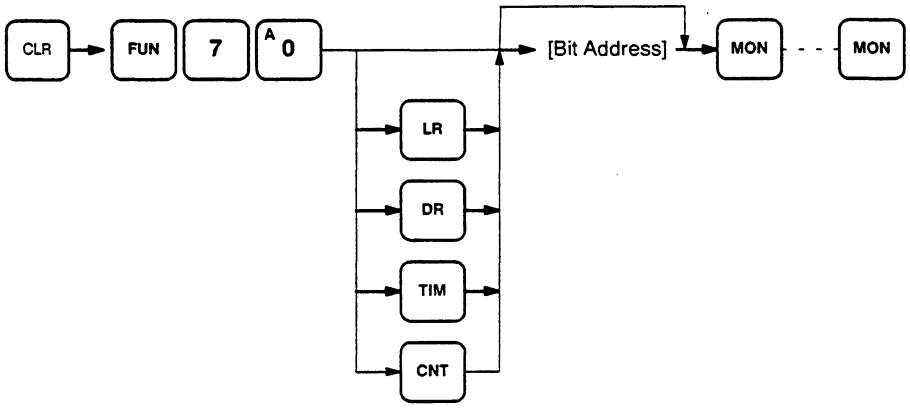
3–5–9 Program Searches

The program can be searched for occurrences of any data area address or timer/counter used in an instruction. Searches can be performed from any currently displayed address or from a cleared display.

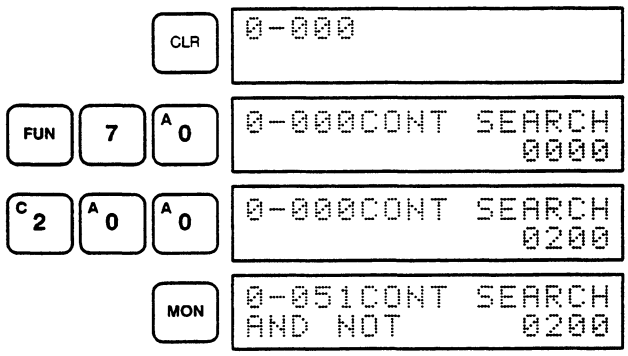
Once an occurrence of an instruction or bit address has been found, any additional occurrences of the same instruction or bit can be found by pressing MON again.

When the first word of a multiword instruction is displayed for a search operation, the other words of the instruction can be displayed by pressing the down key before continuing the search.

Key Sequence



Example:



3–5–10 Inserting and Deleting Instructions

In PROGRAM mode, any instruction that is currently displayed can be deleted or another instruction can be inserted before it. These operations are not possible in RUN mode.

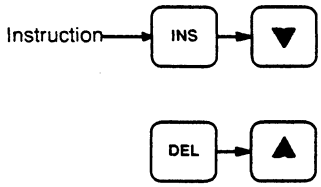
To insert an instruction, display the instruction before which you want the new instruction to be placed, input the instruction word in the same way as when inputting a program initially, and then press INS and the down key. If other words are required for the instruction, input these in the same way as when inputting the program initially.

To delete an instruction, display the instruction word of the instruction to be deleted and then press DEL and the up key. All the words for the designated instruction will be deleted.

Caution Be careful not to inadvertently delete instructions; there is no way to recover them without reinputting them completely.

Key Sequences

Locate position in program, then enter:



When an instruction is inserted or deleted, all addresses in Program Memory following the operation are adjusted automatically so that there are no blank addresses and no unaddressed instructions.

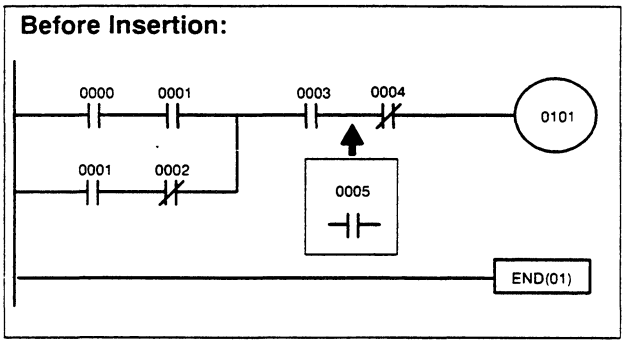
Example

The following mnemonic code shows the changes that are achieved in a program through the key sequences and displays shown below.

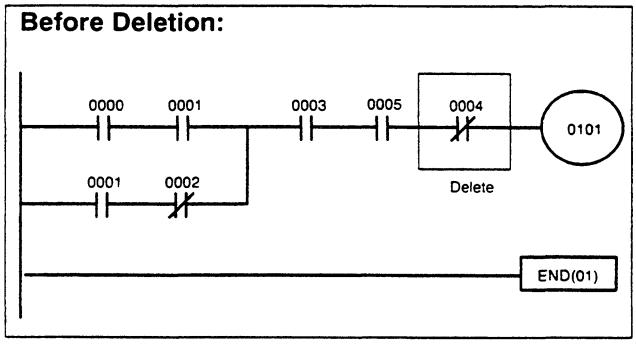
Original Program

Address	Instruction	Operands
000	LD	0000
001	AND	0001
002	LD	0201
003	AND NOT	0002
004	OR LD	-
005	AND	0003
006	AND NOT	0004
007	OUT	0101
008	END(01)	-

Before Insertion:

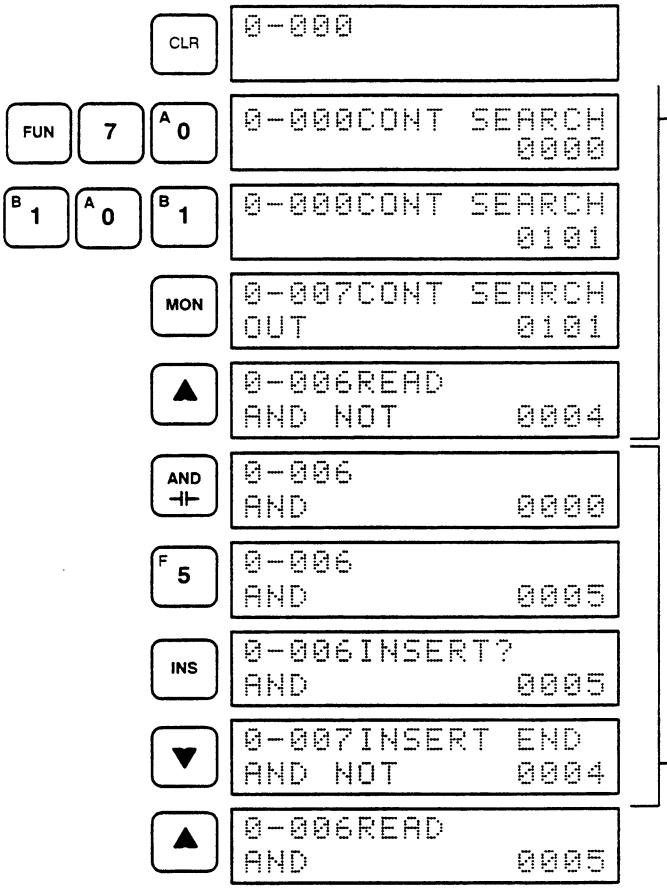


Before Deletion:



The following key inputs and displays show the procedure for achieving the program changes shown above.

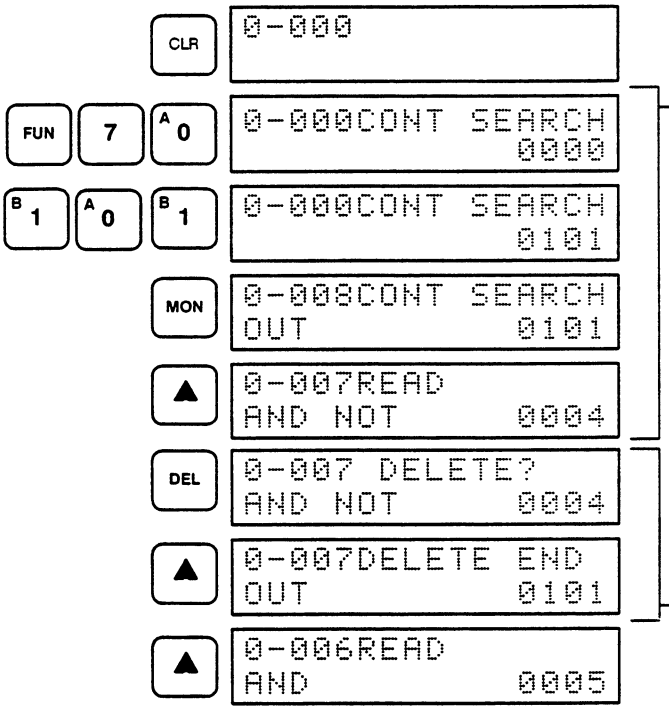
Inserting an Instruction



Program After Insertion

Address	Instruction	Operands
000	LD	0000
001	AND	0001
002	LD	0001
003	AND NOT	0002
004	OR LD	-
005	AND	0003
006	AND	0005
007	AND NOT	0004
008	OUT	0101
009	END(01)	-

Deleting an Instruction



Program After Deletion

Address	Instruction	Operands
000	LD	0000
001	AND NOT	0001
002	LD	0001
003	AND NOT	0002
004	OR LD	-
005	AND	0003
006	AND	0005
007	OUT	0101
008	END(01)	-

3–6 Advanced Programming

3–6–1 Interlocks

When an instruction line branches into two or more lines, it is sometimes necessary to use interlocks to maintain the execution condition that existed at a branching point. This is because instruction lines are executed across to a right-hand instruction before returning to the branching point to execute instructions one a branch line. If a condition exists on any of the instruction lines after the branching point, the execution condition could change during this time making proper execution impossible. The following diagrams illustrate this. In both diagrams, instruction 1 is executed before returning to the branching point and moving on to the branch line leading to instruction 2.

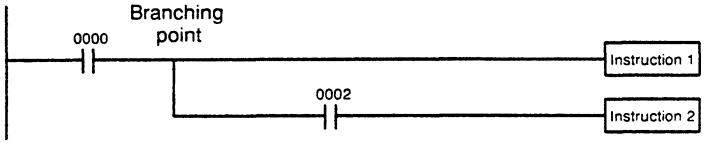


Diagram A: Correct Operation

Address	Instruction	Operands
000	LD	0000
001	Instruction 1	
002	AND	0002
003	Instruction 2	

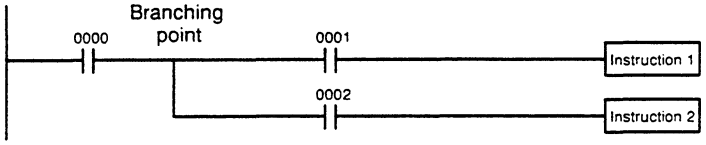


Diagram B: Incorrect Operation

Address	Instruction	Operands
000	LD	0000
001	AND	0001
002	Instruction 1	
003	AND	0002
004	Instruction 2	

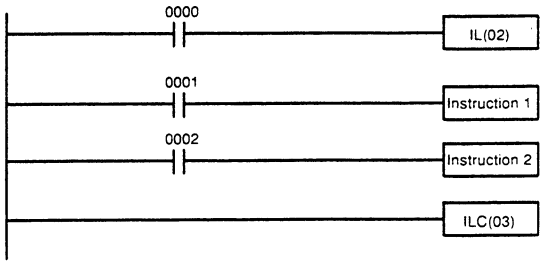
If, as shown in diagram A, the execution condition that existed at the branching point cannot be changed before returning to the branch line (instructions at the far right do not change the execution condition), then the branch line will be executed correctly and no special programming measure is required.

If, as shown in diagram B, a condition exists between the branching point and the last instruction on the top instruction line, the execution condition at the branching point and the execution condition after completing the top instruction line will sometimes be different, making it impossible to ensure correct execution of the branch line.

The problem of storing execution conditions at branching points can be handled by using the INTERLOCK (IL(02)) and INTERLOCK CLEAR (ILC(03)) instructions to eliminate the branching point completely while allowing a specific execution condition to control a group of instructions. The INTERLOCK and INTERLOCK CLEAR instructions are always used together.

When an INTERLOCK instruction is placed before a section of a ladder program, the execution condition for the INTERLOCK instruction will control the execution of all instruction up to the next INTERLOCK CLEAR instruction. If the execution condition for the INTERLOCK instruction is OFF, timers will be reset; counters, shift registers, and the KEEP instruction will be frozen (i.e., their operands and present values will not change); and all other instructions will be ignored through the next INTERLOCK CLEAR instruction.

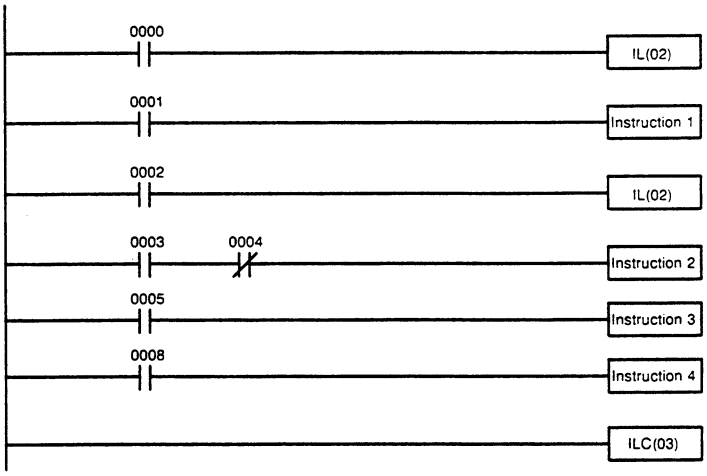
To create an interlocked program section, the conditions leading up to the branching point (i.e., the ones that are to control the interlocked section) are placed on an instruction line for the INTERLOCK instruction, all of lines leading from the branching point are written as separate instruction lines, and another instruction line is added for the INTERLOCK CLEAR instruction. No conditions are allowed on the instruction line for INTERLOCK CLEAR. Neither INTERLOCK nor INTERLOCK CLEAR requires an operand.



Address	Instruction	Operands
000	LD	0000
001	IL(02)	---
002	LD	0001
003	Instruction 1	
004	LD	0002
005	Instruction 2	
006	ILC(03)	---

If bit 0000 is ON in the revised version of diagram B, above, the status of bit 0001 and that of bit 0002 would determine the execution conditions for instructions 1 and 2, respectively. Because bit 0000 is ON (otherwise the interlocked section would not be executed), this would produce the same results as ANDing the status of each of these bits. If bit 0000 is OFF, the INTERLOCK instruction would produce an OFF execution condition for instructions 1 and 2 and then execution would continue with the instruction line following the INTERLOCK CLEAR instruction.

As shown in the following diagram, more than one INTERLOCK instruction can be used within one instruction block; each is effective through the next INTERLOCK CLEAR instruction (i.e., you can have two or more INTERLOCK instructions without an INTERLOCK CLEAR instruction between them, but two or more INTERLOCK CLEAR instructions without an INTERLOCK instruction between them is meaningless).



Address	Instruction	Operands
000	LD	0000
001	IL(02)	---
002	LD	0001
003	Instruction 1	
004	LD	0002
005	IL(02)	---
006	LD	0003
007	AND NOT	0004
008	Instruction 2	
009	LD	0005
010	Instruction 3	
011	LD	0008
012	Instruction 4	
013	ILC(03)	---

If bit 0000 in the above diagram is OFF (i.e., if the execution condition for the first INTERLOCK instruction is OFF), the section of the program from instruction 1 through 4 would be interlocked and execution would move to the instruction following the INTERLOCK CLEAR instruction. If bit 0000 is ON, the status of bit 0001 would be loaded as the execution condition for instruction 1 and then the status of bit 0002 would be loaded to form the execution condition for the second INTERLOCK instruction. If bit 0002 is OFF, the section from instruction 2 through 4 would be interlocked. If bit 0002 is ON, bit 0003, bit 0005, and bit 0008 would determine the first execution condition for the next instruction lines and execution would continue normally.

Note STEP(04) and SNXT(05) cannot be used between the INTERLOCK and INTERLOCK CLEAR instructions.

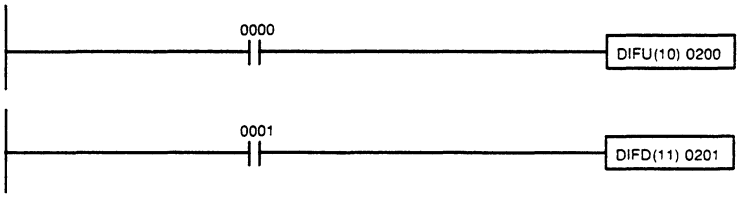
3–6–2 Controlling Bit Status

There are five instructions that can be used generally to control individual bit status. These are the OUTPUT, OUTPUT NOT, DIFFERENTIATE UP, DIF-

FERENTIATE DOWN, and KEEP instructions. All of these instructions appear as the last instruction in an instruction line and take a bit address for an operand.These instructions (except for OUTPUT and OUTPUT NOT, which have already been introduced) are introduced here because of their importance in most programs. Although these instructions are used to turn ON and OFF output bits (i.e., to send or stop output signals to external devices), they are also used to control the status of work bits and other bits in memory.

3–6–3 DIFFERENTIATE UP and DIFFERENTIATE DOWN

DIFFERENTIATE UP and DIFFERENTIATE DOWN instructions are used to turn the operand bit ON for one scan at a time. The DIFFERENTIATE UP instruction turns ON the operand bit for one scan after the execution condition for it goes from OFF to ON; the DIFFERENTIATE DOWN instruction turns ON the operand bit for one scan after the execution condition for it goes from ON to OFF. Both of these instructions require only one line of mnemonic code.



Address	Instruction	Operands
000	LD	0000
001	DIFU(10)	0200

Address	Instruction	Operands
000	LD	0001
001	DIFD(11)	0201

Here, bit 0200 will be turned ON for one scan after bit 0000 goes ON. The next time DIFU(10) 0200 is executed, bit 0200 will be turned OFF, regardless of the status of bit 0000. With the DIFFERENTIATE DOWN instruction, bit 0201 will be turned ON for one scan after bit 0001 goes OFF (bit 0201 will be kept OFF until then), and will be turned OFF the next time DIFD(11) 0201 is executed.

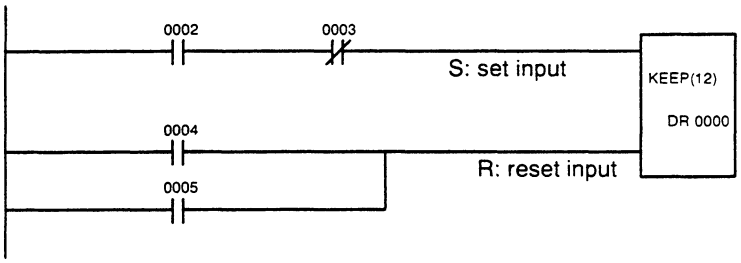
Up to a total of 16 DIFFERENTIATE UP and DIFFERENTIATE DOWN instruction can be used in a program.

3–6–4 KEEP

The KEEP instruction is used to maintain the status of the operand bit based on two execution conditions. To do this, the KEEP instruction is connected to two instruction lines. When the execution condition at the end of the first instruction line is ON, the operand bit of the KEEP instruction is turned ON. When the execution condition at the end of the second instruction line is ON, the operand bit of the KEEP instruction is turned OFF. (If both execution conditions are ON, the operand bit is also turned OFF.) The operand bit for the KEEP instruction will maintain its ON or OFF status even if it is located in an interlocked section of the diagram.

In the following example, DR 0000 will be turned ON when bit 0002 is ON and bit 0003 is OFF. DR 0000 will then remain ON until either bit 0004 or bit 0005 turns ON. With KEEP, as with all instructions requiring more than one

instruction line, the instruction lines are coded first before the instruction that they control.



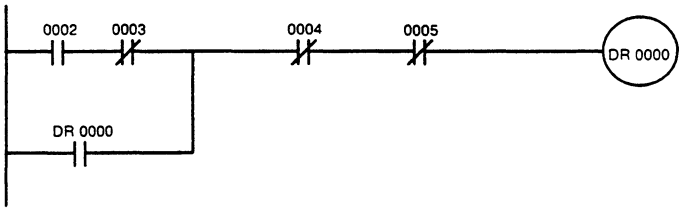
Address	Instruction	Operands
000	LD	0002
001	AND NOT	0003
002	LD	0004
003	OR	0005
004	KEEP(12)	DR 0000

3–6–5 Self-maintaining Bits (Seal)

Although the KEEP instruction can be used to create self-maintaining bits, it is sometimes necessary to create self-maintaining bits in another way so that they can be turned OFF when in an interlocked section of a program.

To create a self-maintaining bit, the operand bit of an OUTPUT instruction is used as a condition for the same OUTPUT instruction in an OR setup so that the operand bit of the OUTPUT instruction will remain ON or OFF until changes occur in other bits. At least one other condition is used just before the OUTPUT instruction to function as a reset. Without this reset, there would be no way to control the operand bit of the OUTPUT instruction.

The above diagram for the KEEP instruction can be rewritten as shown below. The only difference in these diagrams would be their operation in an interlocked program section when the execution condition for the INTERLOCK instruction was ON. Here, just as in the same diagram using the KEEP instruction, two reset bits are used, i.e., DR 0000 can be turned OFF by turning ON either bit 0004 or bit 0005.



Address	Instruction	Operands
000	LD	0002
001	AND NOT	0003
002	OR	DR 0000
003	AND NOT	0004
004	AND NOT	0005
005	OUT	DR 0000

3–6–6 Work Bits (Internal Relays)

In programming, combining conditions to directly produce execution conditions is often extremely difficult. These difficulties are easily overcome, however, by using certain bits to trigger other instructions indirectly. Such programming is achieved by using work bits. Sometimes entire words are required for these purposes. These words are referred to as work words.

Work bits are not transferred to or from the PC. They are bits selected by the programmer to facilitate programming as described above. I/O bits and other dedicated bits cannot be used as works bits. All bits in the bit area that are not allocated as I/O bits, and certain unused bits in the DR area, are available for use as work bits. Be careful to keep an accurate record of how and where you use work bits. This helps in program planning and writing, and also aids in debugging operations.

Work Bit Applications

Examples given later in this subsection show two of the most common ways to employ work bits. These should act as a guide to the almost limitless num-

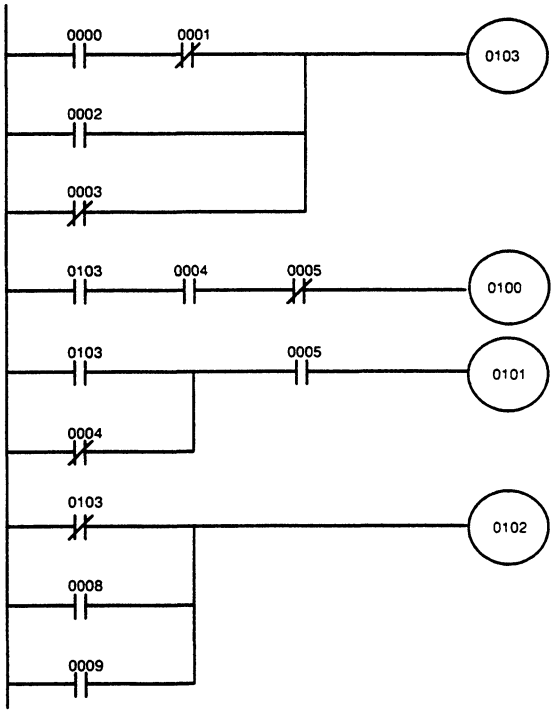
ber of ways in which the work bits can be used. Whenever difficulties arise in programming a control action, consideration should be given to work bits and how they might be used to simplify programming.

Work bits are often used with the OUTPUT, OUTPUT NOT, DIFFERENTIATE UP, DIFFERENTIATE DOWN, and KEEP instructions. The work bit is used first as the operand for one of these instructions so that later it can be used as a condition that will determine how other instructions will be executed. Work bits can also be used with other instructions, e.g., with the SHIFT REGISTER instruction (SFT(33)). An example of the use of work words and bits with the SHIFT REGISTER instruction is provided *3–7–20 SHIFT REGISTER - SFT(33)*.

Although they are not always specifically referred to as work bits, many of the bits used in the examples later in this section use work bits. Understanding the use of these bits is essential to effective programming.

Reducing Complex Conditions

Work bits can be used to simplify programming when a certain combination of conditions is repeatedly used in combination with other conditions. In the following example, bit 0000, bit 0001, bit 0002, and bit 0003 are combined in a logic block that stores the resulting execution condition as the status of bit 0103. Bit 0103 is then combined with various other conditions to determine output conditions for bit 0000, bit 0001, and bit 0002, i.e., to turn the outputs allocated to these bits ON or OFF.



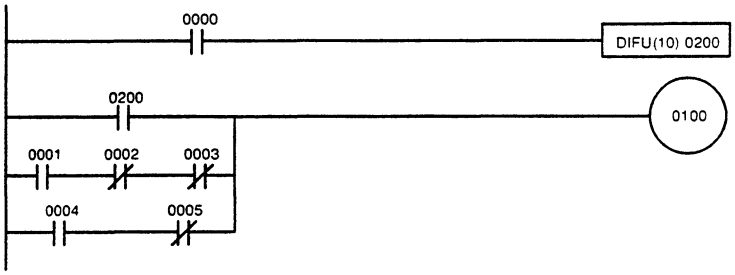
Address	Instruction	Operands
000	LD	0000
001	AND NOT	0001
002	OR	0002
003	OR NOT	0003
004	OUT	0103
005	LD	0103
006	AND	0004
007	AND NOT	0005
008	OUT	0100
009	LD	0103
010	OR NOT	0004
011	AND	0005
012	OUT	0101
013	LD NOT	0103
014	OR	0008
015	OR	0009
016	OUT	0102

Differentiated Conditions

Work bits can also be used if differential treatment is necessary for some, but not all, of the conditions required for execution of an instruction. In this example, bit 0100 must be left ON continuously as long as bit 0001 is ON and both bit 0002 and bit 0003 are OFF, or as long as bit 0004 is ON and bit 0005 is OFF. It must be turned ON for only one scan each time bit 0000 turns ON (unless one of the preceding conditions is keeping it ON continuously).

This action is easily programmed by using bit 0200 as a work bit as the operand of the DIFFERENTIATE UP instruction (DIFU(10)). When bit 0000 turns ON, bit 0100 will be turned ON for one scan and then be turned OFF the next

scan by DIFU(10). Assuming the other conditions controlling bit 0100 are not keeping it ON, the work bit 0200 will turn bit 0100 ON for one scan only.



Address	Instruction	Operands
000	LD	0000
001	DIFU(10)	0200
002	LD	0200
003	LD	0001
004	AND NOT	0002
005	AND NOT	0003
006	OR LD	---
007	LD	0004
008	AND NOT	0005
009	OR LD	---
010	OUT	0100

3–6–7 Programming Precautions

The number of conditions that can be used in series or parallel is unlimited as long as the memory capacity of the PC is not exceeded. Therefore, use as many conditions as required to draw a clear diagram. Although very complicated diagrams can be drawn with instruction lines, there must not be any conditions on lines running vertically between two other instruction lines. Diagram A shown below, for example, is not possible, and should be drawn as diagram B. Mnemonic code is provided for diagram B only; coding diagram A would be impossible.

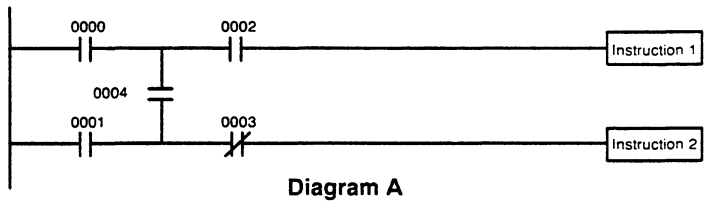


Diagram A

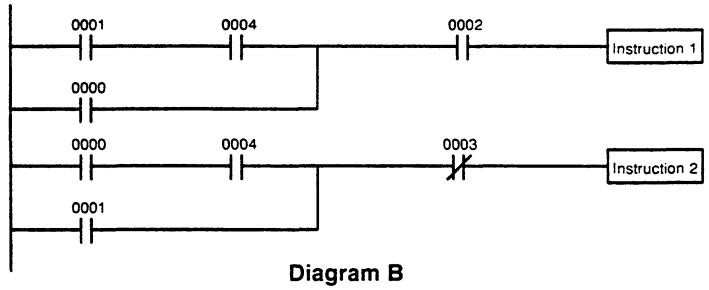
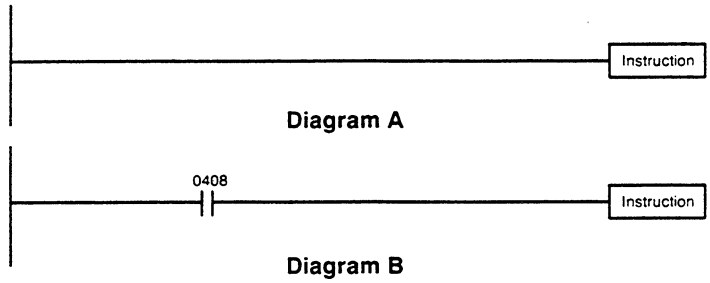


Diagram B

Address	Instruction	Operands
000	LD	0001
001	AND	0004
002	OR	0000
003	AND	0002
004	Instruction 1	
005	LD	0000
006	AND	0004
007	OR	0001
008	AND NOT	0003
009	Instruction 2	

The number of times any particular bit can be assigned to conditions is not limited, so use them as many times as required to simplify your program. Often, complicated programs are the result of attempts to reduce the number of times a bit is used.

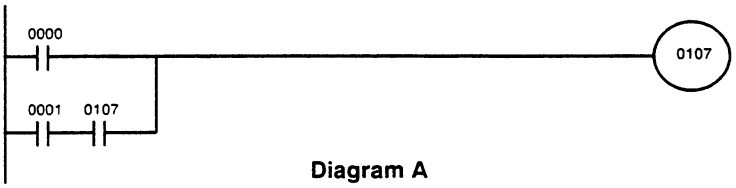
Except for instructions for which conditions are not allowed (e.g., INTERLOCK CLEAR, see below), every instruction line must also have at least one condition on it to determine the execution condition for the instruction at the right. Again, diagram A , below, must be drawn as diagram B. If an instruction must be continuously executed (e.g., if an output must always be kept ON while the program is being executed), the Always ON Flag (bit 0408) can be used.



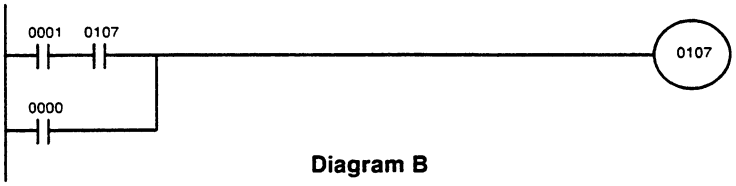
Address	Instruction	Operands
000	LD	0408
001	Instruction	

There are a few exceptions to this rule, including the INTERLOCK CLEAR and step instructions. Each of these instructions is used as the second of a pair of instructions and is controlled by the execution condition of the first of the pair. Conditions should not be placed on the instruction lines leading to these instructions.

When drawing ladder diagrams, it is important to keep in mind the number of instructions that will be required to input it. In diagram A, below, an OR LOAD instruction will be required to combine the top and bottom instruction lines. This can be avoided by redrawing as shown in diagram B so that no AND LOAD or OR LOAD instructions are required. Refer to *3–7–6 AND LOAD and OR LOAD* for more details.



Address	Instruction	Operands
000	LD	0000
001	LD	0001
002	AND	0107
003	OR LD	---
004	OUT	0107



Address	Instruction	Operands
000	LD	0001
001	AND	0107
002	OR	0000
003	OUT	0107

3–7 Instruction Set

The remainder of this section explains SP-series PC instructions individually.

3–7–1 Notation

In the remainder of this manual, all instructions will be referred to by their mnemonics. For example, the OUTPUT instruction will be called OUT; the AND LOAD instruction, AND LD. If you're not sure of the instruction a mnemonic is used for, refer to *Appendix C Programming Instructions and Execution Times*.

If an instruction is assigned a function code, it will be given in parentheses after the mnemonic. These function codes, which are 2-digit decimal numbers, are used to input most instructions into the CPU and are described briefly below. A table of instructions listed in order of function codes, is also provided in *Appendix C*.

3–7–2 Instruction Format

Most instructions have at least one or more operands associated with them. Operands indicate or provide the data on which an instruction is to be per-

formed. These are sometimes input as the actual numeric values (i.e., as constants), but are usually the addresses of data area words or bits that contain the data to be used. A bit whose address is designated as an operand is called an operand bit; a word whose address is designated as an operand is called an operand word. In some instructions, the word address designated in an instruction indicates the first of multiple words containing the desired data.

Each instruction requires one or more words in Program Memory. The first word is the instruction word, which specifies the instruction and contains any definers (described below) or operand bits required by the instruction. Other operands required by the instruction are contained in following words, one operand per word. Some instructions require up to five words.

A definer is an operand associated with an instruction and contained in the same word as the instruction itself. These operands define the instruction rather than telling what data it is to use. Examples of definers are TC numbers, which are used in timer and counter instructions to create timers and counters. Bit operands are also contained in the same word as the instruction itself, although these are not considered definers.

3–7–3 Data Areas, Definer Values, and Flags

In this section, each instruction description includes its ladder diagram symbol, the data areas that can be used by its operands, and the values that can be used as definers. Details for the data areas are also specified by the operand names and the type of data required for each operand (i.e., word or bit and, for words, hexadecimal or BCD).

Not all addresses in the specified data areas are necessarily allowed for an operand, e.g., if an operand requires two words, the last word in a data area cannot be designated as the first word of the operand because all words for a single operand must be within the same data area. Other specific limitations are given in a *Limitations* subsection. Refer to *3–2 Memory Areas* for addressing conventions and the addresses of flags and control bits.

The *Flags* subsection lists flags that are affected by execution of an instruction. These flags include the following.

Abbreviation	Name	Bit
ER	Instruction Execution Error Flag	0311
CY	Carry Flag	0312
LE	Less Than Flag	0313
EQ	Equals Flag	0314
GR	Greater Than Flag	0315

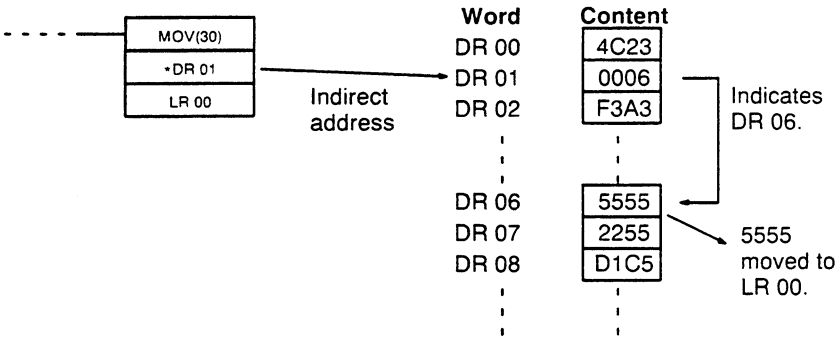
ER is the flag most commonly used for monitoring an instruction's execution. When ER goes ON, it indicates that an error has occurred in attempting to execute the current instruction. The *Flags* subsection of each instruction lists possible reasons for ER being ON. ER will turn ON if operands are not entered correctly. Instructions are not executed when ER is ON. A table of instructions and the flags they affect is provided in *Appendix E Error and Arithmetic Flag Operation*.

Indirect Addressing

When the DR area is specified for an operand, an indirect address can be used. Indirect DR addressing is specified by placing an asterisk before the DR: *DR.

When an indirect DR address is specified, the designated DR word will contain the address of the DR word that contains the data that will be used as the operand of the instruction. If, for example, *DR 01 was designated as the

first operand and LR 00 as the second operand of MOV(30), the contents of DR 01 was 0006, and DR 06 contained 5555, the value 5555 would be moved to LR 00.



When using indirect addressing, the address of the desired word must be in BCD and it must specify a word within the DR area. In the above example, the content of *DR 00 would have to be in BCD and between 0000 and 1515.

Designating Constants

Although data area addresses are most often given as operands, many operands and all definers are input as constants. The available value range for a given definer or operand depends on the particular instruction that uses it. Constants must also be entered in the form required by the instruction, i.e., in BCD or in hexadecimal.

3–7–4 Coding Right-hand Instructions

Writing mnemonic code for ladder instructions has already been described for ladder instructions. Converting the information in the ladder diagram symbol for all other instructions follows the same pattern, as described below, and is not specified for each instruction individually.

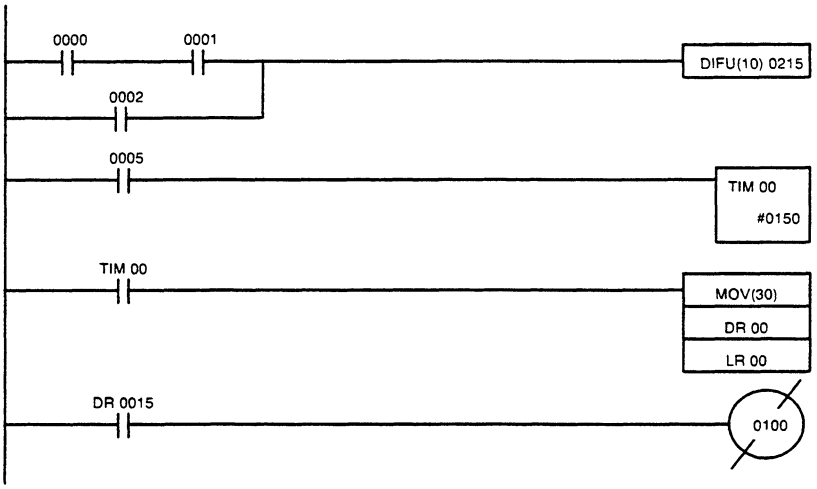
The first word of any instruction defines the instruction and provides any definers. If the instruction requires only a signal bit operand with no definer, the bit operand is also placed on the same line as the mnemonic. All other operands are placed on lines after the instruction line, one operand per line and in the same order as they appear in the ladder symbol for the instruction.

The address and instruction columns of the mnemonic code table are filled in for the instruction word only. For all other lines, the left two columns are left blank. If the instruction requires no definer or bit operand, the data column is left blank for first line. It is a good idea to cross through any blank data column spaces (for all instruction words that do not require data) so that the data column can be quickly scanned to see if any addresses have been left out.

If an I/O bit, work bit, or dedicated bit address is used in the data column, the left side of the column is left blank. If a DR, LR, or TC data address is used, the data area abbreviation is placed on the left side and the address is place on the right side. If a constant to be input, the number symbol (#) is placed on the left side of the data column and the number to be input is placed on the right side. Any numbers input as definers in the instruction word do not require the number symbol on the right side. TC bits, once defined as a timer or counter, take a TIM (timer) or CNT (counter) prefix.

When coding an instruction that has a function code, be sure to write in the function code, which will be necessary when inputting the instruction via the Programming Console.

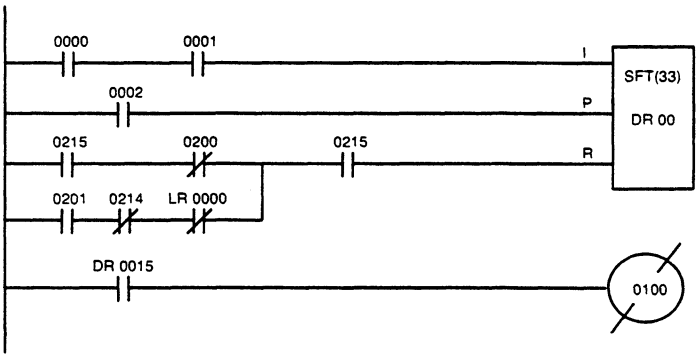
The following diagram and corresponding mnemonic code illustrates the points described above.



Address	Instruction	Data	
000	LD		0000
001	AND		0001
002	OR		0002
003	DIFU(10)		0215
004	LD		0005
005	TIM		00
		#	0150
006	LD	TIM	00
007	MOV(30)		--
		DR	00
		LR	00
008	LD	DR	0015
009	OUT NOT		0100

Multiple Instruction Lines

If a right-hand instruction requires multiple instruction lines (such as KEEP(12)), all of the lines for the instruction are entered before the right-hand instruction. Each of the lines for the instruction is coded, starting with LD or LD NOT, to form 'logic blocks' that are combined by the right-hand instruction. An example of this for SFT(33) is shown below.

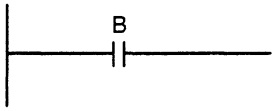
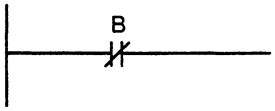
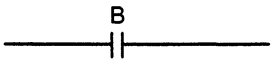
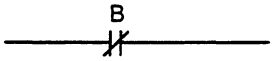
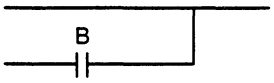
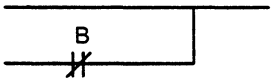


Address	Instruction	Data	
000	LD		0000
001	AND		0001
002	LD		0002
003	LD		0215
004	AND NOT		0200
005	LD		0201
006	AND NOT		0214
007	AND NOT	LR	0000
008	OR LD		--
009	AND		0215
010	SFT(33)		--
		DR	00
011	LD	DR	0015
012	OUT NOT		0100

END(01)

When you have finished coding the program, make sure you have placed END(01) at the last address. If there is not END(01) instruction in the program, the program will not be executed even if you switch to RUN mode.

3–7–5 LOAD, LOAD NOT, AND, AND NOT, OR, and OR NOT

	Ladder Symbols	Operand Data Areas
LOAD - LD		**B**: Bit I/O, work, dedicated, DR, LR, TC
LOAD NOT - LD NOT		**B**: Bit I/O, work, dedicated, DR, LR, TC
AND - AND		**B**: Bit I/O, work, dedicated, DR, LR, TC
AND NOT - AND NOT		**B**: Bit I/O, work, dedicated, DR, LR, TC
OR - OR		**B**: Bit I/O, work, dedicated, DR, LR, TC
OR NOT - OR NOT		**B**: Bit I/O, work, dedicated, DR, LR, TC

Limitations

There is no limit to the number of any of these instructions, or restrictions in the order in which they must be used, as long as the memory capacity of the PC is not exceeded.

Description

These six basic instructions correspond to the conditions on a ladder diagram. As described in *3–4 Basic Programming*, the status of the bits assigned to each instruction determines the execution conditions for all other instructions. Each of these instructions and each bit address can be used as many times as required. Each can be used in as many of these instructions as required.

The status of the bit operand (B) assigned to LD or LD NOT determines the first execution condition. AND takes the logical AND between the execution condition and the status of its bit operand; AND NOT, the logical AND between the execution condition and the inverse of the status of its bit operand. OR takes the logical OR between the execution condition and the status of its bit operand; OR NOT, the logical OR between the execution condition and the inverse of the status of its bit operand. The ladder symbol for loading TR bits is different from that shown above. Refer to *3–4–3 Ladder Instructions* for details.

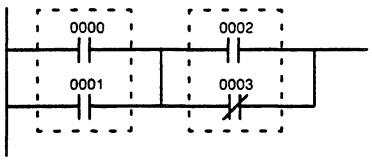
Flags

There are no flags affected by these instructions.

3–7–6 AND LOAD and OR LOAD

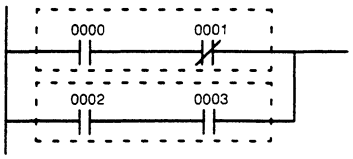
AND LOAD - AND LD

Ladder Symbol



OR LOAD - OR LD

Ladder Symbol



Description

When instructions are combined into blocks that cannot be logically combined using only OR and AND operations, AND LD and OR LD are used. Whereas AND and OR operations logically combine a bit status and an execution condition, AND LD and OR LD logically combine two execution conditions, the current one and the last unused one.

In order to draw ladder diagrams, it is not necessary to use AND LD and OR LD instructions. They are used to convert the program to and input it in mnemonic form.

In order to reduce the number of programming instructions required, a basic understanding of logic block instructions is required. For an introduction to logic blocks, refer to *3–4–6 Logic Block Instructions*.

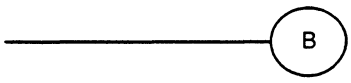
Flags

There are no flags affected by these instructions.

3–7–7 OUTPUT and OUTPUT NOT - OUT and OUT NOT

OUTPUT - OUT

Ladder Symbol

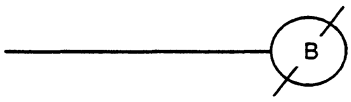


Operand Data Areas

B: Bit
Output bits, work bits, DR, LR

OUTPUT NOT - OUT NOT

Ladder Symbol



Operand Data Areas

B: Bit
Output bits, work bits, DR, LR

Limitations

Any output bit can generally be used in only one instruction that controls its status. Refer to *3–2–2 I/O Bits* for details.

Description

OUT and OUT NOT are used to control the status of the designated bit according to the execution condition.

OUT turns ON the designated bit for an ON execution condition, and turns OFF the designated bit for an OFF execution condition.

OUT NOT turns ON the designated bit for a OFF execution condition, and turns OFF the designated bit for an ON execution condition.

OUT and OUT NOT can be used to control execution by turning ON and OFF bits that are assigned to conditions on the ladder diagram, thus determining execution conditions for other instructions. This is particularly helpful and allows a complex set of conditions to be used to control the status of a single work bit, and then that work bit can be used to control other instructions.

The length of time that a bit is ON or OFF can be controlled by combining the OUT or OUT NOT with TIM. Refer to Examples under *3–7–14 TIMER - TIM* for details.

Flags There are no flags affected by these instructions.

3–7–8 DIFFERENTIATE UP and DIFFERENTIATE DOWN - DIFU(10) and DIFD(11)

Ladder Symbols	Operand Data Areas
DIFU(10) B	**B**: Bit Output bits, work bits, DR, LR
DIFD(11) B	**B**: Bit Output bits, work bits, DR, LR

Limitations The total of all DIFU(10) and DIFD(11) instruction in any one program must be 16 or less. Any output bit can generally be used in only one instruction that controls its status. Refer to *3–2–2 I/O Bits* for details.

Description DIFU(10) and DIFD(11) are used to turn the designated bit ON for one scan only.

Whenever executed, DIFU(10) compares its current execution with the previous execution condition. If the previous execution condition was OFF and the current one is ON, DIFU(10) will turn ON the designated bit. If the previous execution condition was ON and the current execution condition is either ON or OFF, DIFU(10) will either turn the designated bit OFF or leave it OFF (i.e., if the designated bit is already OFF). The designated bit will thus never be ON for longer than one scan, assuming it is executed each scan (see *Precautions*, below).

Whenever executed, DIFD(11) compares its current execution with the previous execution condition. If the previous execution condition is ON and the current one is OFF, DIFD(11) will turn ON the designated bit. If the previous execution condition was OFF and the current execution condition is either ON or OFF, DIFD(11) will either turn the designated bit OFF or leave it OFF. The designated bit will thus never be ON for longer than one scan, assuming it is executed each scan (see *Precautions*, below).

Flags There are no flags affected by these instructions.

Precautions DIFU(10) and DIFD(11) operation can be uncertain when the instructions are programmed between IL and ILC. Refer to *3–7–10 INTERLOCK and INTERLOCK CLEAR - IL(02) and ILC(03)* for details.

Example 1: One-time Execution of Other Instructions In diagram A, below, whenever CMP(32) is executed with an ON execution condition it will compare the contents of the two operand words (DR 10 and DR 00) and set the arithmetic flags (GR, EQ, and LE) accordingly. If the execution condition remains ON, flag status may be changed each scan if the content of one or both operands change. Diagram B, however, is an example of how DIFU(10) can be used to ensure that CMP(32) is executed only once each time the desired execution condition goes ON.

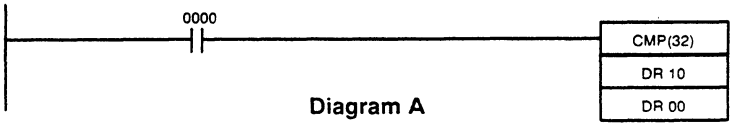


Diagram A

Address	Instruction	Operands
000	LD	0000
001	CMP(32)	
		DR 10
		DR 00

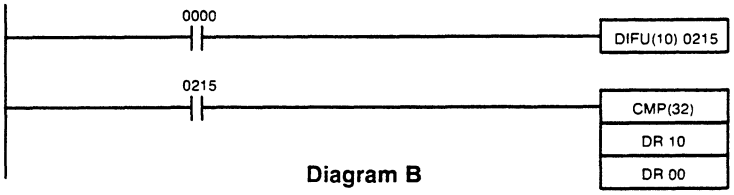
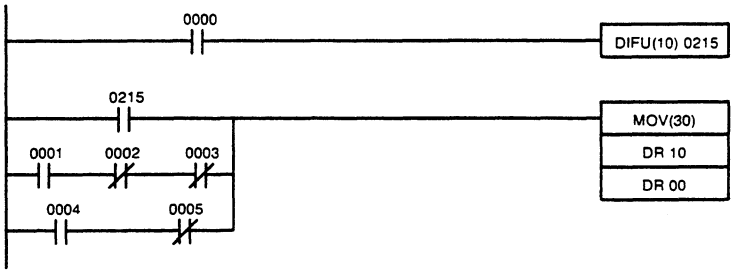


Diagram B

Address	Instruction	Operands
000	LD	0000
001	DIFU(10)	0215
002	LD	0215
003	CMP(32)	
		DR 10
		DR 00

Example 2: Use to Simplify Programming

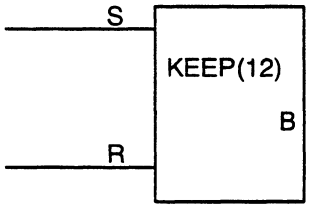
The following diagram would be very complicated to draw without using DIFU(10) because only one of the conditions determining the execution condition for MOV(30) requires differentiated treatment.



Address	Instruction	Operands
000	LD	0000
001	DIFU(10)	0215
002	LD	0215
003	LD	0001
004	AND NOT	0002
005	AND NOT	0003
006	OR LD	---
007	LD	0004
008	AND NOT	0005
009	OR LD	---
010	MOV(30)	
		DR 10
		DR 00

3–7–9 KEEP - KEEP(12)

Ladder Symbol



Operand Data Areas

B: Bit
Output bits, work bits, DR, LR

Limitations

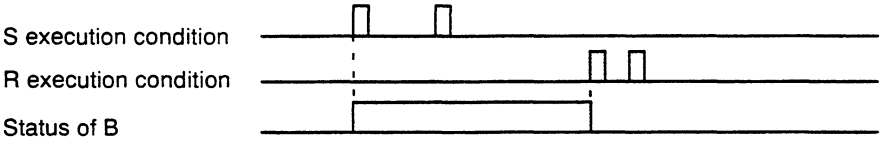
Any output bit can generally be used in only one instruction that controls its status. Refer to *3–2–2 I/O Bits* for details.

Description

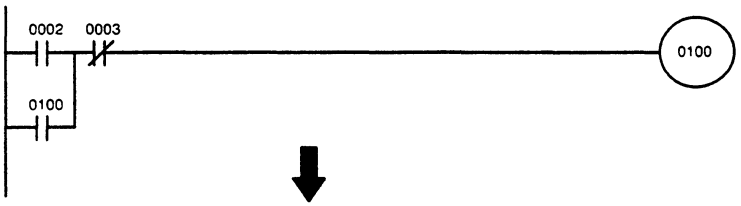
KEEP(12) is used to maintain the status of the designated bit based on two execution conditions. These execution conditions are labeled S and R. S is the set input; R, the reset input. KEEP(12) operates like a latching relay that is set by S and reset by R.

When S turns ON, the designated bit will go ON and stay ON until reset, regardless of whether S stays ON or goes OFF. When R turns ON, the designated bit will go OFF and stay OFF until reset, regardless of whether R stays

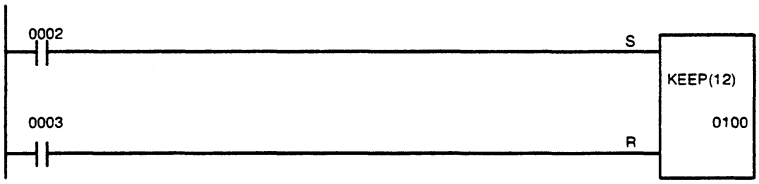
ON or goes OFF. The relationship between execution conditions and KEEP(12) bit status is shown below.



KEEP(12) operates like the self-maintaining bit described in *3–6–5 Self-maintaining Bits (Seal)*. The following two diagrams would function identically, though the one using KEEP(12) requires one less instruction to program and would maintain status even in an interlocked program section.



Address	Instruction	Operands
000	LD	0002
001	OR	0100
002	AND NOT	0003
003	OUT	0100



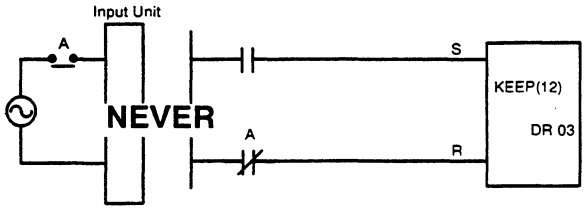
Address	Instruction	Operands
000	LD	0002
001	LD	0003
002	KEEP(12)	0100

Flags

There are no flags affected by this instruction.

Precautions

Never use an input bit in a normally closed condition on the reset (R) for KEEP(12) when the input device uses an AC power supply. The delay in shutting down the PC's DC power supply (relative to the AC power supply to the input device) can cause the designated bit of KEEP(12) to be reset. This situation is shown below.

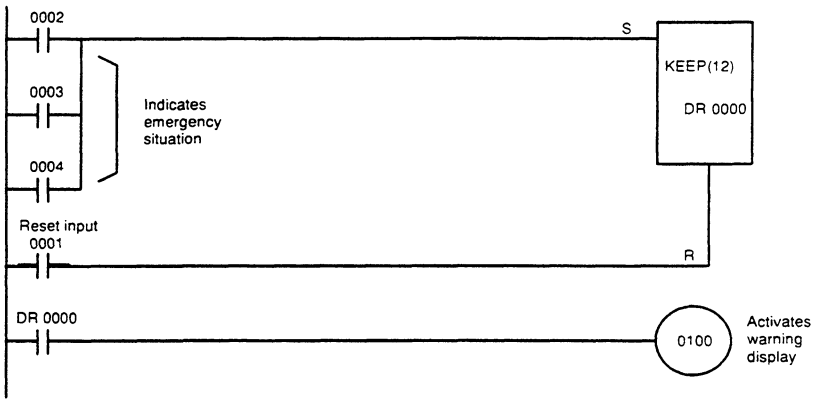


Bits used in KEEP are not reset in interlocks. Refer to the *3–7–10 INTERLOCK and INTERLOCK CLEAR - IL(02) and ILC(03)* for details.

Example

If a DR bit is used, bit status will be retained even during a power interruption. KEEP(12) can thus be used to program bits that will maintain status after restarting the PC following a power interruption. An example of this that can be used to produce a warning display following a system shutdown for an emergency situation is shown below. Bits 0002, 0003, and 0004 would be turned ON to indicate some type of error. Bit 0001 would be turned ON to reset the warning display. DR 0000, which is turned ON when any one of the

three bits indicates an emergency situation, is used to turn ON the warning indicator through 0100.

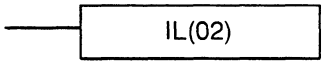


Address	Instruction	Operands
000	LD	0002
001	OR	0003
002	OR	0004
003	LD	0001
004	KEEP(12)	DR 0000
005	LD	DR 0000
006	OUT	0100

KEEP(12) can also be combined with TIM to produce delays in turning bits ON and OFF. Refer to *3–7–14 TIMER - TIM* for details.

3–7–10 INTERLOCK and INTERLOCK CLEAR - IL(02) and ILC(03)

Ladder Symbol



Ladder Symbol

Description

IL(02) is always used in conjunction with ILC(03) to create interlocks. Interlocks are used to create program sections that are either executed normally or partially reset and frozen, depending on the interlock condition (i.e., the execution condition of IL(02)). If the execution condition of IL(02) is ON, the program will be executed as written.

If the execution condition for IL(02) is OFF, the interlocked section between IL(02) and ILC(03) will be treated as shown in the following table:

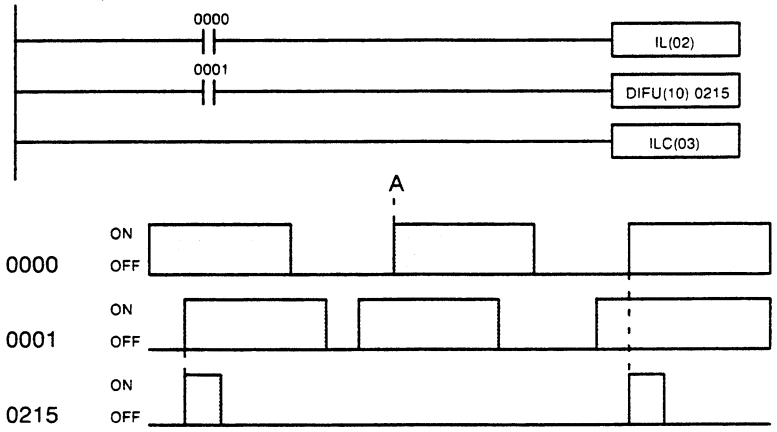
Instruction	Treatment
OUT and OUT NOT	Designated bit turned OFF.
TIM, TIMM(20), TIMH(21), ATIM(22), ATM1(25), and ATM2(26)	Reset.
CNT, RDM(23), and CNTH(24)	Frozen and PV maintained.
KEEP(12)	Bit status maintained.
DIFU(10) and DIFD(11)	Not executed (see below).
All others	Not executed.

IL(02) and ILC(03) do not necessarily have to be used in pairs. IL(02) can be used several times in a row, with each IL(02) creating an interlocked section through the next ILC(03). ILC(03) cannot be used unless there is at least one IL(02) between it and any previous ILC(03).

DIFU(10) and DIFD(11) in Interlocks

Changes in the execution condition for a DIFU(10) or DIFD(11) are not recorded if the DIFU(10) or DIFD(11) is in an interlocked section and the execution condition for the IL(02) is OFF. When DIFU(10) or DIFD(11) is execution in an interlocked section immediately after the execution condition for the IL(02) has gone ON, the execution condition for the DIFU(10) or DIFD(11) will be compared to the execution condition that existed before the interlock

became effective (i.e., before the interlock condition for IL(02) went OFF). The ladder diagram and bit status changes for this are shown below. The interlock is in effect while bit 0000 is OFF. Notice that bit 0215 is not turned ON at the point labeled A even though 0001 has turned OFF and then back ON.



Address	Instruction	Operands
000	LD	0000
001	IL(02)	
002	LD	0001
003	DIFU(10)	0215
004	ILC(03)	

Precautions

There must be an ILC(03) following any one or more IL(02).

Although as many IL(02) instructions as necessary can be used with one ILC(03), ILC(03) instructions cannot be used consecutively without at least one IL(02) in between. Whenever a ILC(03) is executed, all interlocks between the active ILC(03) and the preceding ILC(03) are cleared.

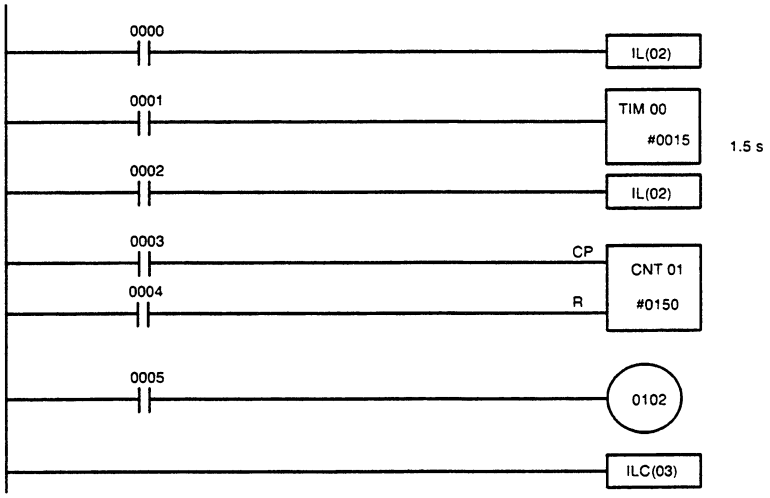
STEP(04) and SNXT(05) cannot be used between the INTERLOCK and INTERLOCK CLEAR instructions.

Flags

There are no flags affected by these instructions.

Example

The following diagram shows IL(02) being used twice with one ILC(03).

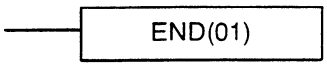


Address	Instruction	Operands
000	LD	0000
001	IL(02)	
002	LD	0001
003	TIM	00
		# 0015
004	LD	0002
005	IL(02)	
006	LD	0003
007	LD	0004
008	CNT	01
		# 0150
009	LD	0005
010	OUT	0102
011	ILC(03)	

When the execution condition for the first IL(02) is OFF, TIM 00 will be reset to 1.5 s, CNT 01 will not be changed, and 0102 will be turned OFF. When the execution condition for the first IL(02) is ON and the execution condition for the second IL(02) is OFF, TIM 00 will be executed according to the status of 0001, CNT 01 will not be changed, and 0102 will be turned OFF. When the execution conditions for both the IL(02) are ON, the program will execute as written.

3–7–11 END - END(01)

Ladder Symbol



Description

END(01) is required as the last instruction in any program. No instructions written after END(01) will be executed. END(01) can be placed anywhere in the program to execute all instructions up to that point, as is sometimes done to debug a program, but it must be removed to execute the remainder of the program.

If there is no END(01) in the program, no instructions will be executed and the error message "NO END INST" will appear.

Flags

END(01) turns OFF the ER, CY, GR, EQ, and LE flags.

3–7–12 NO OPERATION - NOP(00)

Description

NOP(00) is not generally required in programming and there is no ladder symbol for it. When NOP(00) is found in a program, nothing is executed and program execution moves to the next instruction. When memory is cleared prior to programming, NOP(00) is written at all addresses. NOP(00) can be input through the 00 function code.

Flags

There are no flags affected by NOP(00).

3–7–13 Timers and Counters

TIM and TIMM(20) are decrementing ON-delay timer instructions which require a TC number and a set value (SV). The TIM SV is input to the tenths of a second; the TIMM(20) SV is input to the hundredths of a second.

TIMH(21) is a decrementing ON-delay timer instruction which requires an SV, The SV is input to the thousandths of a second.

ATIM(22) is a decrementing ON-delay timer with a hardware adjustment for the SV. On the SP16 and SP20, the hardware adjustment is the same for both ATIM(22) and ATM1(25).

ATM1(25) and ATM2(26) are available with the SP16 and SP20 only. Like ATIM(22), they are decrementing ON-delay timers and the SV can be set by hardware adjustments on the front of the CPU. Unlike ATIM(22), the SV can also be set in a word.

CNT is a decrementing counter instruction and RDM(23) is a reversible drum counter instruction. Both require a TC number and a SV. Both are also connected to multiple instruction lines which serve as an input signal(s), a reset, and for RDM(23), an up/down input. RDM(23) also requires specification of the first word in the results table.

CNTH(24) is a high-speed incrementing counter that is available with the SP16 and SP20 only. It can count pulses as fast as 3.3 kHz.

Any one TC number cannot be defined twice, i.e., once it has been used as the definer in any of the timer or counter instructions, it cannot be used again. Once defined, TC numbers can be used as many times as required as operands in instructions.

TC numbers run from 00 through 15. No prefix is required when using a TC number as a definer in a timer or counter instruction. Once defined as a tim-

er, a TC number can be prefixed with TIM for use as an operand in certain instructions. The TIM prefix is used regardless of the timer instruction that was used to define the timer. Once defined as a counter, a TC number can be prefixed with CNT for use as an operand in certain instructions. The CNT is also used regardless of the counter instruction that was used to define the counter.

TC numbers can be designated as operands that require either bit or word data. When designated as an operand that requires bit data, the TC number accesses a bit that functions as a 'Completion Flag' that indicates when the time/count has expired, i.e., the bit, which is normally OFF, will turn ON when the designated SV has expired. When designated as an operand that requires word data, the TC number accesses a memory location that holds the present value (PV) of the timer or counter. The PV of a timer or counter can thus be used as an operand in CMP(32), or any other instruction for which the TC area is allowed. This is done by designating the TC number used to define that timer or counter to access the memory location that holds the PV.

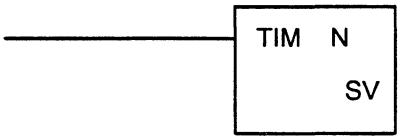
TC numbers TC 11 through TC 15 (just TC 14 and TC 15 for the SP10) are assigned to specific instructions, as shown in the table below.1.

TC number	Instruction	Applicable PCs
TC 11	ANALOG TIMER 1, ATM1(25)	SP16, SP20
TC 12	ANALOG TIMER 2, ATM2(26)	SP16, SP20
TC 13	HIGH-SPEED COUNTER, CNTH(24)	SP16, SP20
TC 14	HIGH-SPEED TIMER, TIMH(21)	SP10, SP16, SP20
TC 15	ANALOG TIMER, ATIM(22)	SP10, SP16, SP20

Note The present value of timers and counters can be monitored through the Programming Console. Refer to the Bit/TC Monitor and Multibit/TC Monitor operations.

3–7–14 TIMER - TIM

Ladder Symbol



Definer Values

N: TC number
(00 through 15)

Operand Data Areas

SV: Set value (BCD)
SP10: #
SP16, SP20: I/O, work, DR, LR, #

Limitations

SV is between 000.0 and 999.9 seconds. The decimal point is not entered.

Each TC number can be used as the definer in only one timer or counter instruction.

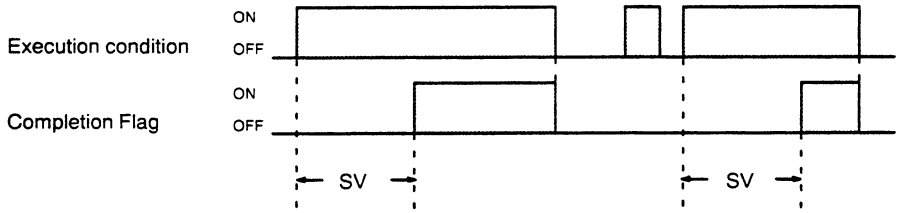
TC 11 through TC 15 should not be used in TIM if they are required for the specific instruction to which they are assigned. Refer to the table on page 82.

Description

A timer is activated when its execution condition goes ON and is reset (to SV) when the execution condition goes OFF. Once activated, TIM measures in units of 0.1 second from the SV. TIM accuracy is +0.0/–0.1 second.

If the execution condition remains ON long enough for TIM to time down to zero, the Completion Flag for the TC number used will turn ON and will remain ON until TIM is reset (i.e., until its execution condition is goes OFF).

The following figure illustrates the relationship between the execution condition for TIM and the Completion Flag assigned to it.



Precautions

Timers in interlocked program sections are reset when the execution condition for IL(02) is OFF. Power interruptions also reset timers. If a timer that is not reset under these conditions is desired, dedicated clock pulse bits can be counted to produce timers using CNT. Refer to *3–7–18 COUNTER - CNT* for details.

Flags

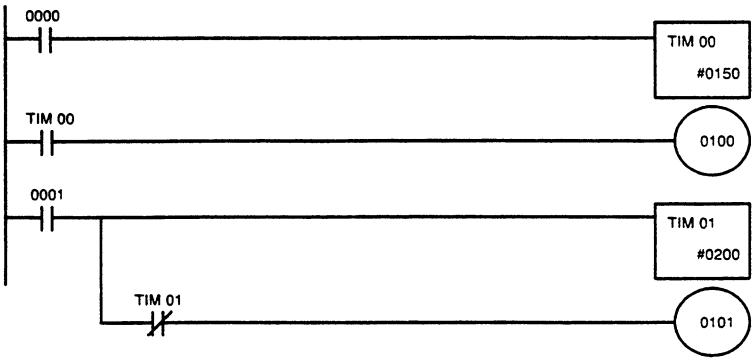
ER: The Error Flag (0311) will be turned ON when the SV is set in a word (SP16 and SP20 only) but the content of the indicated word is not BCD. The instruction will be executed, but operation will not be reliable.

Examples

All of the following examples use OUT to control output bits. There is no reason, however, why these diagrams cannot be modified to control execution of other instructions.

Example 1: Basic Application

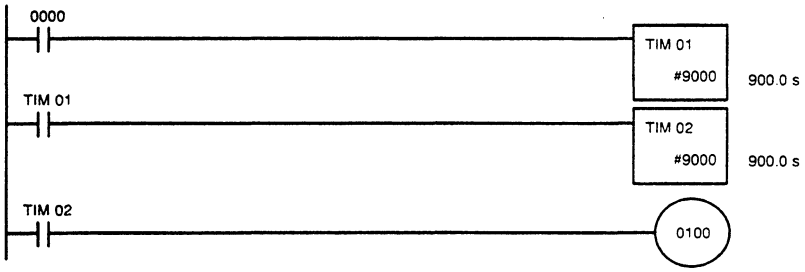
The following example shows two timers. Here, 0100 will be turned ON after bit 0000 goes ON and stays ON for at least 15 seconds. When bit 0000 goes OFF, the timer will be reset and 0100 will be turned OFF. When 0001 goes ON, TIM 01 is started. Bit 0101 is also turned ON when 0001 goes ON. When 20 seconds have expired, 0101 is turned OFF. This bit will also be turned OFF when TIM 01 is reset, regardless of whether or not SV has expired.



Address	Instruction	Operands
000	LD	0000
001	TIM	00
		# 0150
002	LD	TIM 00
003	OUT	0100
004	LD	0001
005	TIM	01
		# 0200
006	AND NOT	TIM 01
007	OUT	0101

Example 2: Extended Timers

There are two ways to achieve timers that operate for longer than 999.9 seconds. One method is to program consecutive timers, with the Completion Flag of each timer used to activate the next timer. A simple example with two 900.0-second (15-minute) timers combined to functionally form a 30-minute timer.



Address	Instruction	Operands
000	LD	0000
001	TIM	01
		# 9000
002	LD	TIM 01
003	TIM	02
		# 9000
004	LD	TIM 02
005	OUT	0100

In this example, bit 0100 will be turned ON 30 minutes after bit 0000 goes ON.

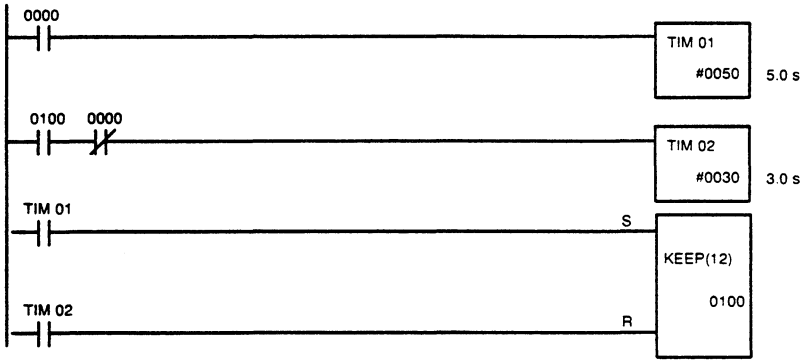
TIM can also be combined with CNT or CNT can be used to count dedicated clock pulse bits to produce longer timers. An example is provided in *3–7–18 COUNTER - CNT*.

Example 3: ON/OFF Delays

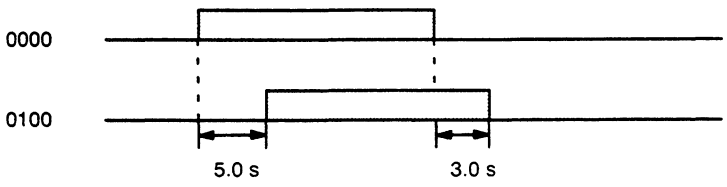
TIM can be combined with KEEP(12) to delay turning a bit ON and OFF in reference to a desired execution condition. KEEP(12) is described *3–7–9 KEEP - KEEP(12)*.

To create delays, the Completion Flags for two TIM are used to determine the execution conditions for setting and reset the bit designated for KEEP(12). The bit whose manipulation is to be delayed is used in KEEP(12). Turning ON and OFF the bit designated for KEEP(12) is thus delayed by the SV for the two TIM. The two SV could naturally be the same if desired.

In the following example, 0100 would be turned ON 5.0 seconds after 0000 goes ON and then turned OFF 3.0 seconds after 0000 goes OFF. It is necessary to use both 0100 and 0000 to determine the execution condition for TIM 02; 0000 in a normally closed condition is necessary to reset TIM 02 when 0000 goes ON and 0100 is necessary to activate TIM 02 (when 0000 is OFF).



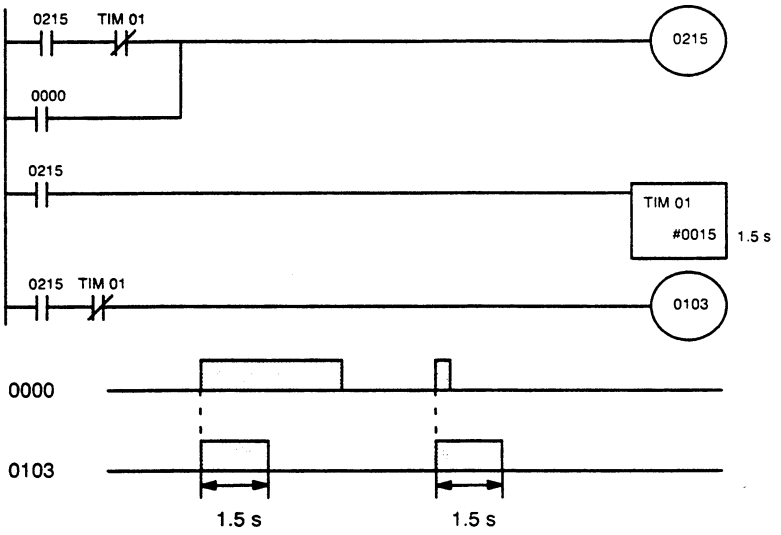
Address	Instruction	Operands
000	LD	0000
001	TIM	01
		# 0050
002	LD	0100
003	AND NOT	0000
004	TIM	02
		# 0030
005	LD	TIM 01
006	LD	TIM 02
007	KEEP(12)	0100



Example 4: One-Shot Bits

The length of time that a bit is kept ON or OFF can be controlled by combining TIM with OUT or OUT NOT. The following diagram demonstrates how this is possible. In this example, bit 0103 would remain ON for 1.5 seconds after 0000 goes ON regardless of the time 0000 stays ON. This is achieved

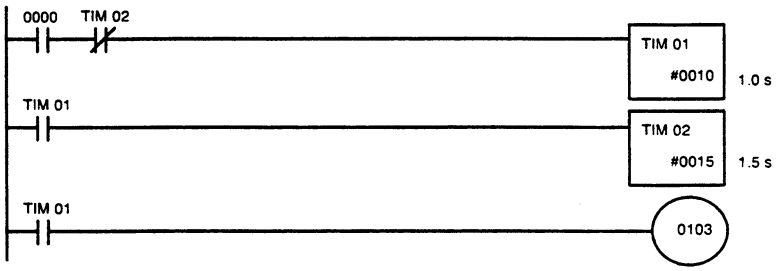
by using 0215 as a self-maintaining bit activated by 0000 and turning ON 0103 through it. When TIM 01 comes ON (i.e., when the SV of TIM 01 has expired), 0103 will be turned OFF through TIM 01 (i.e., TIM 01 will turn ON and because it is programmed as a normally closed condition, an OFF execution condition will be created for OUT 0103).



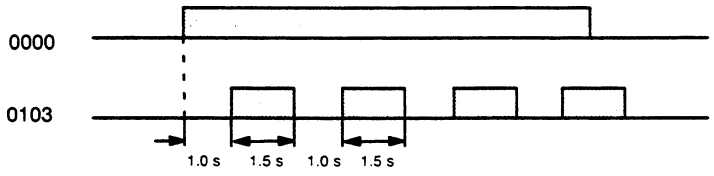
Address	Instruction	Operands
000	LD	0215
001	AND NOT	TIM 01
002	OR	0000
003	OUT	0215
004	LD	0215
005	TIM	01
		# 0015
006	LD	0215
007	AND NOT	TIM 01
008	OUT	0103

Example 5: Flicker Bits

Bits can be programmed to turn ON and OFF at regular intervals while a designated execution condition is ON by using TIM twice. One TIM functions to turn ON and OFF a specified bit, i.e., the Completion Flag of this TIM turns the specified bit ON and OFF. The other TIM functions to control the operation of the first TIM, i.e., when the first TIM's Completion Flag goes ON, the second TIM is started and when the second TIM's Completion Flag goes ON, the first TIM is started.



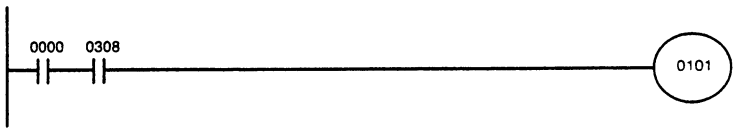
Address	Instruction	Operands
000	LD	0000
001	AND NOT	TIM 02
002	TIM	01
		# 0010
003	LD	TIM 01
004	TIM	02
		# 0015
005	LD	TIM 01
006	OUT	0103



A simpler but less flexible method of creating a flicker bit is to AND one of the dedicated clock pulse bits with the execution condition that is to be ON when the flicker bit is operating. Although this method does not use TIM, it is included here for comparison. This method is more limited because the ON and OFF times must be the same and they depend on the clock pulse bits available.

In the following example the 1-second clock pulse is used (0308) so that 0101 would be turned ON and OFF every second, i.e., it would be ON for 0.5

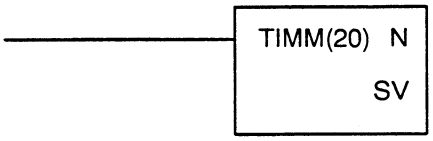
seconds and OFF for 0.5 seconds. Precise timing and the initial status of 0101 would depend on the status of the clock pulse when 0000 goes ON.



Address	Instruction	Operands
000	LD	0000
001	AND	0308
002	OUT	0101

3–7–15 TIMER - TIMM(20)

Ladder Symbol



Definer Values

N: TC number
(00 through 15)

Operand Data Areas

SV: Set value (BCD)
SP10: #
SP16, SP20: I/O, work, DR, LR, #

Limitations

SV is between 00.00 and 99.99 seconds. The decimal point is not entered.

Each TC number can be used as the definer in only one timer or counter instruction.

TC 11 through TC 15 should not be used in TIMM(20) if they are required for the specific instruction to which they are assigned. Refer to the table on page 82.

Description

TIMM(20) operates in the same way as TIM except that TIMM(20) measures in units of 0.01 second.

Refer to *3–7–14 TIMER - TIM* for operational details and examples. Except for the above, and all aspects of operation are the same.

Precautions

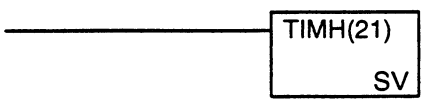
Timers in interlocked program sections are reset when the execution condition for IL(02) is OFF. Power interruptions also reset timers. If a timer that is not reset under these conditions is desired, dedicated clock pulse bits can be counted to produce timers using CNT. Refer to 3–7–18 *COUNTER - CNT* for details.

Flags

ER: The Error Flag (0311) will be turned ON when the SV is set in a word (SP16 and SP20 only) but the content of the indicated word is not BCD. The instruction will be executed, but operation will not be reliable.

3–7–16 HIGH-SPEED TIMER - TIMH(21)

Ladder Symbol



Operand Data Areas

SV: Set value (BCD)
SP10: #
SP16, SP20: I/O, work, DR, LR, #

Limitations

SV is between 0.000 and 9.999 seconds. The decimal point is not entered. In practice, the accuracy of TIMH(21) is limited to the scan time (i.e., because outputs are refreshed only once each scan, the accuracy of TIMH(21) is limited to the order of magnitude of the scan time). Refer to *3-9 Program Execution* for details on the scan time.

The TC number is automatically set to TIM 14 when TIMH(21) is designated and does not need to be input.

Description

TIMH(21) operates in the same way as TIM and TIMM(20) except that TIMH(21) measures in units of 0.001 second.

Refer to *3–7–14 TIMER - TIM* for operational details and examples. Except for the above, and all aspects of operation are the same.

Precautions

Timers in interlocked program sections are reset when the execution condition for IL(02) is OFF. Power interruptions also reset timers. If a timer that is not reset under these conditions is desired, dedicated clock pulse bits can be counted to produce timers using CNT. Refer to *3–7–18 COUNTER - CNT* for details.

Flags

ER: The Error Flag (0311) will be turned ON when the SV is set in a word (SP16 and SP20 only) but the content of the indicated word is not BCD. The instruction will be executed, but operation will not be reliable.

3–7–17 ANALOG TIMER - ATIM(22)

Ladder Symbol

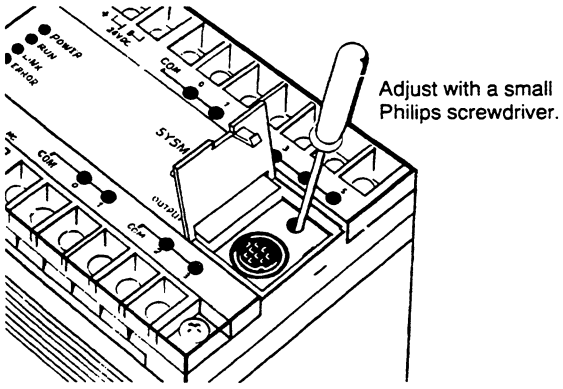
Limitations

The SV is determined by a hardware setting (see below) and does not require numeric input with the instruction.

The TC number is automatically set to TIM 15 when ATIM(22) is designated and does not need to be input.

Description

ATIM(22) operates in the same way as TIM and TIMM(20) except that the SV is determined by the hardware analog timer adjustment on the front of the CPU. The adjustment for the SP10 is shown below. The hardware setting is converted to BCD and stored inside the PC. This setting is between 0.1 and 25.0 seconds. Both ATIM(22) and ATM1(25) are adjusted with the #1 analog timer adjustment on the front of the SP16 and SP20.



Refer to *3–7–14 TIMER - TIM* for other operational details and examples. Except for the above, and all aspects of operation are the same.

Precautions

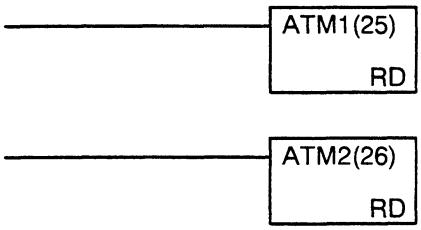
Timers in interlocked program sections are reset when the execution condition for IL(02) is OFF. Power interruptions also reset timers. If a timer that is not reset under these conditions is desired, dedicated clock pulse bits can be counted to produce timers using CNT. Refer to *3–7–18 COUNTER - CNT* for details.

The SV of the analog timer can vary up to 10% with changes in the ambient temperature.

3–7–18 ANALOG TIMER 1 and 2 - ATM1(25) and ATM2(26)

SP16 and SP20 Only

Ladder Symbols



Operand Data Areas

RD: Range designation (BCD)
I/O, work, DR, LR, #

Limitations

The SV cannot be entered directly. A range designation is entered to indicate the range within which the SV is set using a hardware adjustment (see below).

The TC number is automatically set to TIM 11 when ATM1(25) is designated and TIM 12 when ATM2(26) is designated. The TC number cannot be input as any other number.

Description

ATM1(25) and ATM2(26) operate in the same way as TIM and TIMM(20) except that their SVs are determined by the #1 and #2 analog timer adjustments on the front of the CPU. The hardware setting is converted to BCD and stored in dedicated word 08. The ranges within which the hardware adjustment operations is designated as the operand (RD) of the instruction. These designations are shown in the following table.

RD	SV range
0000	1 to 250 seconds
0001	0.1 to 25.0 seconds
0002	0.01 to 2.50 seconds

RD can be designated either as a constant, or as the contents of a word by designated a word address.

Both ATIM(22) and ATM1(25) are adjusted with the #1 analog timer adjustment on the front of the SP16 and SP20. Although both of these instructions can be used at the same time, their SVs cannot be adjusted independently, although the range of the set value for ATM1(25) can be controlled as described above. ATM2(26) is adjusted with the #2 analog timer adjustment and can thus be set independently from other timers.

Refer to *3–7–14 TIMER - TIM* for other operational details and examples. Except for the above, and all aspects of operation are the same.

Precautions

Timers in interlocked program sections are reset when the execution condition for IL(02) is OFF. Power interruptions also reset timers. If a timer not re-

set under these conditions is desired, clock pulse bits can be counted to produce timers using CNT. Refer to *3–7–18 COUNTER - CNT* for details.

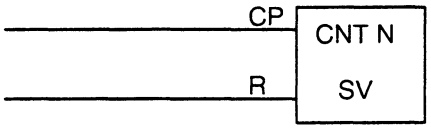
The SV of the analog timer can vary up to 10% with changes in the ambient temperature.

Flags

ER: The Error Flag (0311) will be turned ON when the RD contained in a word is not BCD. The instruction will be executed, but operation will not be reliable.

3–7–19 COUNTER - CNT

Ladder Symbol



Definer Values

N: TC number
(00 through 15)

Operand Data Areas

SV: Set value (BCD)
SP10: #
SP16, SP20: I/O, work, DR, LR, #

Limitations

Each TC number can be used as the definer in only one timer or counter instruction.

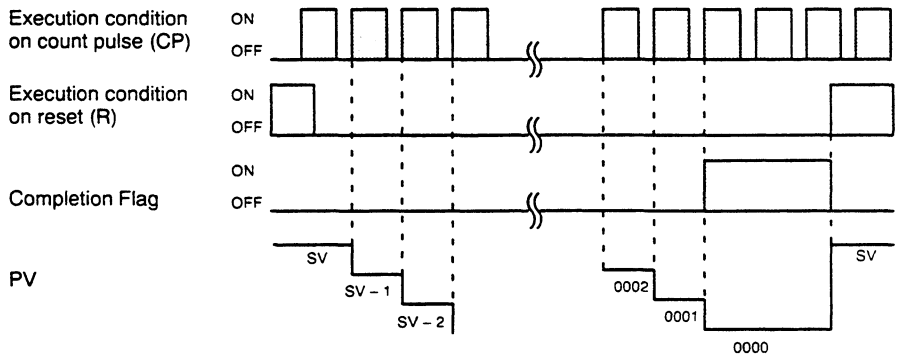
TC 11 through TC 15 should not be used in CNT if they are required for the specific instruction to which they are assigned. Refer to the table on page 82.

Description

CNT is used to count down from SV when the execution condition on the count pulse, CP, goes from OFF to ON, i.e., the present value (PV) will be decremented by one whenever CNT is executed with an ON execution condition for CP and the execution condition was OFF for the last execution. If the execution condition has not changed or has changed from ON to OFF, the PV of CNT will not be changed. The Completion Flag for a counter is turned ON when the PV reaches zero and will remain ON until the counter is reset.

CNT is reset with a reset input, R. When R goes from OFF to ON, the PV is reset to SV. The PV will not be decremented while R is ON. Counting down from SV will begin again when R goes OFF. The PV for CNT will not be reset in interlocked program sections or by power interruptions.

Changes in execution conditions, the Completion Flag, and the PV are illustrated below. PV line height is meant only to indicate changes in the PV.

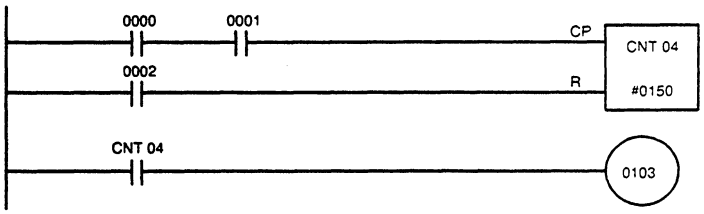


Flags

ER: The Error Flag (0311) will be turned ON when the SV is contained in a word (SP16 and SP20 only) but the content of the indicated word is not BCD. The instruction will be executed, but operation will not be reliable.

Example 1: Basic Application

In the following example, the PV will be decremented whenever both 0000 and 0001 are ON provided that 0002 is OFF and either 0000 or 0001 was OFF the last time CNT 04 was executed. When 150 pulses have been counted down (i.e., when PV reaches zero), 0103 will be turned ON.

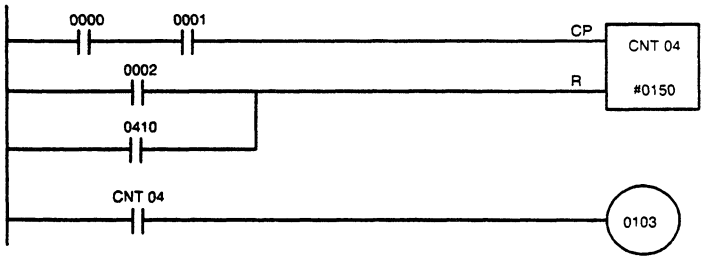


Address	Instruction	Operands
000	LD	0000
001	AND	0001
002	LD	0002
003	CNT	04
		# 0150
004	LD	CNT 04
005	OUT	0103

Here, 0000 can be used to control when CNT is operative and 0001 can be used as the bit whose OFF to ON changes are being counted.

Example 2: Reset for Power Interruptions

The above CNT can be modified to restart from SV each time power is turned ON to the PC. This is done by using the First Scan Flag (0410) to reset CNT as shown below.



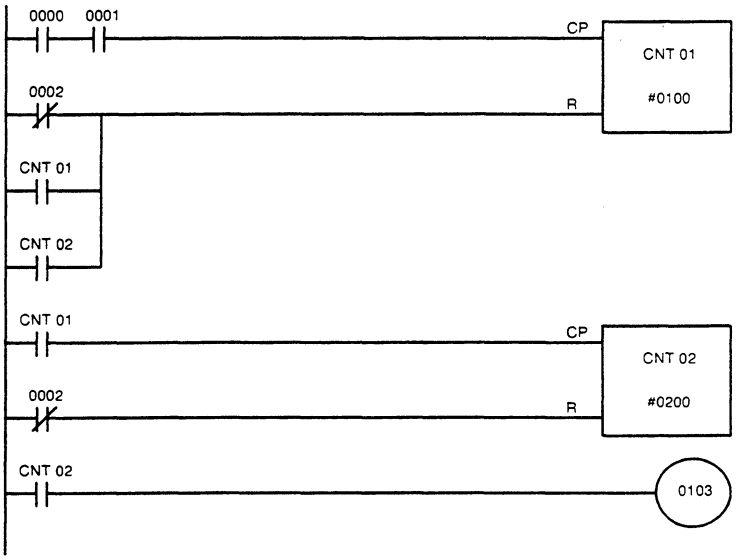
Address	Instruction	Operands
000	LD	0000
001	AND	0001
002	LD	0002
003	OR	0410
004	CNT	04
		# 0150
005	LD	CNT 04
006	OUT	0103

Example 3: Extended Counter

Counters that can count past 9,999 can be programmed by using one CNT to count the number of times another CNT has counted to zero from SV.

In the following example, 0000 is used to control when CNT 01 operates. CNT 01, when 0000 is ON, counts down the number of OFF to ON changes in 0001. CNT 01 is reset by its Completion Flag, i.e., it starts counting again as soon as its PV reaches zero. CNT 02 counts the number of times the Completion Flag for CNT 01 goes ON. Bit 0002 serves as a reset for the entire extended counter, resetting both CNT 01 and CNT 02 when it is OFF. The Completion Flag for CNT 02 is also used to reset CNT 01 to inhibit CNT 01 operation, once SV for CNT 02 has been reached, until the entire extended counter is reset via 0002.

Because in this example the SV for CNT 01 is 100 and the SV for CNT 02 is 200, the Completion Flag for CNT 02 turns ON when 100 x 200 or 20,000 OFF to ON changes have been counted in 0001. This would result in 0103 being turned ON.



Address	Instruction	Operands	
000	LD		0000
001	AND		0001
002	LD NOT		0002
003	OR	CNT	01
004	OR	CNT	02
005	CNT		01
		#	0100
006	LD	CNT	01
007	LD NOT		0002
008	CNT		02
		#	0200
009	LD	CNT	02
010	OUT		0103

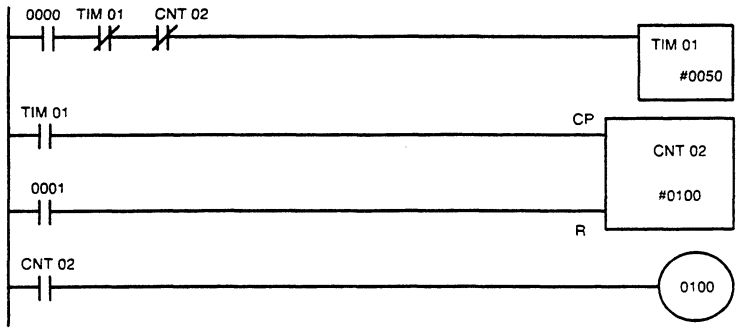
CNT can be used in sequence as many times as required to produce counters capable of counting any desired values.

Example 4: Extended Timers

CNT can be used to create extended timers in two ways: by combining TIM with CNT and by counting dedicated clock pulse bits.

In the following example, CNT 02 counts the number of times TIM 01 reaches zero from its SV. The Completion Flag for TIM 01 is used to reset TIM 01 so that is runs continuously and CNT 02 counts the number of times the Completion Flag for TIM 01 goes ON (CNT 02 would be executed once each time between when the Completion Flag for TIM 01 goes ON and TIM 01 is reset by its Completion Flag). TIM 01 is also reset by the Completion Flag for CNT 02 so that the extended timer would not start again until CNT 02 was reset by 0001, which serves as the reset for the entire extended timer.

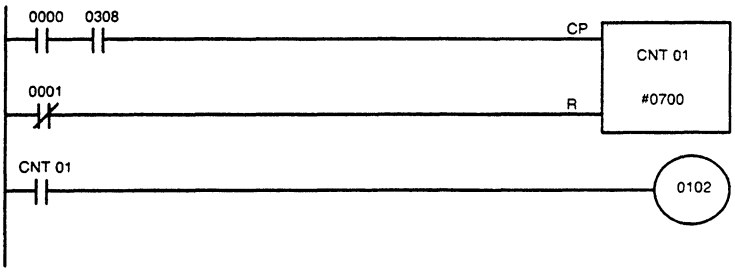
Because in this example the SV for TIM 01 is 5.0 seconds and the SV for CNT 02 is 100, the Completion Flag for CNT 02 turns ON when 5 seconds x 100 times, i.e., 500 seconds (or 8 minutes and 20 seconds) have expired. This would result in bit 0100 being turned ON.



Address	Instruction	Operands	
000	LD		0000
001	AND NOT	TIM	01
002	AND NOT	CNT	02
003	TIM		01
		#	0050
004	LD	TIM	01
005	LD		0001
006	CNT		02
		#	0100
007	LD	CNT	02
008	OUT		0100

In the following example, CNT 01 counts the number of times the 1-second clock pulse bit (0308) goes from OFF to ON. Here again, 0000 is used to control the times when CNT is operating.

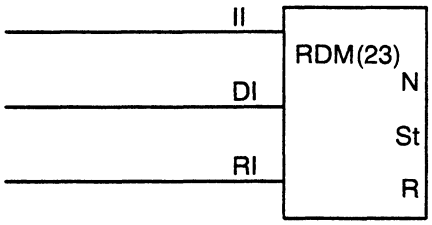
Because in this example the SV for CNT 01 is 700, the Completion Flag for CNT 02 turns ON when 1 second x 700 times, or 11 minutes and 40 seconds have expired. This would result in 0102 being turned ON.



Address	Instruction	Operands
000	LD	0000
001	AND	0308
002	LD NOT	0001
003	CNT	01
		# 0700
004	LD	CNT 01
005	OUT	0102

3–7–20 REVERSIBLE DRUM COUNTER -RDM(23)

Ladder Symbol



Definer Values

N: TC number
(00 to 15)

Operand Data Areas

St: Starting table word
DR

R : Results word
Output bits, work bits, DR, LR

Limitations

St must be between 0000 and 0003. All unused bits in R can be used as work bits.

TC 11 through TC 15 should not be used in RDM(23) if they are required for the specific instruction to which they are assigned. Refer to the table on page 82.

Description

The reversible drum counter is a ring counter with a counting range of 0000 to 9999. It uses three execution conditions, the count input (II), the decrement/increment input (DI) and the reset input (RI).

RDM(23) is executed each time the ON execution condition for II has changed from OFF to ON since the last scan, i.e., on the rising edge of II. RDM(23) increments the present value if DI is OFF and decrements the present value if it is ON.

The value (n) in St indicates the number of comparison ranges with which the PV is to be compared. Up to 6 ranges are possible (4 in the SP10). The number of ranges is one greater than the value in St. The present value of the counter is compared with the upper and lower limits of a set of ranges which have been preset in St+1 through St+2(n+1).

If the lower limit is less than the upper limit, the corresponding bit of the result word, R, will be turned ON whenever the present value is within the preset range. The bit in the result word will remain ON until the current value is no longer within the specified range.

If the lower limit is greater than the upper limit, the corresponding bit of the result word, R, will be turned ON whenever the present value is outside of the preset range. The bit in the result word will remain ON until the current value moves within the specified range.

When the reset input (RI) goes ON, the present value is reset to 0000.

Upper and Lower Limit Settings

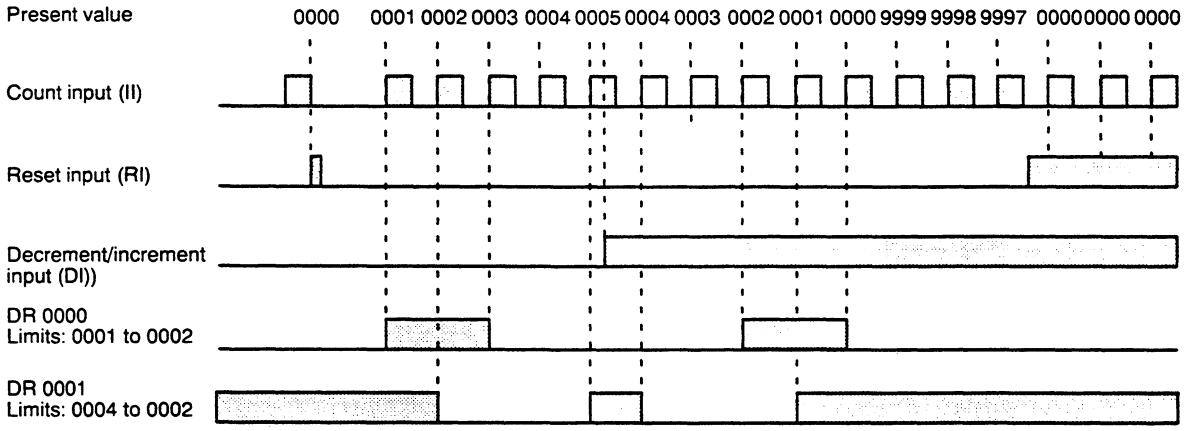
The following table shows the upper and lower limits that need to be set in St + 1 through St + 2n +2. PV is the present value of the counter.

Lower limit	Upper limit	Present value of the counter	Corresponding bit of R	Applicable PCs
St + 1	St + 2	Value of St + 1 ≤ PC ≤ value of St + 2	00	SP10, SP16, SP20
St + 3	St + 4	Value of St + 3 ≤ PC ≤ value of St + 4	01	
St + 5	St + 6	Value of St + 5 ≤ PC ≤ value of St + 6	02	
St + 7	St + 8	Value of St + 7 ≤ PC ≤ value of St + 8	03	
St + 9	St + 10	Value of St + 9 ≤ PC ≤ value of St + 10	04	SP16, SP20
St + 11	St + 12	Value of St + 11 ≤ PC ≤ value of St + 12	05	

The values must be four-digit BCD in the range 0000 through 9999.

Timing Example

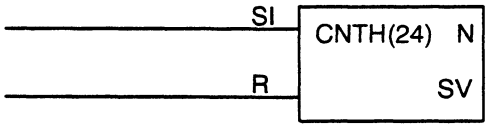
The following timing example uses DR 00 as the results word. Here, the first range is 0001 to 0002 (the content of St+1 is 0001 and St+2 is 0002), and the second range is 0004 to 0002 (the content of St+3 is 0004 and St+4 is 0002).



3–7–21 HIGH-SPEED COUNTER - CNTH(24)

SP16 and SP20 Only

Ladder Symbol



Operand Data Areas

SV: Set value (BCD)
I/O, work, DR, LR, #

Limitations

The TC number is automatically set to CNT 13, the count pulse (CP) to input bit 0000, and the hard reset input (R) to input bit 0001 when CNTH(24) is designated. Inputs 0000 and 0001 cannot be used as normal input terminals when CNTH(24) is being used.

Description

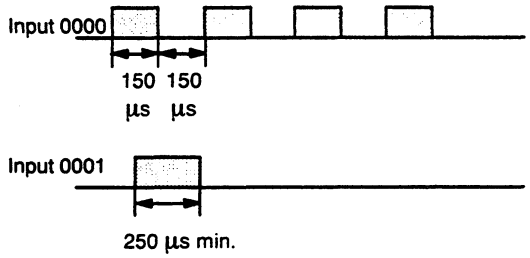
CNTH(24) is a high-speed incrementing counter. The present value (PV) begins at 0000 and will be incremented by one whenever CP (input bit 0000) goes from OFF to ON as long as the start input condition (SI) is ON and the reset input (R) is OFF. The start input and reset input conditions are entered with LD before CNTH(24) is entered. The Completion Flag, CNT 13, is turned ON when the PV reaches the SV and will remain ON for one scan only. When the SV is reached, the PC will be automatically reset to zero.

The maximum counter value can be set by setting the SV to 0000 rather than to 9999, i.e., the counter will count to 10,000 when the SV is set to 0000.

CNTH(24) is reset with R. When R goes from OFF to ON, the PV is reset to zero. The PV will not be incremented while R is ON. Counting from zero will begin again when R goes OFF. The PV for CNT 13 will not be reset in interlocked program sections or by power interruptions.

CNTH(24) counting is enabled with SI. When SI is OFF, the PV is not changed even if R is OFF and CP goes from OFF to ON. Counting will resume when SI is turned ON again.

The count pulse for CNTH(24) is input bit 0000 and the hard reset input is input bit 0001. The count signal must be at least 150 μs (2 kHz) wide and have a duty factor of 1:1, and the reset signal must have an ON time of at least 250 μs, as shown below.

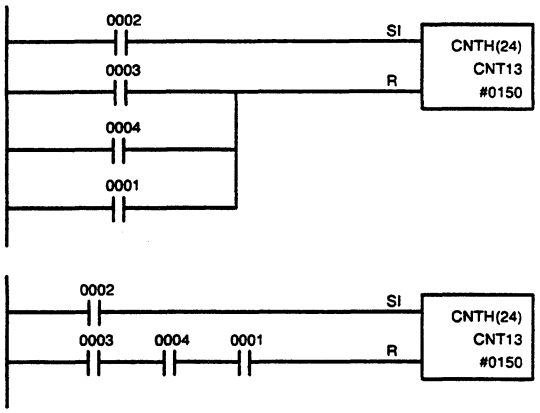


Inputs 0000 and 0001 can be used as normal inputs when CNTH(24) is not used, but the input signals must be 1 kHz max. (500 ms wide min.).

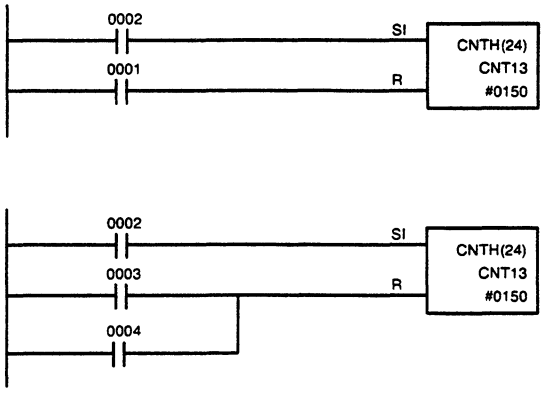
Precautions

Do not use the 0001 reset input in combination with other input bits in the reset execution condition.

Incorrect



Correct



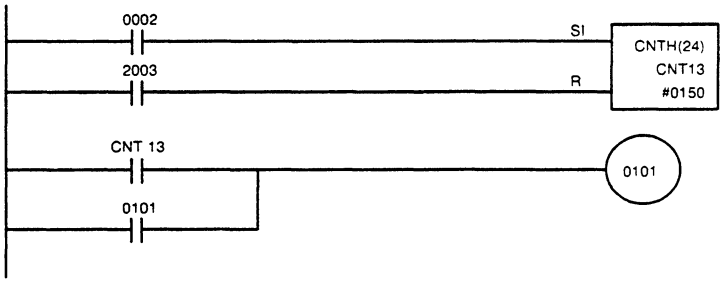
Flags

ER: The Error Flag (0311) will be turned ON when the SV is not BCD. The instruction will be executed, but operation will not be reliable.

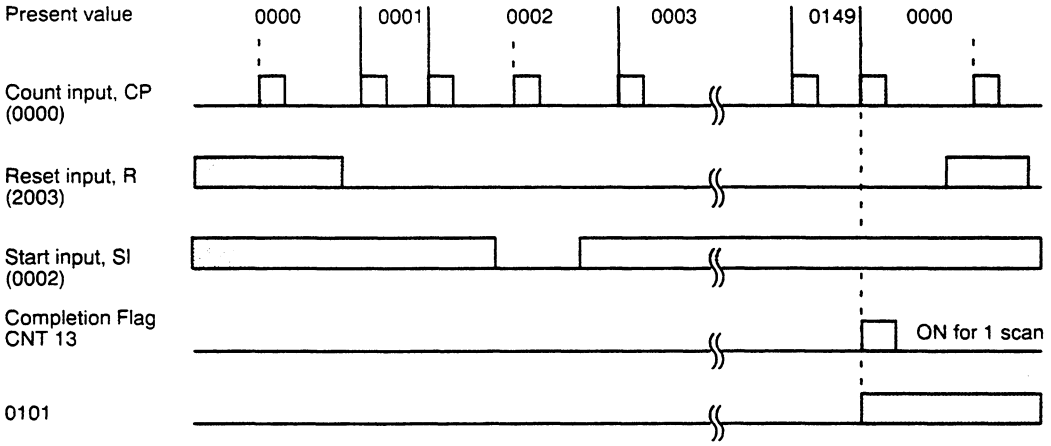
Example 1: Basic Application

In the following example, the PV will be incremented whenever the count pulse, 0000, goes from OFF to ON provided that the start input, 0002, is ON and the reset input, 2003 is OFF. When 150 pulses have been counted (i.e.,

when the PV reaches the SV), the Completion Flag, CNT 13, and 0101 will be turned ON.

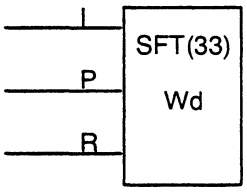


Address	Instruction	Operands
000	LD	0002
001	LD	2003
002	CNTH(24)	CNT 13
		# 0150
003	LD	CNT 13
004	OR	0101
005	OUT	0101



3–7–22 SHIFT REGISTER - SFT(33)

Ladder Symbol



Operand Data Areas

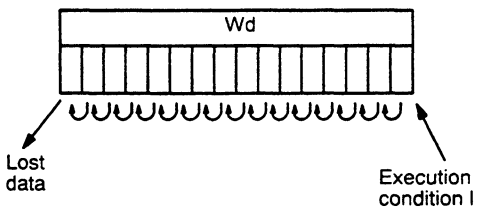
Wd: Shift word
Output bits, work bits, DR, LR

Limitations

A maximum of 16 SFT(33) instructions can be used in any one program.

Description

SFT(33) is controlled by three execution conditions, I, P, and R. If SFT(33) is executed and 1) execution condition P is ON and was OFF the last execution and 2) R is OFF, then execution condition I is shifted into the rightmost bit of a shift register defined between St and E, i.e., if I is ON, a 1 is shifted into the register; if I is OFF, a 0 is shifted in. When I is shifted into the register, all bits previously in the register are shifted to the left and the leftmost bit of the register is lost.

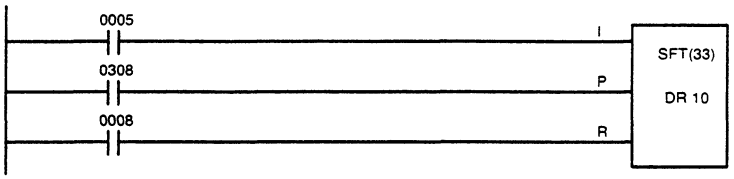


The execution condition on P functions like a differentiated instruction, i.e., I will be shifted into the register only when P is ON and was OFF the last time SFT(33) was executed. If execution condition P has not changed or has gone from ON to OFF, the shift register will remain unaffected.

When execution condition R goes ON, all bits in the shift register will be turned OFF (i.e., reset to 0) and the shift register will not operate until R goes OFF again.

Example 1: Basic Application

The following example uses the 1-second clock pulse bit (0308) so that the execution condition produced by 0005 is shifted into register DR 10 every second.

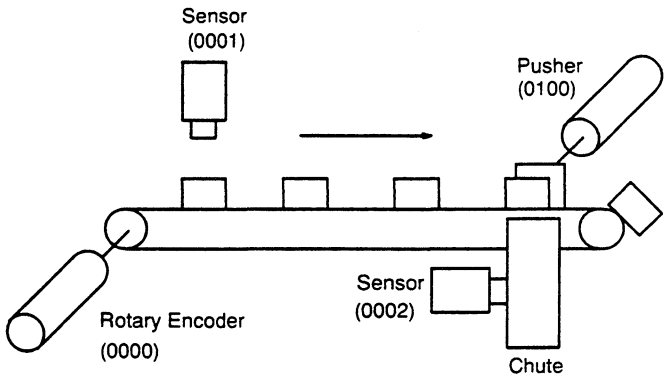


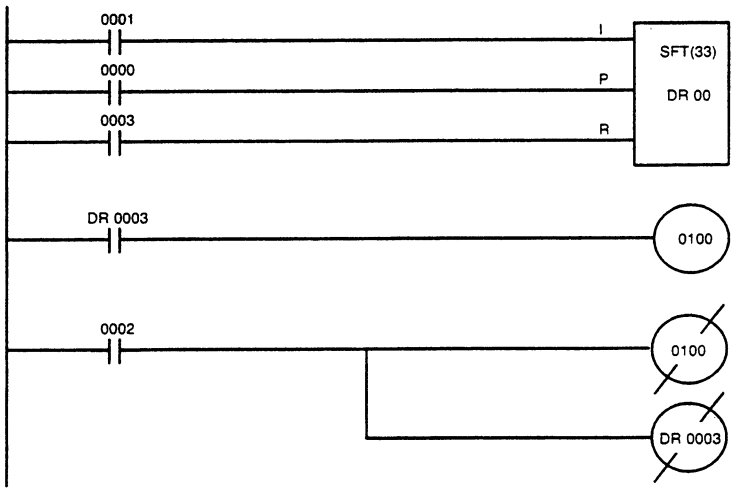
Address	Instruction	Operands
000	LD	0005
001	LD	0308
002	LD	0008
003	SFT(33)	
		DR 10

Example 2: Control Action

The following program controls the conveyor line shown below so that faulty products detected at the sensor are pushed down a chute. To do this, the execution condition determined by inputs from the first sensor (0001) are stored in a shift register: ON for good products; OFF for faulty ones. Conveyor speed has been adjusted so that DR 0003 of the shift register can be used to activate a pusher (0100) when a faulty product reaches it, i.e., when DR 0003 turns ON, 0100 is turned ON to activate the pusher.

The program is set up so that a rotary encoder (0000) controls execution of SFT(33) through a DIFU(10), the rotary encoder is set up to turn ON and OFF each time a product passes the first sensor. Another sensor (0002) is used to detect faulty products in the chute so that the pusher output and DR 0003 of the shift register can be reset as required.

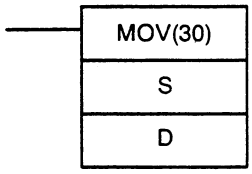




Address	Instruction	Operands	
000	LD		0001
001	LD		0000
002	LD		0003
003	SFT(33)		
		DR	00
004	LD	DR	0003
005	OUT		0100
006	LD		0002
007	OUT NOT		0100
008	OUT NOT	DR	0003

3–7–23 MOVE - MOV(30)

Ladder Symbol

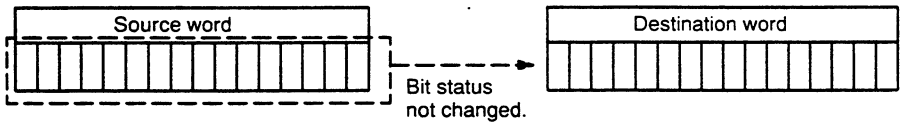


Operand Data Areas

S: Source word
I/O, work, dedicated (03 only), DR, LR, TC, #

D: Destination word
Output bits, work bits, DR, LR

Description When the execution condition is OFF, MOV(30) is not executed. When the execution condition is ON, MOV(30) copies the content of S to D.



Precautions MOV(30) will be executed every scan unless programmed with DIFU(10) or DIFD(11).

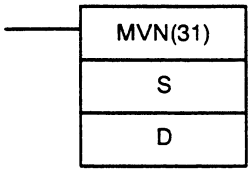
Flags

ER: Indirectly addressed DR word is non-existent. (Content of *DR word is not BCD, or the DR area boundary has been exceeded.)

EQ: ON when all zeros are transferred to D.

3–7–24 MOVE NOT - MVN(31)

Ladder Symbol

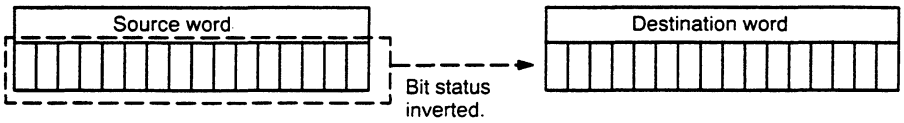


Operand Data Areas

S: Source word
I/O, work, dedicated (03 only), DR, LR, TC, #

D: Destination word
Output bits, work bits, DR, LR

Description When the execution condition is OFF, MVN(31) is not executed. When the execution condition is ON, MVN(31) transfers the complement of S to D, i.e., for each ON bit in S, the corresponding bit in D is turned OFF, and for each OFF bit in S, the corresponding bit in D is turned ON.



Precautions

MVN(31) will be executed every scan unless programmed with DIFU(10) or DIFD(11).

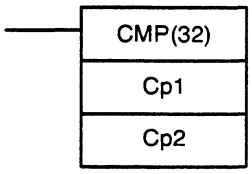
Flags

ER: Indirectly addressed DR word is non-existent. (Content of *DR word is not BCD, or the DR area boundary has been exceeded.)

EQ: ON when all zeros are transferred to D.

3–7–25 COMPARE - CMP(32)

Ladder Symbol



Operand Data Areas

Cp1: First compare word
I/O, work, dedicated (03 only), DR, LR, TC, #

Cp2: Second compare word
I/O, work, dedicated (03 only), DR, LR, TC, #

Limitations

When comparing a value to the PV of a timer or counter, the value must be in BCD.

Description

When the execution condition is OFF, CMP(32) is not executed. When the execution condition is ON, CMP(32) compares Cp1 and Cp2 and outputs the result to the GR, EQ, and LE flags in the dedicated area.

Precautions

Placing other instructions between CMP(32) and the operation which accesses the EQ, LE, and GR flags may change the status of these flags. Be sure to access them before the desired status is changed.

Flags

ER: Indirectly addressed DR word is non-existent. (Content of *DR word is not BCD, or the DR area boundary has been exceeded.)

EQ: ON if Cp1 equals Cp2.

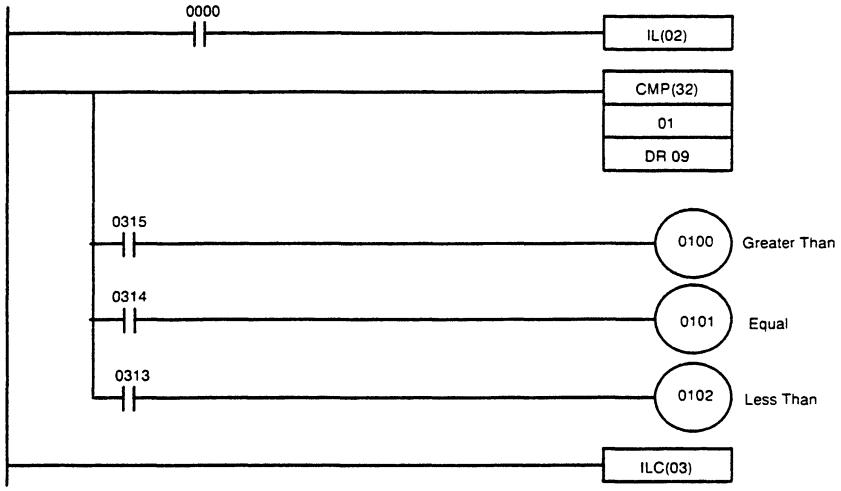
LE: ON if Cp1 is less than Cp2.

GR: ON if Cp1 is greater than Cp2.

Example 1: Saving CMP(32) Results

The following example shows how to save the comparison result immediately. If the content of 01 is greater than that of DR 09, 0100 is turned ON; if the two contents are equal, 0101 is turned ON; if content of 01 is less than that of DR 09, 0102 is turned ON. In some applications, only one of the three OUTs would be necessary, making the use of TR 0 unnecessary. With this type of

programming, 0100, 0101, and 0102 are changed only when CMP(32) is executed.

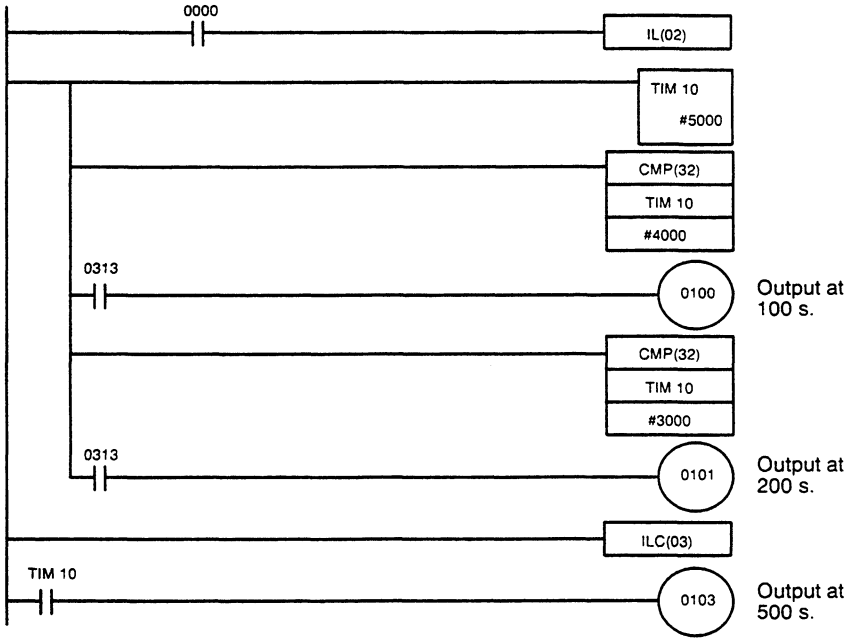


Address	Instruction	Operands
000	LD	0000
001	IL(02)	
002	CMP(32)	
		01
		DR 09
003	LD	0315

Address	Instruction	Operands
004	OUT	0100
005	LD	0314
006	OUT	0101
007	LD	0313
008	OUT	0102
009	ILC(03)	

Example 2: Obtaining Indications during Timer Operation

The following example uses TIM, CMP(32), and the LE flag (0313) to produce outputs at particular times in the timer's countdown. The timer is started by turning ON 0000. When 0000 is OFF, TIM 10 is reset and the second two CMP(32)s are not executed (i.e., executed with OFF execution conditions). Output 0100 is produced after 100 seconds; output 0101, after 200 seconds; and output 0103, after 500 seconds.



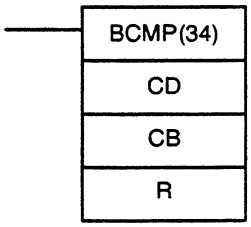
Address	Instruction	Operands	
000	LD		0000
001	IL(02)		
002	TIM		10
		#	5000
003	CMP(32)		
		TIM	10
		#	4000
004	AND		0313
005	OUT		0100

Address	Instruction	Operands	
006	CMP(32)		
		TIM	10
		#	3000
007	AND		0313
008	OUT		0101
009	ILC(03)		
010	LD	TIM	10
011	OUT		0103

3–7–26 BLOCK COMPARE - BCMP(34)

SP16 and SP20 Only

Ladder Symbol



Operand Data Areas

CD: Compare data
I/O, work, dedicated (03 only), DR, LR, TC, #

CB: First comparison block word
DR (00 to 13 only)

R: Result word
I/O (01 only), work, DR, LR

Limitations

All data must be in BCD. Press the CONT/# Key before entering a constant for CD.

Description

N is the rightmost digit of CB and determines the size of the comparison block; there will be N+1 comparison ranges. BCMP(34) compares CD to the ranges defined by a block consisting of CB+1, CB+2, ..., CB+(2N+2). Each range is defined by two words, the first one providing the lower limit and the second word providing the upper limit, as shown below. If the lower limit is less than the upper limit, the corresponding bit of the result word, R, will be turned ON whenever CD is within the preset range.

Table Comparisons	**Bit in R**
CB+1 ≤ CD ≤ CB+2	Bit 00
CB+3 ≤ CD ≤ CB+4	Bit 01
. . .	
. . .	
. . .	
CB+ (2N+1)≤ CD ≤ CB+(2N+2)	Bit N

If the lower limit is greater than the upper limit, the corresponding bit of the result word will be turned ON whenever CD is not within the preset range.

Table Comparisons	**Bit in R**
CD ≤ CB+1 or CB+2 ≤ CD	Bit 00
CD ≤ CB+3 or CB+4 ≤ CD	Bit 01
. . .	
. . .	
. . .	
CB+(2N+1) ≤ CD or CB+(2N+2) ≤ CB	Bit N

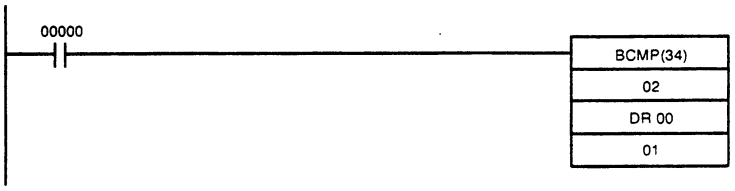
If the content of CB or the table data are changed during execution, execution will continue with the new values.

Flags

ER: Indirectly addressed DM word is non-existent. (Content of *DM word is not BCD, or the DM area boundary has been exceeded.)

Example

The following example shows the comparisons made and the results provided for BCMP(34). Here, the comparison is made during each scan when 0000 is ON. The rightmost digit of CB (DR 00) is 5, so the comparison block is CB+1 to CB+(2N+2) or DR 01 to DR 12.



Address	Instruction	Operands
000	LD	0000
001	BCMP(34)	
		02
		DR 00
		01

CB: DR 00

DR 00	0005

Comparison block: CB+1 to CB+(2N+2) or DR 01 to DR12.

CD: 02

02	0210

Compare data in 02 (which contains 0210).

Lower limits

DR 01	0000
DR 03	0101
DR 05	0201
DR 07	0501
DR 09	1401
DR 11	1501

Upper limits

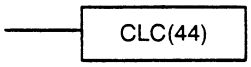
DR 02	0100
DR 04	0200
DR 06	0300
DR 08	0600
DR 10	1500
DR 12	1600

R: 01

0100	0
0101	0
0102	1
0103	0
0104	0
0105	0

3–7–27 CLEAR CARRY - CLC(44)

Ladder Symbol

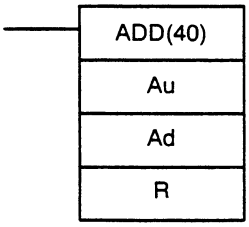


When the execution condition is OFF, CLC(44) is not executed.When the execution condition is ON, CLC(44) turns OFF CY (0312).

3–7–28 BCD ADD - ADD(40)

Operand Data Areas

Ladder Symbol



Au: Augend word (BCD)
I/O, work, dedicated (03 only), DR, LR, TC, #

Ad: Addend word (BCD)
I/O, work, dedicated (03 only), DR, LR, TC, #

R: Result word
Output bits, work bits, DR, LR

Description

When the execution condition is OFF, ADD(40) is not executed. When the execution condition is ON, ADD(40) adds the contents of Au, Ad, and CY, and places the result in R. CY will be set if the result is greater than 9999.

Au + Ad + CY → CY R

Flags

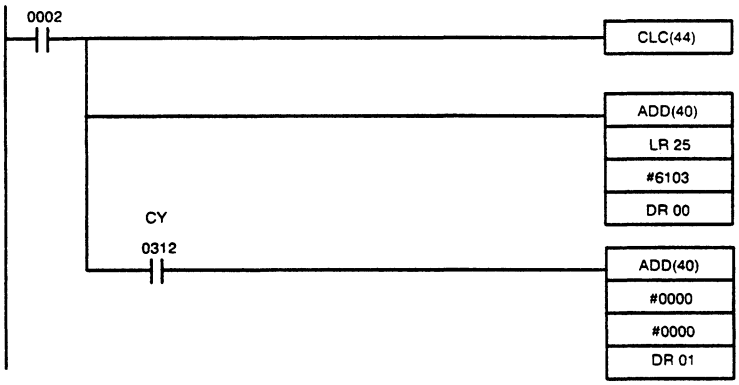
ER: Au and/or Ad is not BCD.
Indirectly addressed DR word is non-existent. (Content of *DR word is not BCD, or the DR area boundary has been exceeded.)

CY: ON when there is a carry in the result.

EQ: ON when the result is 0.

Example

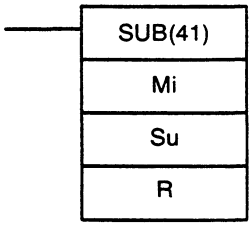
If 0002 is ON, the program represented by the following diagram clears CY with CLC(44), adds the content of LR 25 to a constant (6103), places the result in DR 0100, and then moves either all zeros or 0001 into DR 0101 depending on the status of CY (0312). This ensures that any carry from the last digit is preserved in R+1 so that the entire result can be later handled as eight-digit data.



Address	Instruction	Operands	
000	LD		0002
001	CLC(44)		
002	AND(40)		
		LR	25
		#	6103
		DR	00
003	AND		0312
004	AND(40)		
		#	0000
		#	0000
		DR	01

3–7–29 BCD SUBTRACT - SUB(41)

Ladder Symbol



Operand Data Areas

Mi: Minuend word (BCD)
I/O, work, dedicated (03 only), DR, LR, TC, #
Su: Subtrahend word (BCD)
I/O, work, dedicated (03 only), DR, LR, TC, #
R: Result word
Output bits, work bits, DR, LR

Description

When the execution condition is OFF, SUB(41) is not executed. When the execution condition is ON, SUB(41) subtracts the contents of Su and CY from Mi, and places the result in R. If the result is negative, CY is set and the 10's complement of the actual result is placed in R. To convert the 10's complement to the true result, subtract the content of R from zero (see example below).

Mi − Su − CY → CY R

Flags

ER: Mi and/or Su is not BCD.
Indirectly addressed DR word is non-existent. (Content of *DR word is not BCD, or the DR area boundary has been exceeded.)

CY: ON when the result is negative, i.e., when Mi is less than Su plus CY.

EQ: ON when the result is 0.

Caution Be sure to clear the carry flag with CLC(44) before executing SUB(41) if its previous status is not required, and check the status of CY after doing a subtraction with SUB(41). If CY is ON as a result of executing SUB(41) (i.e., if the result is negative), the result is output as the 10's complement of the true answer. To convert the output result to the true value, subtract the value in R from 0.

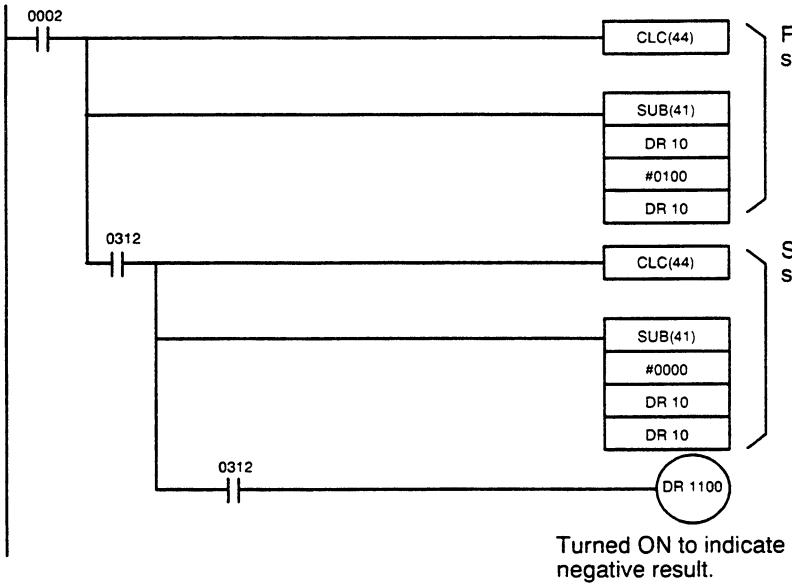
Example

When 0002 is ON, the following ladder program clears CY, subtracts the contents of DR 0100 and CY from the content of 010 and places the result in DR 00.

If CY is set by executing SUB(41), the result in DR 00 is subtracted from zero (note that CLC(44) is again required to obtain an accurate result), the result is placed back in DR 00, and DR 2100 is turned ON to indicate a negative result.

If CY is not set by executing SUB(41), the result is positive, the second subtraction is not performed, and DR 1000 is not turned ON. DR 1000 is programmed as a self-maintaining bit so that a change in the status of CY will not turn it OFF when the program is rescanned.

In this example, differentiated forms of SUB(41) are used so that the subtraction operation is performed only once each time 0002 is turned ON. When another subtraction operation is to be performed, 0002 will need to be turned OFF for at least one scan (resetting DR 1000) and then turned back ON.



Address	Instruction	Operands	
000	LD		0002
001	CLC(44)		
002	SUB(41)		
		DR	10
		#	0100
		DR	10
003	AND		0312
004	CLC(44)		
005	SUB(41)		
		#	0000
		DR	10
		DR	10
006	AND		0312
007	OUT	DR	1100

The first and second subtractions for this diagram are shown below using example data for DR 10.

Note The actual SUB(41) operation involves subtracting Su and CY from 10,000 plus Mi. For positive results the leftmost digit is truncated. For negative results the 10s complement is obtained. The procedure for establishing the correct answer is given below.

First Subtraction

DR 10	0089	
#	– 0100	
CY	– 0	
DR 10	9989	(0089 + (10,000 – 0100))
CY	1	(negative result)

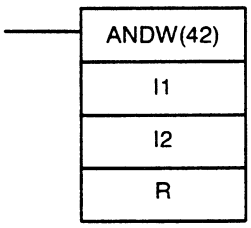
Second Subtraction

#	0000	
DR 10	–9989	
CY	–0	
DR 10	0011	(0000 + (10,000 – 9989))
CY	1	(negative result)

In the above case, the program would turn ON DR 1100 to indicate that the value held in DR 10 is negative.

3–7–30 AND WORD- ANDW(42)

Ladder Symbol



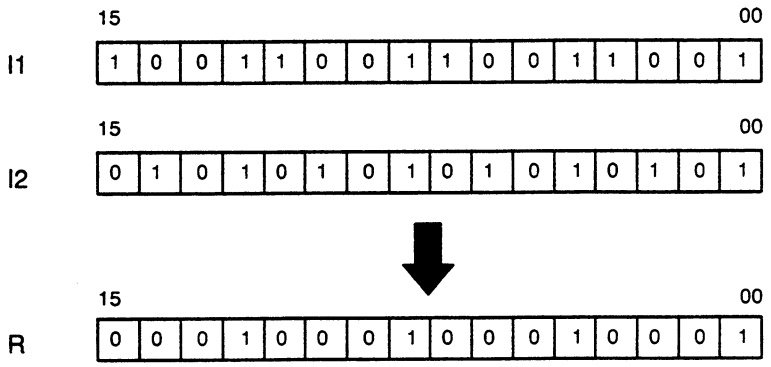
Operand Data Areas

I1: Input 1
I/O, work, dedicated (03 only), DR, LR, TC, #
I2: Input 2
I/O, work, dedicated (03 only), DR, LR, TC, #
R: Result word
Output bits, work bits, DR, LR

Description

When the execution condition is OFF, ANDW(42) is not executed. When the execution condition is ON, ANDW(42) logically AND's the contents of I1 and I2 bit-by-bit and places the result in R.

Example



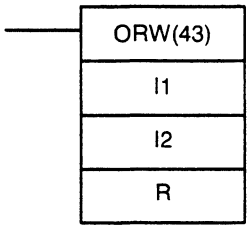
Flags

ER: Indirectly addressed DR word is non-existent. (Content of *DR word is not BCD, or the DR area boundary has been exceeded.)

EQ: ON when the result is 0.

3–7–31 OR WORD - ORW(43)

Ladder Symbol



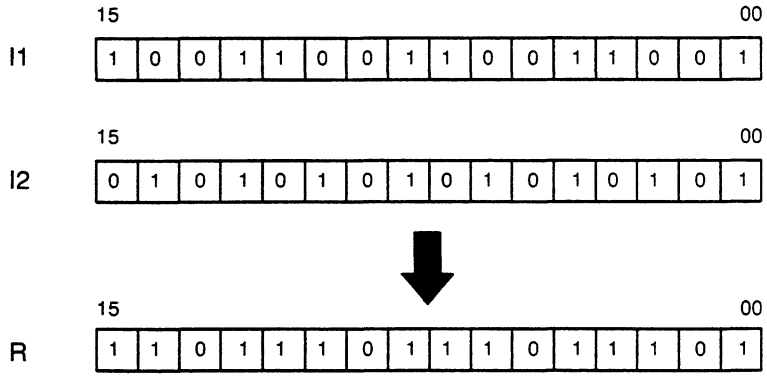
Operand Data Areas

I1: Input 1
I/O, work, dedicated (03 only), DR, LR, TC, #
I2: Input 2
I/O, work, dedicated (03 only), DR, LR, TC, #
R: Result word
Output bits, work bits, DR, LR

Description

When the execution condition is OFF, ORW(43) is not executed. When the execution condition is ON, ORW(43) logically OR's the contents of I1 and I2 bit-by-bit and places the result in R.

Example



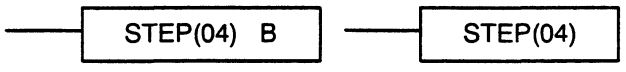
Flags

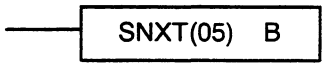
ER: Indirectly addressed DR word is non-existent. (Content of *DR word is not BCD, or the DR area boundary has been exceeded.)

EQ: ON when the result is 0.

3–7–32 STEP DEFINE and STEP START-STEP(04)/SNXT(05)

Ladder Symbols





Definer Data Areas

B: Control bit
Output bits, work bits, DR, LR

B: Control bit
Output bits, work bits, DR, LR

Limitations

Control bits within one section of step programming must be sequential and from the same word.

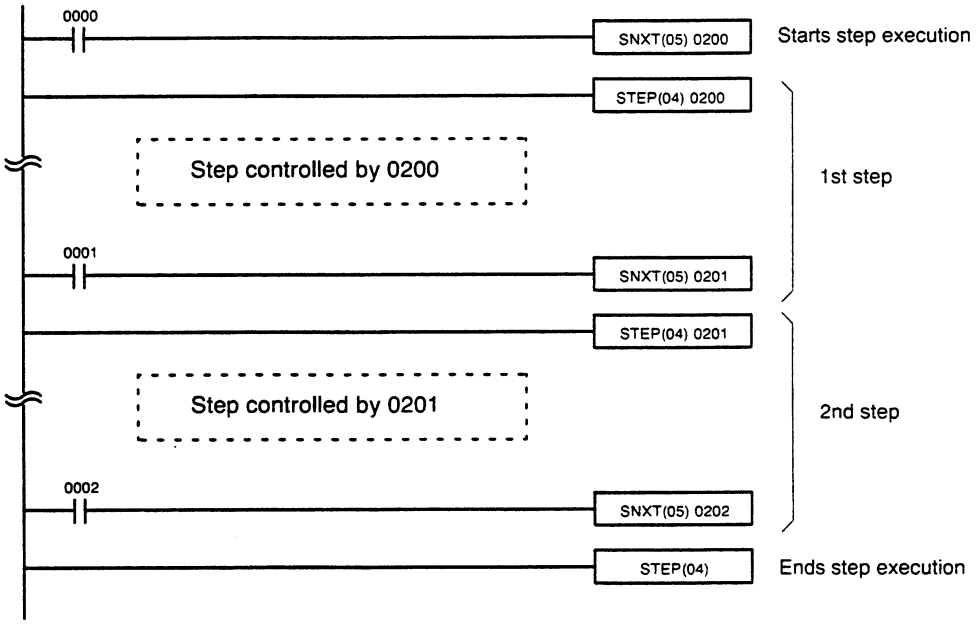
Description

STEP(04) uses a control bit that is either an output bit, a work bit, or a bit in the LR or DR area to define the beginning of a section of the program called a step. STEP(04) does not require an execution condition, i.e., its execution is controlled through the control bit. To start execution of the step, SNXT(05)

is used with the same control bit as used for STEP(04). If SNXT(05) is executed with an ON execution condition, the step with the same control bit is executed. If the execution condition is OFF, the step is not executed. The SNXT(05) instruction must be written into the program so that it is executed before the program reaches the step it starts. It can be used at different locations before the step to control the step according to two different execution conditions. Any step in the program that has not been started with SNXT(05) will not be executed.

Once SNXT(05) is used in the program, step execution will continue until STEP(04) is executed without a control bit. STEP(04) without a control bit must be preceded by SNXT(05) with a dummy control bit.

Execution of a step is completed either by execution of the next SNXT(05) or by turning OFF the control bit for the step. When the step is completed, all of operand bits in the step are turned OFF and all timers in the step are reset to their SVs. Counters, shift registers, and bits used in KEEP(12) maintain status. Two simple steps are shown below.



Address	Instruction	Operands
000	LD	0000
001	SNXT(05)	0200
002	STEP(04)	0200
	Step controlled by 0200.	
030	LD	0001
031	SNXT(05)	0201

Address	Instruction	Operands
032	STEP(04)	0201
	Step controlled by 0201.	
051	LD	0002
052	SNXT(05)	0202
053	STEP(04)	---

Steps can be programmed in consecutively. Each step must start with STEP(04) and generally ends with SNXT(05). When steps are programmed in series, three types of execution are possible: sequential, branching, or parallel. The execution conditions for, and the positioning of, SNXT(05) determine how the steps are executed.

Precautions

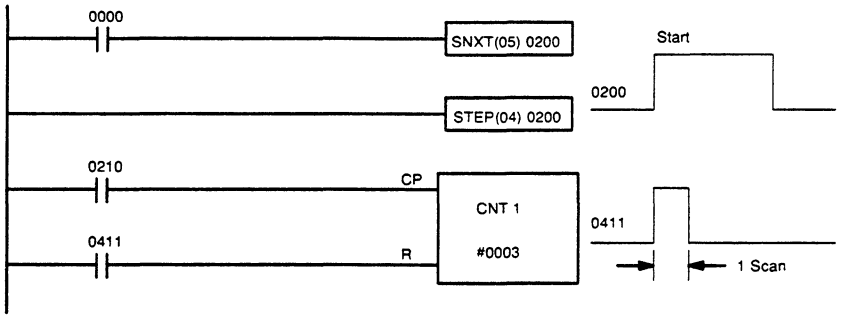
Interlocks and END(01) cannot be used within step programs.

Bits used as control bits must not be used anywhere else in the program unless they are being used to control the operation of the step.

If output bits, work bits, or LR bits are used for control bits, their status will be lost during any power interruption. If it is necessary to maintain status to resume execution at the same step, DR bits must be used.

Flags

0411: Step Start Flag; turns ON for one scan when STEP(04) is executed and can be used to reset counters in steps as shown below if necessary.



Address	Instruction	Operands
000	LD	0000
001	SNXT(05)	0200
002	STEP(04)	0200
003	LD	0210

Address	Instruction	Operands
004	LD	0411
005	CNT	1
		# 0003

3–8 Debugging

After inputting a program and correcting it for syntax errors, it must be executed and all execution errors must be eliminated. Execution errors include an excessively long scan and inappropriate control actions, i.e., the program not doing what it is designed to do.

If desired, the program can first be executed isolated from the actual control system and wired to dummy inputs and outputs to check for certain types of errors before actual trial operation with the controlled system.

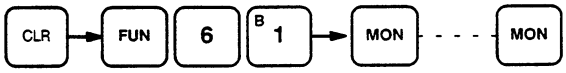
3–8–1 Displaying and Clearing Error Messages

When an error occurs during program execution, it can be displayed for identification by pressing CLR, FUN, 6, 1 and then MON. If an error message is displayed, MON can be pressed to access any other error messages that are stored by the system in memory. If MON is pressed in PRGM mode, the error message will be cleared from memory; be sure to write down the error message when required before pressing MON. CHECK OK will be displayed when the last message has been cleared.

In RUN mode errors cannot be cleared by pressing MON. Also, if the cause of the error still exists, it must be eliminated before the error message can be cleared. Refer to *Section 5 Troubleshooting* for all details on all error messages. The sequence in which error messages are displayed depends on the priority of the errors.

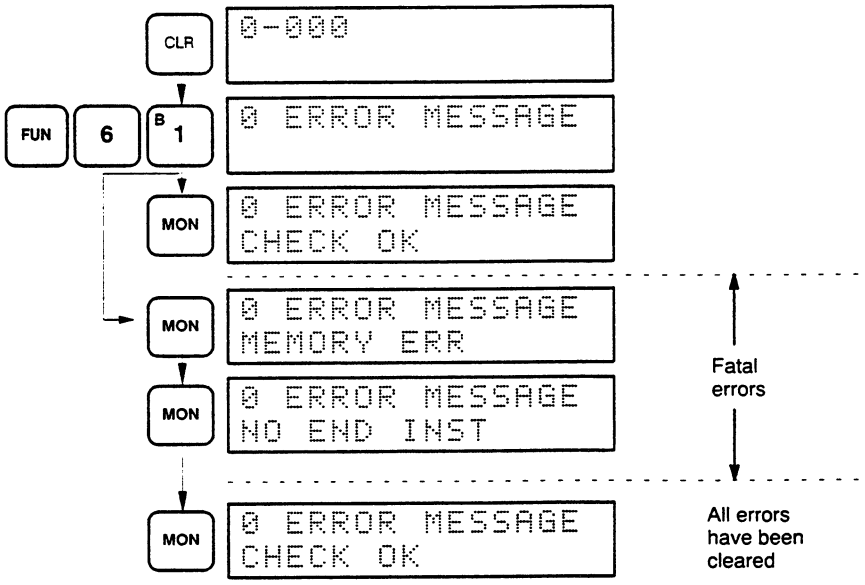
Although error messages can be displayed in any mode, they can be cleared only in PROGRAM mode. There is no way to restart the PC following a fatal error without first clearing the error message in PROGRAM mode.

Key Sequence



Example

The following displays show some of the messages that may appear. Refer to *Section 5 Troubleshooting* for an extensive list of error messages, their meanings, and the appropriate responses.

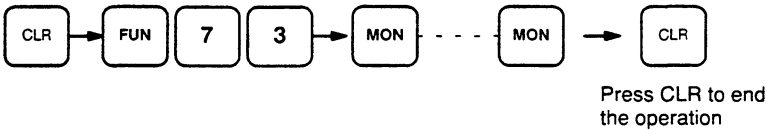


3–8–2 Reading the Scan Time

The following operation can be used to read the present scan time and the maximum scan time. The Monitor Key can be pressed consecutively to repeat the operation. The PC must be in RUN mode.

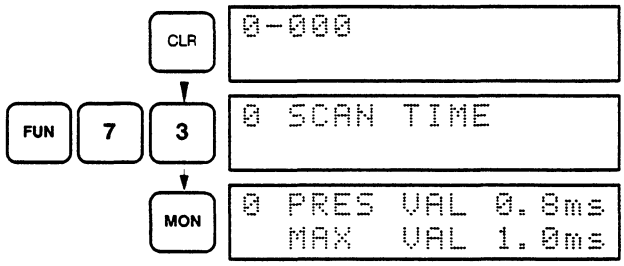
This operation is supported only by the SP16 and SP20.

Key Sequence



Example

The following displays show the scan time displays.



3–9 Program Execution

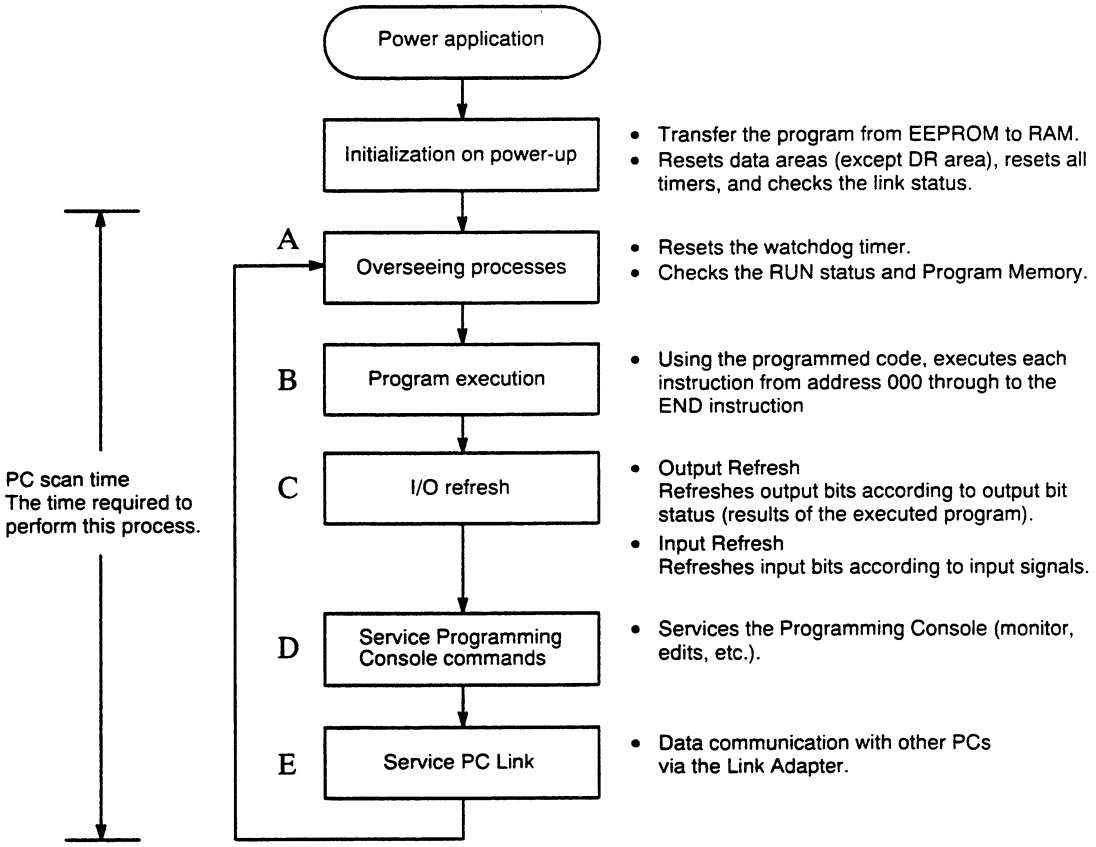
The timing of various operations must be considered both when writing and debugging a program. The time required to execute the program and perform other CPU operations is important, as is the timing of each signal coming into and leaving the PC in order to achieve the desired control action at the right time. This section explains the scan and shows how to calculate the scan time and I/O response times.

3–9–1 Scan

The major factors in determining program timing are the scan time and the I/O response time. When program execution is started, the CPU scans the program from top to bottom, checking all conditions and executing all instructions accordingly as it moves down the bus bar. It is important that instructions be placed in the proper order so that, for example, the desired data is moved to a word before that word is used as the operand for an instruction. Remember that an instruction line is completed to the terminal instruction at the right before executing an instruction lines branching from the first instruction line to other terminal instructions at the right.

One cycle of CPU operation is called a scan; the time required for one cycle is called the scan time. The time required to produce a control output signal following reception of an input signal is called the I/O response time.

The overall flow of CPU operation is as shown in the following flowchart:



The first initialization process is performed only once, immediately after power is applied to the PC. The remaining operations are performed in cyclic fashion, with each cycle forming one scan. The scan time is the time that is required for the CPU to complete one of these cycles. This cycle includes basically five types of operation.

1. Overseeing
2. Program execution
3. I/O refresh
4. Programming Console servicing
5. PC Link servicing

The scan time is the total time required for the PC to perform all of the above operations. The present and maximum scan time can be read out from the Programming Console with the SP16 or SP20. Refer to page 109 for details.

All peripheral devices are serviced once each scan in the order given above.

Watchdog Timer and Long Scan Times

Within the PC, the watchdog timer measures the scan time and compares it to a set value. If the scan time exceeds the set value of the watchdog timer, 100 ms, a CPU error is generated and the CPU stops. One scan time is approximately 300 ms plus the time required for program execution.

3–10 I/O Response Time

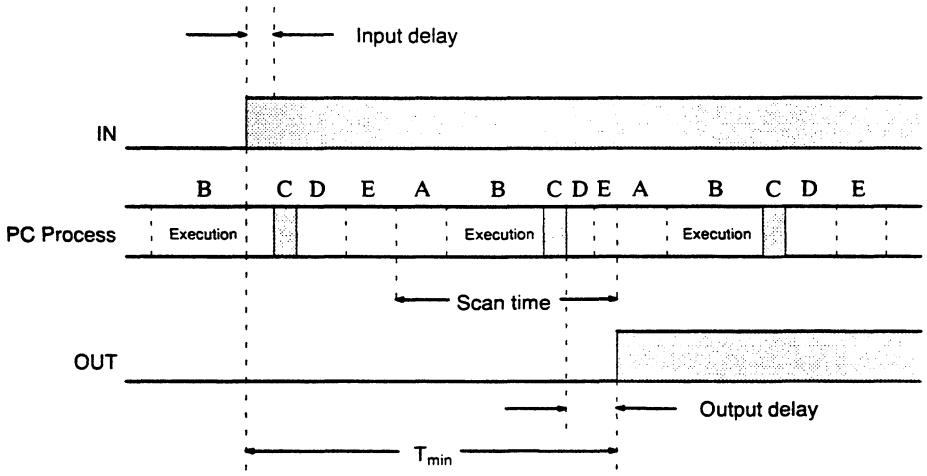
The I/O response time is the time it takes for the PC to output a control signal after it has received an input signal. The time it takes to respond depends on the scan time and when the CPU receives the input signal relative to the I/O refresh period.

3–10–1 Single PCs

Both input and output refreshes are performed at the same time in the CPU cycle, after the program process has been completed. The following section show how the maximum and minimum I/O response times may be calculated.

Minimum I/O Response Time

The PC responds most quickly when it receives an input signal just prior to the I/O refresh period in the scan. Once the input bit corresponding to the signal has been turned ON, the program will have to be executed once to turn ON the output bit for the desired output signal. The I/O response time in this case is thus found by adding the input delay, the scan time, and the output delay. This situation is illustrated below.



T_{min} = Minimum I/O response time
= input delay + filter time + scan time + output delay
= C + B + (300 μs + program execution time) + A

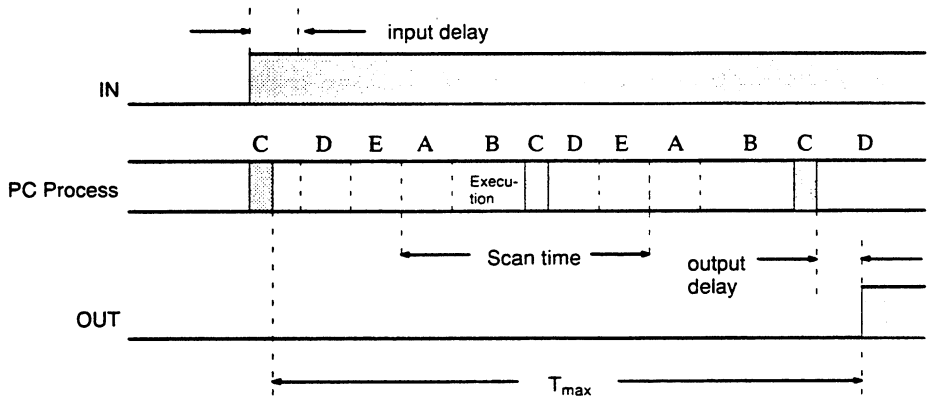
where A: Output delay (see table below)
B: Filter value (refer to *Section 2 Installation*)
C: Input delay (see table below)
and Program execution time = sum of instruction execution times (refer to *Appendix C Programming Instructions and Execution Times)*

The ON input delay is 300 μs maximum and the OFF input delay is 250 μs maximum. Output delays are given in the following table.

Output	Relay	Transistor
ON output delay	10 ms max.	20 μs max.
OFF output delay	10 ms max.	300 μs typical

Maximum I/O Response Time

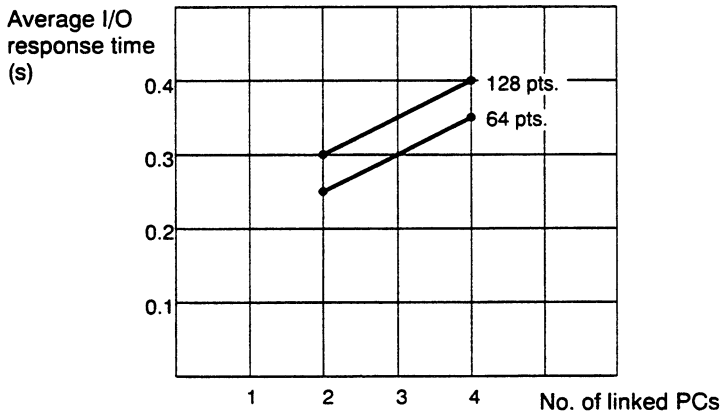
The PC takes longest to respond when it receives the input signal just after the I/O refresh phase of the scan. In this case the CPU does not recognize the input signal until the end of the next scan. The maximum response time is thus one scan longer than the minimum I/O response time.



T_{max} = Maximum I/O response time
= input delay + filter time + (scan time x 2) + output delay
= C + (B + 0.5 ms) + ((300 μs + program execution time) x 2) + A

3–10–2 Multiple PCs

If more than one PC is linked via a Link Adapter in a distributed control system, the I/O response time is increased with every PC having their own response time. The average I/O response time given as a function of the number of linked PCs is shown in the following graph. For a system of four PCs, the maximum I/O response time is 0.4 s.



Note The duration required to process linked PCs is increased if a PC already linked is disconnected from the network.

SECTION 4
Operation

This section describes how to monitor and maintain PC operation once a program has been input and transferred. It also provides the procedure for initializing memory cards. Refer to *3–5–8 Program Transfer* for the procedures for transferring programs and data between the Programming Console and the PC or Memory Cards.

4–1 Monitoring Operation and Modifying Data

The simplest form of operation monitoring is to display the address whose operand bit status is to be monitored using the Program Read or the search operation. As long as the operation is performed in RUN mode, the status of any bit displayed will be indicated.

This section provides other procedures for monitoring data as well as procedures for modifying data that already exists in a data area. Data that can be modified includes the PV (present value) for any timer or counter.

All monitor operations in this section can be performed in RUN or PROGRAM mode and can be cancelled by pressing CLR.

All data modification operations are performed after first performing one of the monitor operations. Data modification is possible in either PROGRAM or RUN mode.

4–1–1 Bit/Multibit Monitor

The status of any bit or word in any data area can be monitored using the following operation. Although the operation is possible in any mode, ON/OFF status displays will be provided for bits in RUN mode only.

The Bit/Multibit Monitor operation can be entered either from a cleared display by designating the first bit or word to be monitored or it can be entered from any address in the program by displaying the bit or word address whose status is to be monitored and pressing MON.

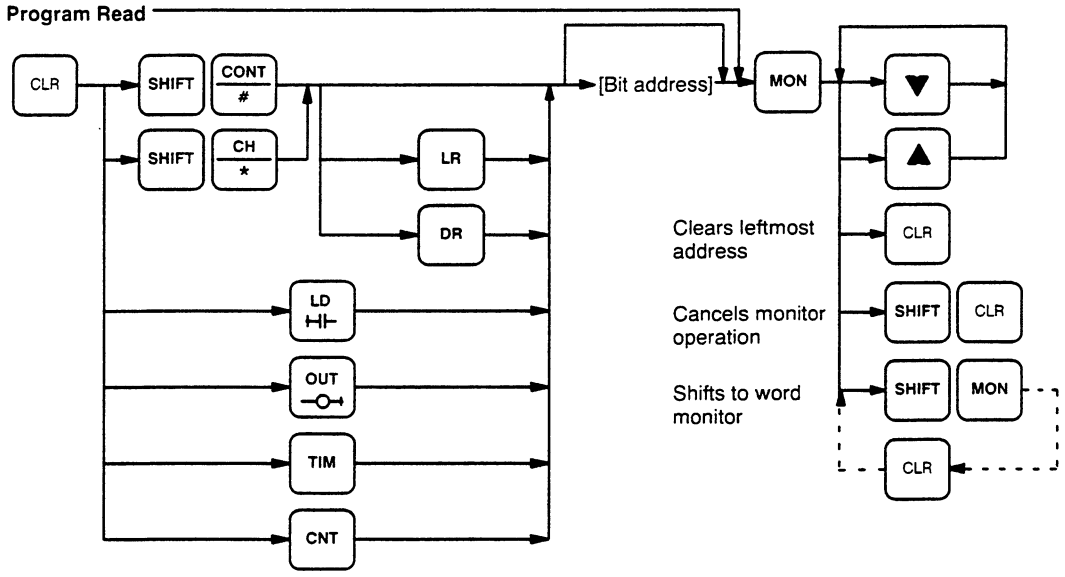
When a bit is monitored, it's ON/OFF status will be displayed (in RUN mode); when a word address is designated other than a timer or counter, the digit contents of the word will be displayed; and when a timer or counter number is designated, the PV of the timer will be displayed and a small box will appear if the completion flag of a timer or counter is ON. When multiple words are monitored, a space will appear between the different address designations. The status of the arithmetic flags are cleared when END(01) is executed and cannot be monitored.

Up to three memory addresses, either bits, words, or a combination of both, can be monitored at once. To continue designating addresses with the second of the following key sequences.

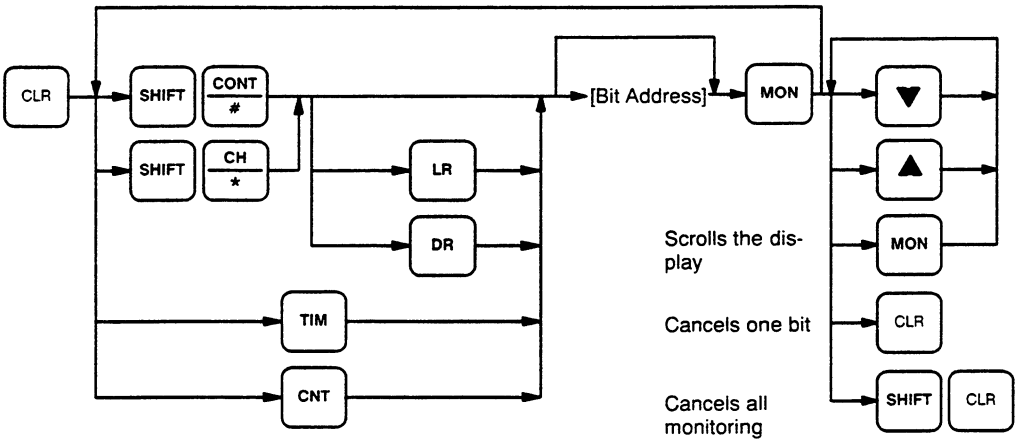
During a monitor operation the up and down keys can be pressed to increment and decrement the leftmost address on the display and CLR can be pressed to cancel monitoring the leftmost address on the display. If the last address is cancelled, the monitor operation will be cancelled. The monitor operation can also be cancelled regardless of the number of addresses being monitored by pressing SHIFT and then CLR.

LD and OUT can be used only to designate the first address to be displayed; they cannot be used when an address is already being monitored.

Bit/Word Monitor Key Sequence

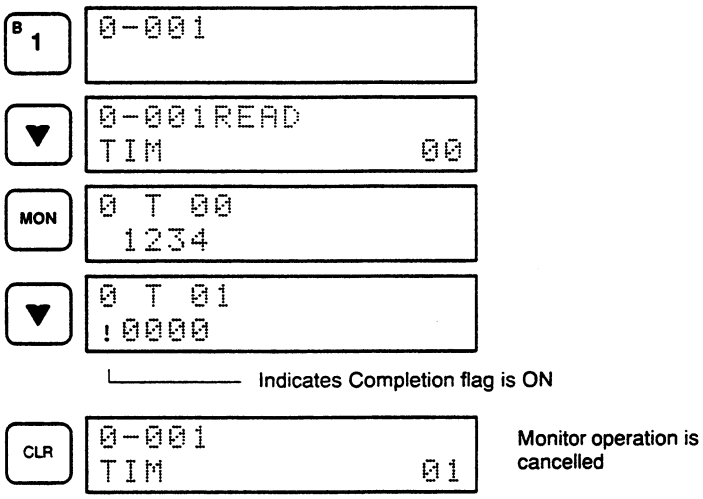


Multibit/Word Monitor Key Sequence

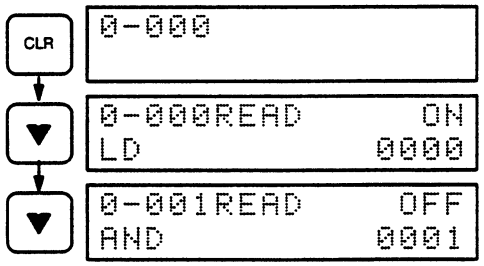


Examples The following examples show various applications of this monitor operation.

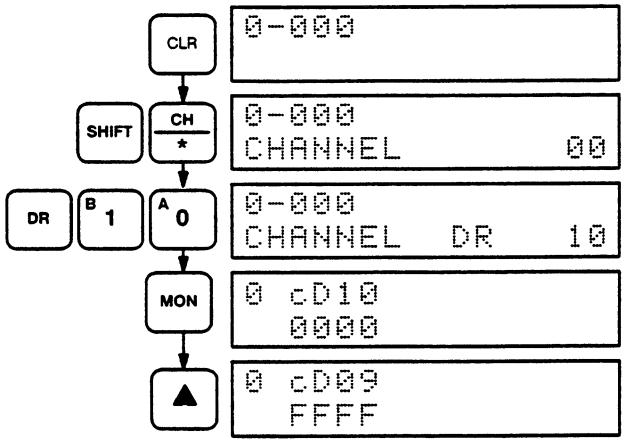
Program Read then Monitor



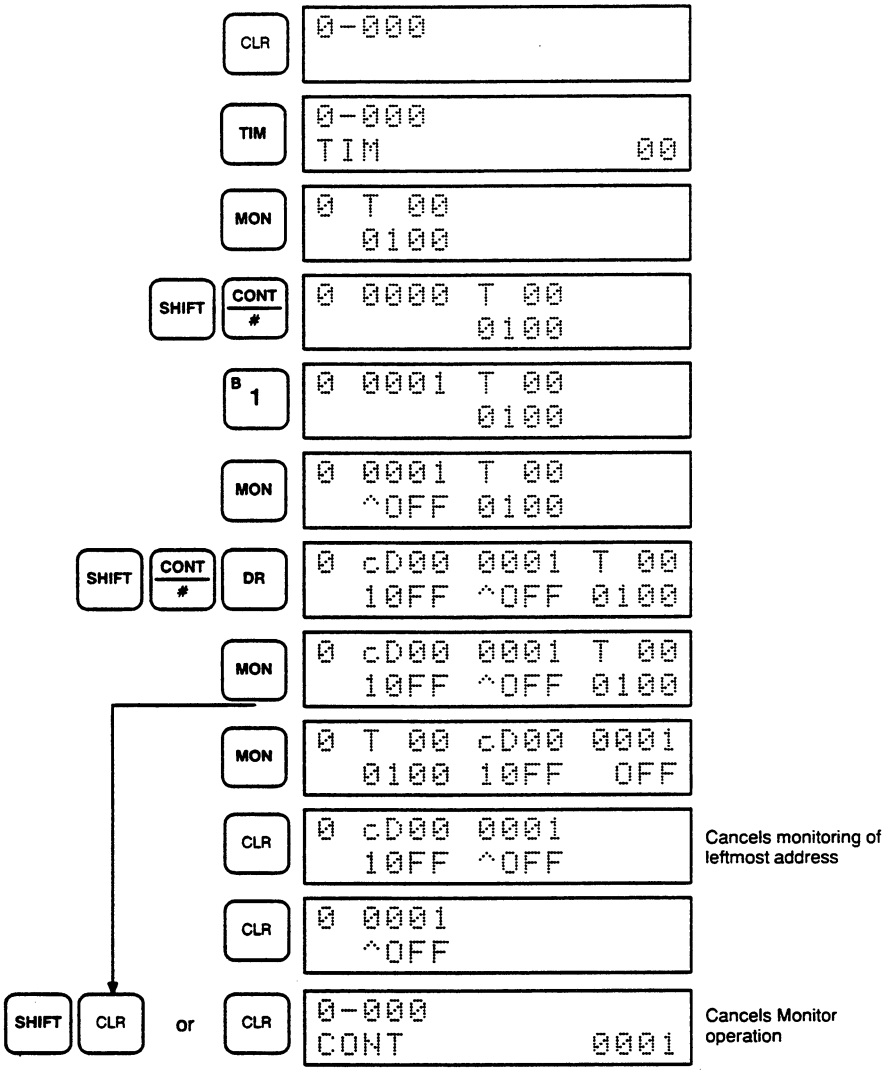
Bit Monitor



Word Monitor



Multi-address Monitoring



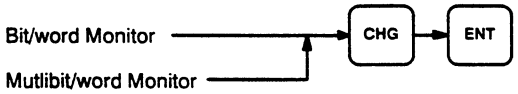
4–1–2 Force Set/Reset

When the Bit/Multibit Monitor operation is being performed and a bit, timer, or counter address is leftmost on the display, CHG and ENT can be pressed to turn ON/OFF the bit, start/reset the timer, or increment/reset the counter. Timers will not operate in PROGRAM mode. Dedicated flags and bits cannot be turned ON and OFF with this operation.

Bit status will remain ON or OFF until the I/O bit status is refreshed, which occurs each scan. Hence, forced status will be canceled at each I/O refresh. If a timer is started, the Completion Flag for it will be turned ON when SV has been reached.

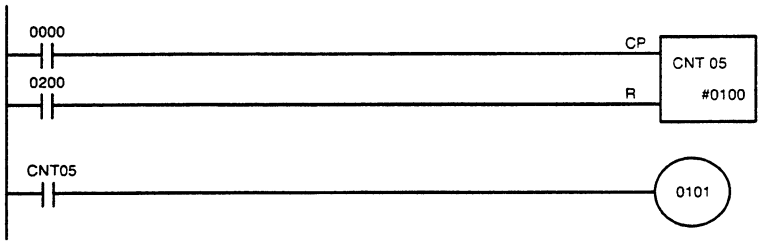
This operation can be used in RUN mode to check wiring of outputs from the PC prior to actual program execution.

Key Sequence



Example

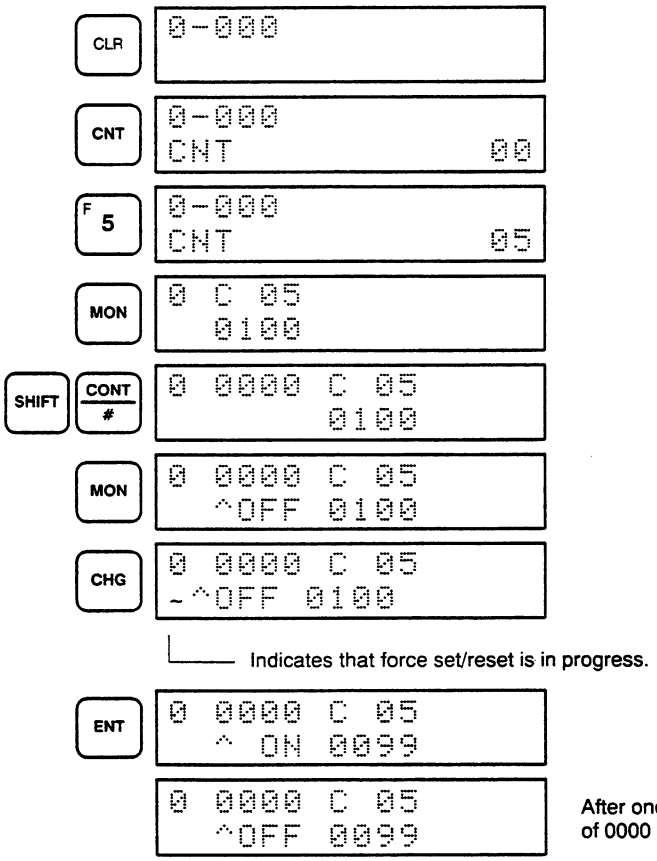
The following example shows how either bits or counters can be controlled with the Force Set/Reset operation. The displays shown below are for the following program section.



Address	Instruction	Operands
000	LD	0000
001	LD	0200
002	CNT	05
		# 0100
003	LD	CNT 05
004	OUT	0101

The following displays show what happens when CNT 05 is set when bit 0000 is ON.

(This example is in RUN mode.)



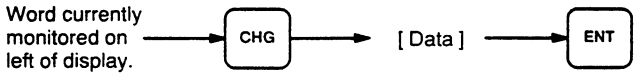
4–1–3 Hexadecimal/BCD Data Modification

When the Bit/Multibit Monitor operation is being performed and a BCD or hexadecimal value is leftmost on the display, CHG can be input to change the value. Dedicated words cannot be changed.

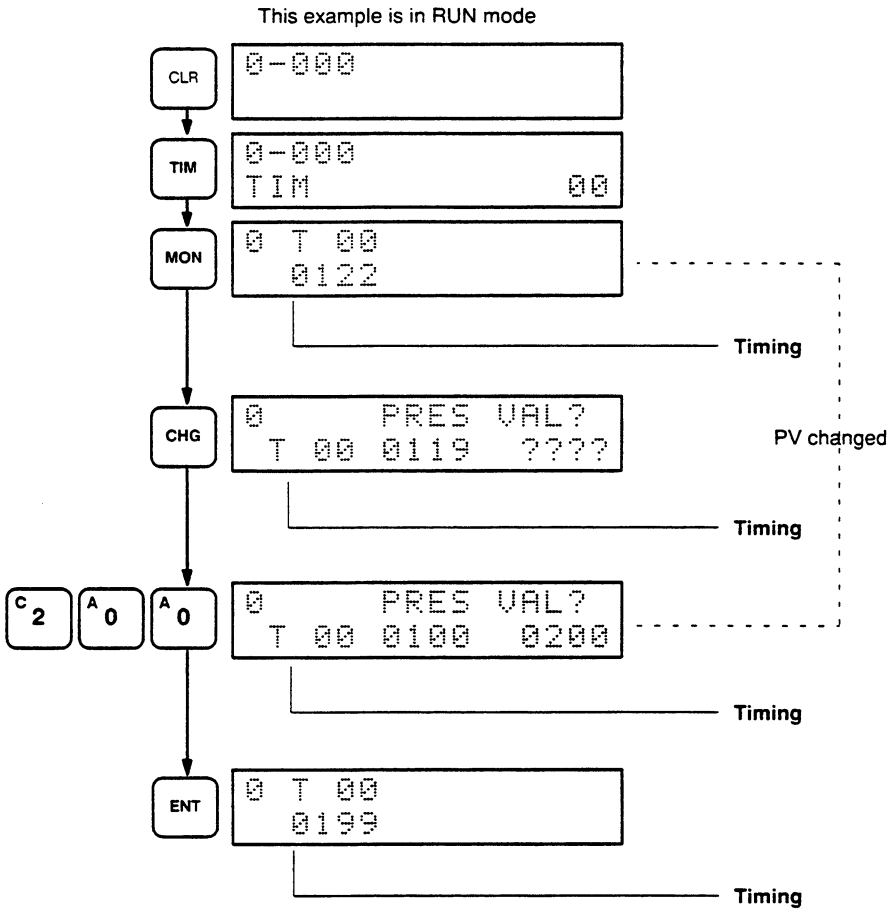
If a timer or counter is leftmost on the display, the PV will be displayed and will be the value changed. PV can be changed in RUN mode only when the timer or counter is operating.

To change contents of the leftmost word address, press CHG, input the desired value, and press ENT.

Key Sequence



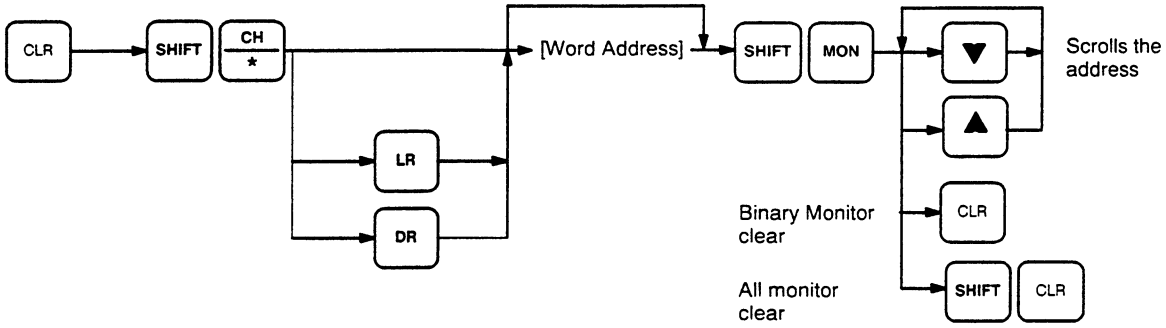
Example The following example shows the effects of changing the PV of a timer.



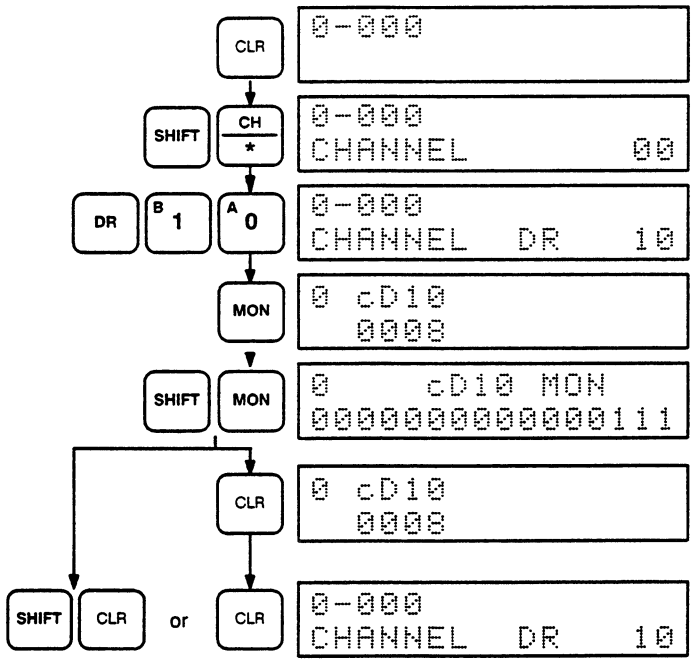
4–1–4 Binary Monitor

You can specify that the contents of a monitored word be displayed in binary by pressing SHIFT and MON after the word address has been input. Words can be successively monitored by using the up and down keys to increment and decrement the displayed word address. To clear the binary display, press CLR.

Key Sequence



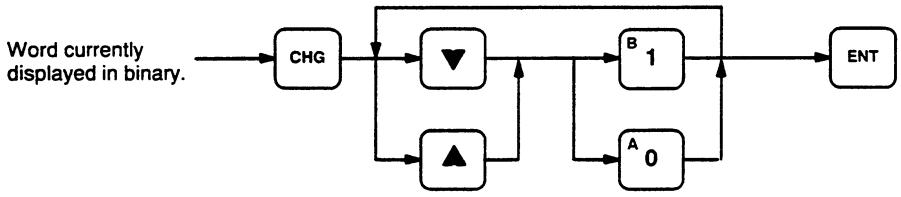
Example



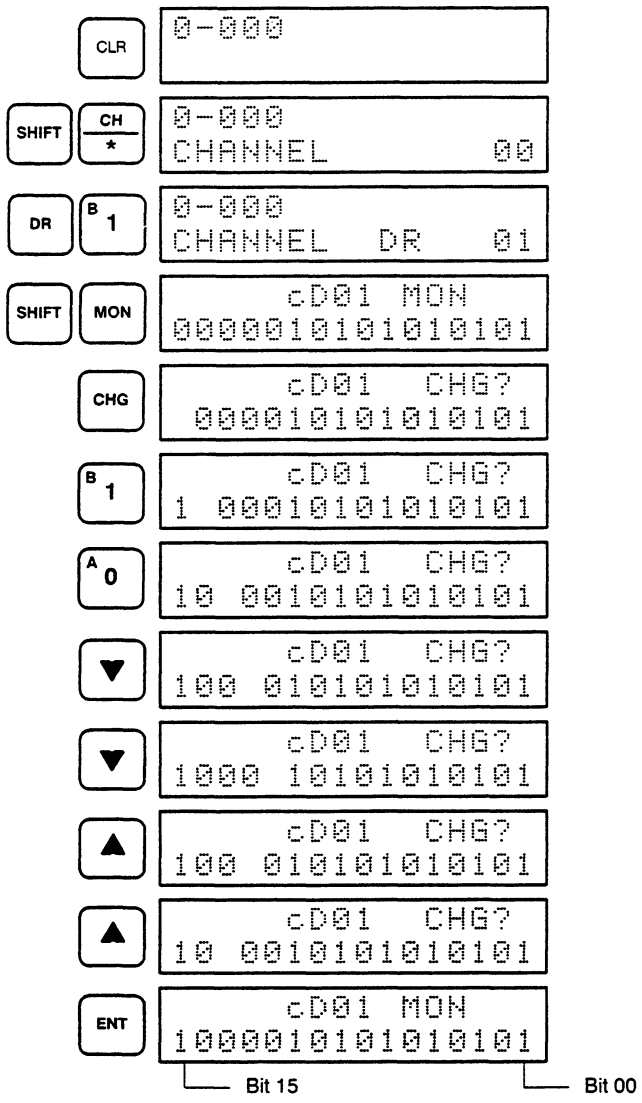
4–1–5 Binary Data Modification

This operation can be used to change individual binary bit status. The cursor, which can be shifted to the left with the up key and to the right with the down key, indicates the position of the bit that can be changed. After positioning at the desired bit, the 0 or 1 Key can be entered to specify the bit value. After a bit value has been changed, the blinking square will appear at the next position to the right of the changed bit.

Key Sequence



Example



4–2 Memory Card Initialization

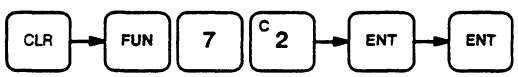
The Programming Console provides a slot for a Memory card to allow the backup of programs. Only one model of Memory Card, HMC-ES141, may be used. Each Memory Card has 16 Kbytes of S-RAM. A battery is built-in to the Memory Card to allow the data to be retained. One Memory Card can hold up to 27 SP10 programs or 18 SP16 or SP20 programs.

A program can be saved to a full memory card and read out, but an error message ("ERR CARD FULL") will be displayed when the corrected file is saved again.

Initialization Procedure

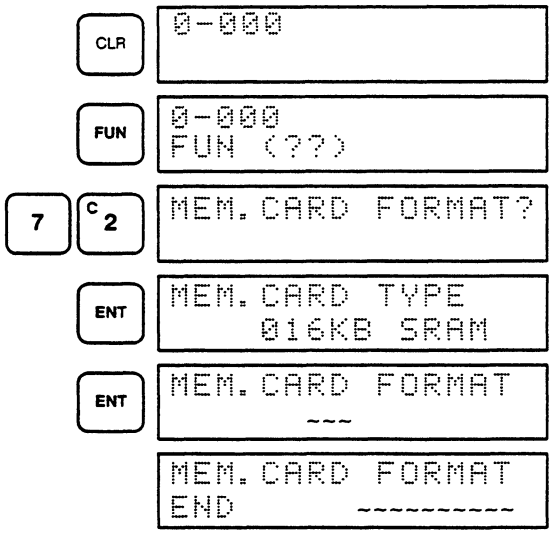
After inserting a new Memory Card into the Programming Console, the Card must be initialized using the following key sequence.

Key Sequence



The corresponding displays are shown following.

Pressing ENT for the first time, the Programming Console displays the specifications of the Memory Card. By pressing ENT again the Programming Console commences formatting the Memory Card. While the card is being formatted, cursors on the display indicate the progress of the format operation. When END is displayed, formatting is complete.



Note
1. The battery of the memory card (model HMC-BAT01 lithium battery CR2325 3 V) has to be replaced within the time period indicated on the back of the memory card. If the battery is not replaced by this date, the program or the data in the card will be lost. When replacing the battery, the new battery must be installed within one minute or data in the card will be lost.
2. While the memory card is being accessed, the M/C ON LED on the Programming Console will be lit. If the memory card is pulled out from the Programming Console while the LED is ON, data on the card will be damaged.

Error Messages

A number of errors relating to the Memory Card may occur during the program check. If one of the following errors is shown on the display, program transfer cannot proceed.

Error Message	Meaning/Correction
NO END INST	END instruction cannot be found. Include an END instruction at the end of the program.
(Program address) or ????	Displayed address has the inappropriate operand or inappropriate instruction. Correct the program and transfer it to the PC again.
ERR CARD → ProCo	The program stored in the memory card contains errors. The memory card data is transferred to the Programming Console's RAM to allow it to be checked. Perform a program check to confirm the program.
NO SUPPORT CARD	The Memory Card is not initialized, or is not supported by the Programming Console (SP10-PRO01-V1). Initialize the Memory Card (or check the specification of the memory card).
ERR CARD FULL	The memory card is full. Delete unneeded files or use another initialized Memory Card.
PTCT ON OR EPROM	The Memory Card protect switch is ON, or the inserted memory card is an EPROM memory card. Adjust the protect switch, or use a RAM Memory Card.
NO MEM. CARD	Memory Card is not inserted properly in the card slot.

SECTION 5
Troubleshooting

5–1 Alarm Indicators

There are three indicators on the front of the CPU that provide visual indication of errors in the PC. The power indicator (POWER) indicates errors due to incorrect application of power to the PC; the error indicator (ERROR) indicates fatal errors (i.e., ones that will stop PC operation); the link indicator (LINK) indicates PC link errors.

Caution The PC will turn ON the error indicator (ERROR), stop program execution, and turn OFF all outputs from the PC for most hardware errors and for fatal software errors. PC operation will continue for all other errors. It is the user's responsibility to take adequate measures to ensure that a hazardous situation will not result from automatic system shutdown for fatal errors and to ensure that proper actions are taken for errors for which the system is not automatically shut down. System flags can be used to program proper actions.

5–2 Reading and Clearing Errors and Messages

System error messages can be displayed on the Programming Console.

On the Programming Console, press the CLR, FUN, 6, 1, and MON keys. If there are multiple error messages stored by the system, the MON key can be pressed again to access the next message. If the system is in PROGRAM mode, pressing the MON key will clear the error message, so be sure to write down all message errors as you read them.

It is not possible to clear an error or a message while in RUN mode; the PC must be in PROGRAM mode.

When all messages have been cleared, "CHECK OK" will be displayed and the ERROR LED will turn OFF.

If a "COMM ERR" occurs, the Programming Console will ignore any key operation other than CLR, so press CLR to clear the communication error and allow access via the PC Link to CPU Unit #0. Ensure the correct connection of cables between the Units.

Note If a memory error occurs in the PC, the Programming Console may not function properly if the program is transferred from the PC to the Programming Console. Perform a program comparison between the transferred program and the program in the PC to see if the programs are the same.

5–3 Error Messages

There are basically two types of errors for which messages are displayed: programming errors and fatal operating errors.

The type of error can be quickly determined from the indicators on the CPU, as described below for the two types of errors. If the status of an indicator is not mentioned in the description, it makes no difference whether it is lit or not.

After eliminating the cause of an error, clear the error message from memory before resuming operation.

Programming Errors

The following error messages appear if an error occurs during programming. The Programming Console will be in PROGRAM mode. The POWER indicator will be lit and the RUN indicator will not be lit for either of these.

Error Messages

The following error messages may appear when inputting a program. Correct the error as indicated and continue with the input operation.

Error Message	Error Type	Possible Cause/Correction
PRGM OVER	Program too large	Program size exceeds the capacity. (The last address is not a NOP instruction, so the program cannot be written.) Clear any data after the END instruction or shorten the program.
ADR OVER	Address too large	Program exceeds program memory's last address. Set the address again.
I/O No. ERR	Operand error	An illegal value has been entered for an operand. Reconfirm the allowable operand area for each instruction, and correct the data.

Operating Errors

The following error messages appear for errors that occur after program execution has been started. PC operation and program execution will halt when any of the following errors occur. The status of the indicators is given for each error.

Error Type	Error Message	LED Indicator			Possible Cause/Correction
		POWER	RUN	ERROR	
Power failure	-	●	●	●	Check the power supply, voltage, and wiring.
CPU error	-	☼	●	☼	The watchdog timer (100 ms) has timed out. Turn power OFF, change to PROGRAM mode, and turn power ON again.
Memory error	MEMORY ERR	☼	●	☼	The program may contain an error. Correct the program, then transfer the corrected program to the PC from the Programming Console. Alternatively, turn the PC power OFF and ON. If the memory error occurs when power is switched ON, an error may have occurred when the program was transferred from EEPROM to RAM.
No END instruction	NO END INST	☼	●	☼	The END instruction cannot be found in the program. Change to PROGRAM mode, and add an END instruction to the end of the program.

☼ Lit ● Not lit

Note When a communication error occurs during a PC Link operation, the LINK LED will go OFF.

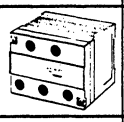
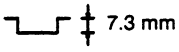
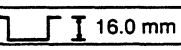
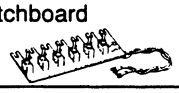
Other Error Messages

A number of other error messages are detailed within this manual. Errors in program input can be found in *3–5 Inputting the Program* and errors in program transfer are detailed in *3–5–8 Program Transfers*.

5–4 Error Flags

A number of flags are available in the dedicated bit area that can be used for troubleshooting. Details are provided in *3–2–5 Dedicated Bits*.

Appendix A
Standard Models

Name	Specifications				Model number
SP10 PC	6 inputs (DC; one common) 4 outputs (2 commons, 2 pts each)	100 to 240 VAC	Relay outputs		SP10-DR-A
			Transistor outputs		SP10-DT-A
		24 VDC	Relay outputs		SP10-DR-D
			Transistor outputs		SP10-DT-D
SP16 PC	10 inputs (DC; one common, 2 pts; one common, 8 pts) 6 outputs (3 commons, 2 pts each)	100 to 240 VAC	Relay outputs		SP16-DR-A
			Transistor outputs		SP16-DT-A
		24 VDC	Relay outputs		SP16-DR-D
			Transistor outputs		SP16-DT-D
SP20 PC	12 inputs (DC; 2 commons, 2 pts each; one common, 8 pts) 8 outputs (4 commons, 2 pts each)	100 to 240 VAC	Relay outputs		SP20-DR-A
			Transistor outputs		SP20-DT-A
		24 VDC	Relay outputs		SP20-DR-D
			Transistor outputs		SP20-DT-D
Programming Console 	Vertical, hand-held with backlit LCD display. Compatible with Memory Cards. Memory Card and Connecting Cable sold separately (see below). Usable only with SP-series PCs.				SP10-PRO01-V1
Programming Console Connecting Cable 	Connect Programming Console to CPU or to Link Adapter. Same as 2-m and 4-m Link Adapter Connecting Cables.*		2-m cable		SP10-CN221
			4-m cable		SP10-CN421
Link Adapter 	Used to link up to 4 SP-series CPUs. Cables sold separately (see below).				SP10-AL001
Link Adapter Connecting Cable 	Connect the CPUs and Link Adapter.*		20-cm cable	Do not connect to Programming Console.	SP10-CN211
			1-m cable		SP10-CN121
			2-m cable		SP10-CN221
			4-m cable		SP10-CN421
Memory Card 	16-Kbyte SRAM cards (battery built in). Battery will last 5 years from when it is mounted to the Memory Card.				HMC-ES141
Mounting Accessories 	50-cm DIN Track		Depth  7.3 mm		PFP-50N
	1-m DIN Track				PFP-100N
	1-m DIN Track		Depth  16.0 mm		PFP-100N2
	End Plate				PFP-M
	Spacer				PFP-S
Simulation Switchboard 	Board with 6 switches and AC power cord (power switch also included)				SP10-ETL01 SP16-ETL01 SP20-ETL01

***Note:** The cables between CPUs and the Link Adapter must be 4 m or less. The sum of the distance between CPU #0 and the Link Adapter and the distance between the Link Adapter and the Programming Console must be 4.2 m or less.

Appendix B
Specifications

General Ratings

Item		SP__ -D_ -A	SP__ -D_ -D*
Power supply voltage		100 to 240 VAC, 50/60 Hz	24 VDC
Operating voltage range		85 to 264 VAC	20.4 to 26.4 VAC
Power consumption		30 VA max.	10 W max.
24-VDC output terminal	SP10	0.1 A max. at 24 VDC +10%	None
	SP16, SP20	0.2 A max. at 24 VDC +10%	
Insulation resistance		20 MΩ (at 500 VDC) between current-carrying and noncurrent-carrying metal parts	
Dielectric strength		2,300 VAC, 50/60 Hz for 1 min between current-carrying and noncurrent-carrying metal parts	
Noise immunity		1,000 V_{p-p} with 100-ns to 1-μs pulse width and 1-ns pulse rise	
Vibration resistance		10 to 58 Hz with 0.15-mm double amplitude or 58 to 150 Hz (1G) for 80 min in X, Y, Z directions (JIS C0911)	
Shock resistance		Destruction: 15G three times in X, Y, Z directions (JIS C0912)	
Ambient operating temperature		0° to 55°C (Programming Console: 0° to 45°C)	
Ambient operating humidity		10% to 90% (with no condensation)	
Ambient atmosphere		No corrosive gases	
Ambient storage temperature		–20°to 75°C (Programming Console: –20°to 65°C)	
Structure		Control panel mountable (IP30)	
Weight	SP10	500 g max.	
	SP16	600 g max.	
	SP20	700 g max.	
Dimensions	SP10	92 x 68 x 81 (W x H x D) without cables	
	SP16	135 x 68 x 81 (W x H x D) without cables	
	SP20	160 x 68 x 81 (W x H x D) without cables	

***Note:** Do not use normally closed contacts for inputs to models that run on DC power. Doing so can cause counters and shift registers to be reset and bits programmed with KEEP(12) to reverse status during power interruptions.

Input Specifications

Input voltage		24 VDC $^{+10\%}/_{-15\%}$
Input impedance		3.3 kΩ
Input current		7 mA typ. (at 24 VDC)
Input frequency		Standard counters: 1 kHz max. High-speed counters (SP16, SP20): 3.3 kHz max.
ON voltage		15 VDC min.
OFF voltage		5 VDC max.
ON/OFF delays		ON: 200 μs max. OFF: 250 μs max.
No. of inputs	SP10	6 points (1 circuit)*
	SP16	10 points (1 circuit, 8 pts; 1 circuit, 2 pts)*
	SP20	12 points (1 circuit, 8 pts; 2 circuits, 2 pts each)*

*The common terminal can be connected either as + or –.

Output Specifications

Item		Relays	Transistors
Switching capacity		Resistive loads: 2 A, 250 VAC (cosϕ=1); 2 A, 24 VDC; 4 A/common Inductive loads: 0.5 A, 250 VAC (cosϕ=0.4)	0.3 A, 24 VDC $^{+10\%}/_{-15\%}$
ON/OFF delays		ON: 10 μs max. OFF: 10 μs max.	ON: 20 μs max. OFF: 300 μs typical
Minimum permissible load		100 mA, 5 VDC	---
Leakage current		---	0.1 mA max.
Residual voltage		---	1.0 V max
No. of outputs	**SP10**	4 pts. (2 circuits, 2 pts each)	
	SP16	6 pts. (3 circuits, 2 pts each)	
	SP20	8 pts. (4 circuits, 2 pts each)	
Relay life		Electrical: 100,000 operations min. Mechanical: 20,000,000 operations min.	---

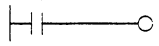
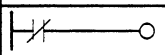
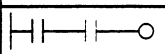
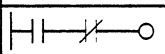
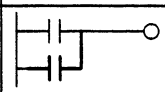
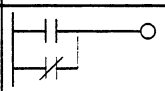
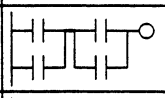
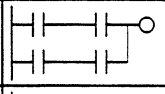
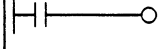
CPU Characteristics

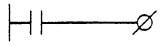
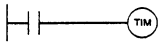
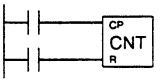
Control method		Stored program
I/O control		Cyclic scan
Program		Ladder diagram
Instruction length		1 step/instruction; 1 to 5 words/instruction
No. of instructions	**SP10**	34: 12 basic, 5 arithmetic, 17 special
	SP16, SP20	38: 12 basic, 5 arithmetic, 21 special
Processing speed		0.2 μs min./instruction; 0.72 μs min. average for reading/processing I/O status from memory
Program capacity	**SP10**	144 words (approximately 100 instructions)
	SP16, SP20	348 words (approximately 200 instructions)
I/O bits	**SP10**	10 (bit 0000 to bit 0005 and bit 0100 to bit 0103)
	SP16	16 (bit 0000 to bit 0009 and bit 0100 to bit 0105)
	SP20	20 (bit 0000 to bit 0011 and bit 0100 to bit 0107)
Work bits	**SP10**	36 (bit 0008 to bit 0015 and bit 0104 to bit 0215)
	SP16	208 (bit 0010 to bit 0015, bit 0106 to bit 0215, and bit 1000 to bit 2015)
	SP20	204 (bit 0012 to bit 0015, bit 0108 to bit 0215, and bit 1000 to bit 2015)
Dedicated bits	**SP10**	20 (bit 0300 to bit 0315 and bit 0408 to bit 0411)
	SP16, SP20	69 (bit 0300 to bit 0315, bit 0408 to bit 0411, bit 0515, bit 0700 to bit 0715, bit 0800 to bit 0815, and bit 0900 to bit 0915)
Data-holding bits/link bits		256 data-holding bits of which 0, 64, or 128 can be designated as link bits
Timers/counters	**SP10**	16 total: one 1-ms timer and one analog timer (0.1 to 25.0 s) plus 10-ms timers, 100-ms timers, reversible drum counters, and decrementing counters
	SP16, SP20	16 total: one 1-ms timer and two analog timers (0.01 to 2.50 s, 0.1 to 25.0 s, or 1 to 250 s) plus 10-ms timers, 100-ms timers, reversible drum counters, a high-speed counter, and decrementing counters
Memory protection		User program memory: RAM/EEPROM Data-holding bits: RAM (20 days at 25°C*), can be stored in EEPROM *The power supply must be turned on for at least 10 minutes to charge battery. Length of memory backup is reduced at higher temperatures.
Program check		Check for no END(01) instruction and for instruction errors when RUN mode is entered.
PC link		Up to 4 CPUs linkable via Link Adapter (sold separately)

Appendix C
Programming Instructions and Execution Times

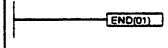
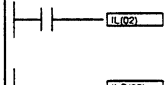
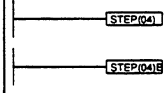
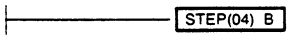
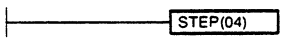
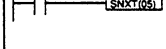
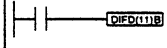
In the operand column, I refers to input bits, O to output bits, W to work bits, D to bits in the designated area, LR to Link Relay bits, DR to Data Retention bits, and TC to timer and counter Completion Flags and PVs. Refer to *3–2 Memory Areas* for details on bit and word designation.

Basic Instructions

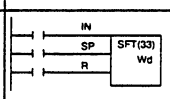
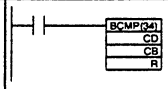
Name/ mnemonic	Symbol	Key inputs	Description	Operands*
LOAD LD		LD Bit address ENT	Creates a normally open condition as the first condition off the bus bar. All instruction lines begin with either LOAD or LOAD NOT.	**B:** I/O W D (03 and 04) LR DR TC
LOAD NOT LD NOT		LD NOT Bit address ENT	Creates a normally closed condition as the first condition off the bus bar. All instruction lines begin with either LOAD or LOAD NOT.	**B:** I/O W D (03 and 04) LR DR TC
AND AND		AND Bit address ENT	Combines a normally open condition in series with a previous condition.	**B:** I/O W D (03 and 04) LR DR TC
AND NOT AND NOT		AND NOT Bit address ENT	Combines a normally closed condition in series with a previous condition.	**B:** I/O W D (03 and 04) LR DR TC
OR OR		OR Bit address ENT	Combines a normally open condition in parallel with a previous condition.	**B:** I/O W D (03 and 04) LR DR TC
OR NOT OR NOT		OR NOT Bit address ENT	Combines a normally closed condition in parallel with a previous condition.	**B:** I/O W D (03 and 04) LR DR TC
AND LOAD AND LD		AND LD ENT	Combines two groups of conditions in series. These groups are called blocks.	---
OR LOAD OR LD		OR LD ENT	Combines two parallel groups of conditions. These groups are called blocks.	---
OUTPUT OUT		OUT Bit address ENT	Specifies an output bit that is to be turned ON for an ON execution condition and OFF for an OFF condition.	**B:** O W LR DR

Name/ mnemonic	Symbol	Key inputs	Description	Operands*
OUTPUT NOT OUT NOT		OUT NOT Bit address ENT	Specifies an output bit that is to be turned OFF for an ON execution condition and ON for an OFF condition.	**B:** O W LR DR
TIMER TIM		TIM TC number ENT SV ENT	Creates a 0.1-s decrementing timer that starts from the set value (SV) when the execution condition turns ON. SV can be between 0.0 and 999.9 s. When the SV has been timed out, the Completion Flag is turned ON.	**SP10** **N: SV:** TC # **SP16, SP20** **N: SV:** TC I/O W LR DR #
COUNTER CNT		CNT TC number ENT SV ENT	Counts down the number of times the input condition turns ON. Each time the input condition turns ON, the present value (PV) is reduced by 1 and when the count reaches 0, the Completion Flag (accessed through the counter number) turns ON. The SV can be between 0 and 9999.	**SP10** **N: SV:** TC # **SP16, SP20** **N: SV:** TC I/O W LR DR #
NO OPERATION NOP(00)	None	FUN 0 0 ENT	Does nothing. Can be inserted into a program before or after modifications are made to prevent program addresses from changing.	---

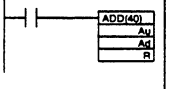
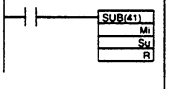
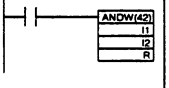
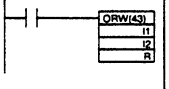
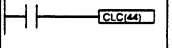
Special Instructions

Name/ mnemonic	Symbol	Key inputs	Description	Operands*
END END(01)		FUN 0 1 ENT	Indicates the end of the program. A program will not be executed unless the END instruction is used.	---
INTERLOCK IL(02) INTERLOCK CLEAR ILC(03)		FUN 0 2 ENT FUN 0 3 ENT	INTERLOCK and INTERLOCK CLEAR are combined to control the status of multiple outputs based on the execution condition of INTERLOCK, and are generally used to prevent specific bits from being ON simultaneously.	---
STEP DEFINE STEP(04)		FUN 0 4 Bit address ENT	Divides a program into sections called steps that can be executed as separate processes. Up to five steps can be created. Defining the Beginning of a Step (Operand Required)  Ending Step Execution (No Operand) 	**B:** O W LR DR
STEP START SNXT(05)		FUN 0 5 Bit address ENT	Turns OFF any previous steps and starts the designated step.	**B:** O W LR DR
DIFFERENTIATE UP DIFU(10)		FUN 1 0 Bit address ENT	Turns ON the designated bit for one scan only on the rising edge of the execution condition (input signal). Used when an operation is to be performed only once each time a signal turns ON.	**B:** O W LR DR
DIFFERENTIATE DOWN DIFD(11)		FUN 1 1 Bit address ENT	Turns ON the designated bit for one scan on the falling edge of the execution condition (input signal). Used when an operation is to be performed only once each time a signal turns OFF.	

Name/ mnemonic	Symbol	Key inputs	Description	Operands*
KEEP KEEP(12)	KEEP S R	[FUN] [1] [2] Bit address [ENT]	Latches bit status. The bit is set when the set input (I) turns ON and stays set until the reset input (R) turns ON.	B: O W LR DR
10-MS TIMER TIMM(20)	TIMM(20) N SV	[FUN] [2] [0] TC no. [ENT] SV [ENT]	Creates a 10-ms decrementing timer that starts from the set value (SV) when the execution condition turns ON. The SV can be between 0.00 and 99.99 s	N: SV: TC #
HIGH-SPEED TIMER TIMH(21)	TIMH(21) SV	[FUN] [2] [1] [ENT] SV [ENT]	Creates a 0.001-s decrementing timer that starts from the set value (SV) when the execution condition turns ON. The SV can be between 0.000 and 9.999 s.	SV: #
ANALOG TIMER ATIM(22)	ATIM(22)	[FUN] [2] [2] [ENT]	Creates a 0.1-s decrementing timer that starts from the set value (SV: 0.1 to 25.0 s) when the execution condition turns ON. Not all timer/counter instruction require input of the SV. Here it is adjusted with a manual adjustment on the PC.	---
REVERSIBLE DRUM COUNTER RDM(23)	RDM(23) N St R	[FUN] [2] [3] [ENT] TC no. [ENT] St [ENT] R [ENT]	Creates a counter that indicates when the present value is within specified ranges by turning ON specific bits in R. Used to turn operations ON and OFF for specific count ranges. St defines the size of the table, which starts in St+1.	N: St: R: TC DR O W LR DR
HIGH-SPEED COUNTER CNTH(24) (SP16, SP20)	CNTH(24) R	[FUN] [2] [4] [ENT] SV [ENT]	Creates a high-speed incrementing counter. The present value (PV) will be incremented by one whenever CP goes from OFF to ON as long as the start input (SI) is ON and the reset input (R) is OFF. The Completion Flag, CNT 13, is turned ON when the PV reaches the SV and will remain ON for one scan only. The PV is automatically reset to zero when the SV is reached. The SV can be between 0000 and 9999; setting 0000 creates an SV of 10,000.	SV: I/O W LR DR *DR #
ANALOG TIMER 1 ATM1(25) (SP16, SP20)	ATM1(25) RD	[FUN] [2] [5] [ENT] RD [ENT]	Creates a decrementing timer that starts from the set value determined by the #1 analog timer adjustment on the front of the CPU. If RD=0000, the SV ranges from 1 to 250 s. If RD=0001, the SV ranges from 0.1 to 25.0 s. If RD=0002, the SV ranges from 0.01 to 2.50 s. The SV can be input only via the hardware adjustment.	RD: I/O W LR DR *DR #
ANALOG TIMER 2 ATM2(26) (SP16, SP20)	ATM2(26) RD	[FUN] [2] [6] [ENT] RD [ENT]	Creates a decrementing timer that starts from the set value determined by the #2 analog timer adjustment on the front of the CPU. If RD=0000, the SV ranges from 1 to 250 s. If RD=0001, the SV ranges from 0.1 to 25.0 s. If RD=0002, the SV ranges from 0.01 to 2.50 s. The SV can be input only via the hardware adjustment.	RD: I/O W LR DR *DR #
MOVE MOV(30)	MOV(30) S D	[FUN] [3] [0] [ENT] S [ENT] D [ENT]	Moves the content of a specified word or a specified constant to a destination word. The Equals Flag will turn ON when 0 is moved.	S: D: I/O O W W D LR LR DR TC *DR DR *DR #
MOVE NOT MVN(31)	MVN(31) S D	[FUN] [3] [1] [ENT] S [ENT] D [ENT]	Moves the inverse content of a specified word or a specified constant to a destination word. The Equals Flag will turn ON when 0 is moved.	
COMPARE CMP(32)	CMP(32) Cp1 Cp2	[FUN] [3] [2] [ENT] Cp1 [ENT] Cp2 [ENT]	Compares the contents of two words or constants and turns ON the Equals, Less Than, or Greater Than Flag to indicate which value is larger. These flags can then be used to control operation based on this comparison.	Cp1./Cp2: I/O W LR TC DR *DR #

Name/ mnemonic	Symbol	Key inputs	Description	Operands*
SHIFT REGISTER SFT(33)		FUN 3 3 ENT Wd ENT	Shifts the input condition (IN) into a register and shifts the bits in the register on each rising edge of the shift pulse (SP). The register is reset to 0 when the reset input (R) turns ON.	**Wd:** O W LR DR
BLOCK COMPARE BCMP(34)		FUN 3 4 ENT CD ENT CB ENT R ENT	N is the least significant digit of CB and determines the size of the comparison block; there will be N+1 comparison ranges. BCMP(34) compares CD to the ranges defined by a block consisting of CB+1, CB+2, ..., CB+(2N+2). Each range is defined by two words, the first one providing the lower limit and the second word providing the upper limit. The corresponding bit of the result word, R, will be turned ON whenever CD is within the preset range.	**CD: CB: R:** I/O DR O W W D LR LR DR TC DR *DR # *DR

Arithmetic Instructions

Name/ mnemonic	Symbol	Key inputs	Description	Operands*
BCD ADD ADD(40)		FUN 4 0 ENT Au ENT Ad ENT R ENT	Adds two BCD (binary-coded decimal) values and the contents of the Carry Flag and places the result in the result word (R) and the Carry Flag. The Carry Flag must normally be cleared before executing ADD(40). Au + Ad + CY → R CY	**Au/Ad: R:** I/O O W W D LR LR DR TC *DR DR *DR #
BCD SUBTRACT SUB(41)		FUN 4 1 ENT Mi ENT Su ENT R ENT	Subtracts one BCD (binary-coded decimal) value and the contents of the Carry Flag from another BCD value and places the result in the result word (R) and the Carry Flag. The Carry Flag must normally be cleared before executing SUB(41). Mi - Su → CY → R CY	**Mi/Su: R:** I/O O W W D LR LR DR TC *DR DR *DR #
LOGICAL AND ANDW(42)		FUN 4 2 ENT I1 ENT I2 ENT R ENT	Performs an AND between two words one bit at a time and places the result in the result word (R). I1 AND I2 → R	**I1/I2: R:** I/O O W W D LR LR DR TC *DR DR *DR #
LOGICAL OR ORW(43)		FUN 4 3 ENT I1 ENT I2 ENT R ENT	Performs an OR between two words one bit at a time and places the result in the result word (R). I1 OR I2 → R	**I1/I2: R:** I/O O W W D LR LR DR TC *DR DR *DR #
CLEAR CARRY CLC(44)		FUN 4 4 ENT	Resets the Carry Flag (bit 0312) to 0. Generally used to clear the Carry Flag just before using ADD(40) or SUB(41).	---

Instruction Execution Times

The execution time is given in microseconds. "Word" indicates any data area address except for indirectly addressed DR (*DR).

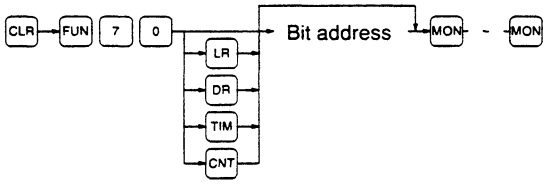
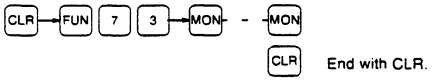
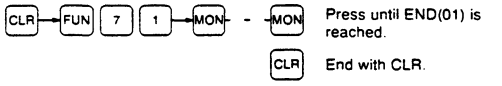
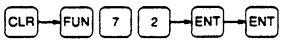
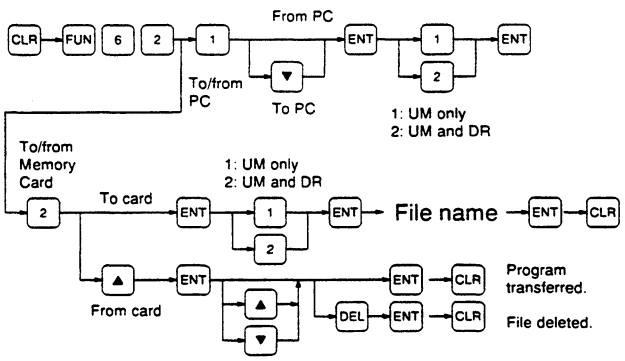
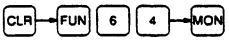
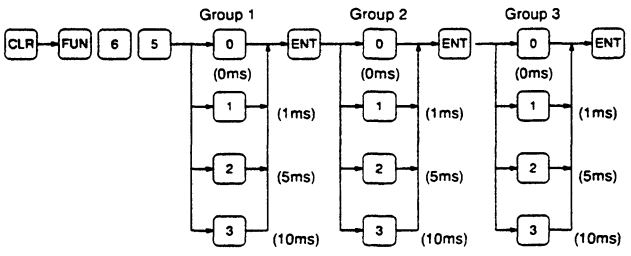
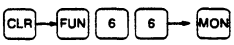
Instruction	Number of words		ON execution time		Conditions	OFF execution time
	I/O, work, or dedicated	DR, LR, or TC	I/O, work, or dedicated	LR, DR, or TC		
LD	2	3	0.4	0.8	Always	Same as ON time.
LD NOT	2	3	0.4	0.8	Always	Same as ON time.
AND	1	2	0.2	0.6	Always	Same as ON time.
AND NOT	1	2	0.2	0.6	Always	Same as ON time.
OR	1	2	0.2	0.6	Always	
OR NOT	1	2	0.2	0.6	Always	Same as ON time.
AND LD	2	2	0.4	0.6	Always	Same as ON time.
OR LD	2	2	0.4	0.6	Always	Same as ON time.
OUT	2	3	2.4	7.0	Always	Same as ON time.
OUT NOT	2	3	2.4	7.0	Always	Same as ON time.
TIM	---	4	---	24.4	Constant for SV	R: 23.0 IL: 24.8
				38.5	Word for SV (SP16, SP20 only)	R: 28.6 IL: 27.1
				63.0	*DR for SV (SP16, SP20 only)	
CNT	---	4	---	24.4	Constant for SV	R: 22.4 IL: 19.8
				38.8	Word for SV (SP16, SP20 only)	R: 28.9 IL: 4.8
				63.0	*DR for SV (SP16, SP20 only)	
NOP(00)	1	1	0.2		Always	N.A.
END(01)	1	1	9.8		Always	N.A.
IL(02)	2	2	19.6		Always	19.8
ILC(03)	1	1	0.2		Always	0.2
STEP(04)	4	4	35.4		Always	23.0
SNXT(05)	4	4	38.2		Always	28.4
DIFU(10)	4	4	39.2		Always	Normal: 32.8 IL: 20.0
DIFD(11)	4	4	40.4		Always	Normal: 34.4 IL: 20.2
KEEP(12)	4	4	29.0		Always	28.4
TIMM(20)	4	4	29.5		Constant for SV	R: 23.6 IL: 22.2
			25.8		Word for SV (SP16, SP20 only)	R: 28.8 IL: 27.1
			63.0		*DR for SV (SP16, SP20 only)	
TIMH(21)	4	4	26.0		Constant for SV	R: 23.6 IL: 22.2
			22.8		Word for SV (SP16, SP20 only)	R: 28.8 IL: 27.1
			59.9		*DR for SV (SP16, SP20 only)	
ATIM(22)	3	3	24.8		Always	R: 25.5 IL: 23.9
RDM(23)	5	5	69.4		Always	R: 68.0 IL: 19.4

Instruction	Number of words		ON execution time		Conditions	OFF execution time
	I/O, work, or dedicated	DR, LR, or TC	I/O, work, or dedicated	LR, DR, or TC		
CNTH(24) (SP16, SP20)	4	4	39.9		Constant for SV	R: 51.3 IL: 4.7
			49.4		Word for SV	
			73.6		*DR for SV	
ATM1(25) (SP16, SP20)	4	4	47.8		Constant for SV	R: 47.1 IL: 45.4
			56.3		Word for SV	
			81.4		*DR for SV	
ATM2(25) (SP16, SP20)	4	4	47.8		Constant for SV	R: 47.1 IL: 45.4
			56.3		Word for SV	
			81.4		*DR for SV	
MOV(30)	4	4	41.6 to 43.4		Moving constant to word.	20.6
			104.8		Moving *DR content to *DR word.	
MVN(31)	4	4	42.0 to 43.8		Moving constant to word.	20.6
			104.8		Moving *DR content to *DR word.	
CMP(32)	4	4	33.4 to 36.2		Comparing constant with word.	20.0
			97.2		Comparing *DR content.	
SFT(33)	3	3	35.2 to 41.8		Always	R: 32.2 IL: 19.6
BCMP(34) (SP16, SP20)	5	5	41.5 to 134.4		0 to 5 comparison ranges with a constant for compare data	R: 13.1 IL: 4.9
			43.0 to 136.0		0 to 5 comparison ranges with a word for compare data	
ADD(40)	5	5	70.2 to 72.6		Adding constant to word with results placed in word.	20.8
			167.8		Adding *DR to *DR with results placed in *DR.	
SUB(41)	5	5	70.2 to 72.6		Subtracting constant from word with results placed in word.	20.8
			167.4		Subtracting *DR from *DR with results placed in *DR.	
ANDW(42)	5	5	49.0 to 51.4		ANDing constant and word with results placed in word.	20.8
			146.6		ANDing *DR and *DR with results placed in *DR	
ORW(43)	5	5	49.0 to 51.6		ORing constant and word with results placed in word.	20.8
			146.6		ORing *DR and *DR with results placed in *DR	
CLC(44)	2	2	19.8		Always.	19.6

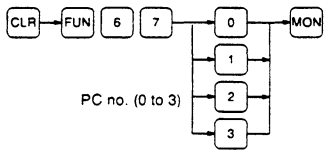
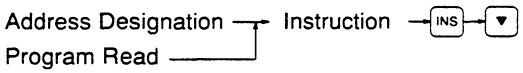
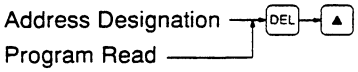
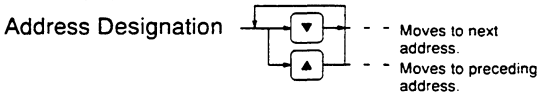
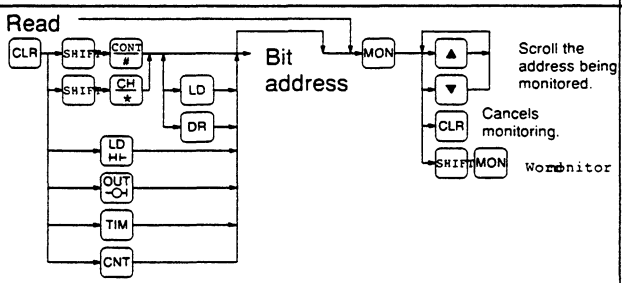
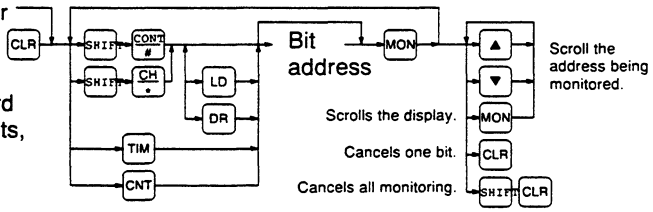
Appendix D
Programming Console Operations

If the display is not cleared to all zeros when the CLR Key is pressed at the beginning of a Programming Console operation, continue pressing the CLR Key until the display shows all zeros.

Name	Modes	Basic key sequences	Page
Password Input	RUN or PRGM	Check the operation mode and then input as follows: CLR → MON	51
Buzzer ON/OFF	RUN or PRGM	Input as follows after changing the mode: SHIFT → 1	52
Data Clear	PRGM only	Deletes the contents of user program memory. Press CNT and/or DR to preserve the contents of these areas. Specify an address to delete from that address to the end of user memory. User memory in both the PC and the Programming Console are delete simultaneously (including EEPROM). CLR → FUN 6 0 → CNT / DR → Address → ENT	53
Address Designation	RUN or PRGM	Jumps to the designated address. CLR → Address	54
Program Input	PRGM only	Used to input programs into user program memory. Address Designation / Program Read → Instruction → Operand → ENT → SV → ENT	57
Program Read	RUN or PRGM	Used to read the contents of user program memory. If executed in RUN mode, I/O bit status will be displayed. Address Designation / Program Read → ▼ Moves to next address. ▲ Moves to preceding address.	54
Binary Monitor	RUN or PRGM	Used to monitor up to 4 memory words in binary. Bit/Word Monitor / Multibit/Word Monitor. CLR → SHIFT → CH (*) → LR / DR → Word address → SHIFT → MON. ▲ ▼ Scroll address. CLR Shifts to Bit/Word or Multibit/Word Monitor. SHIFT CLR Cancels monitoring completely.	120
Force Set/Reset	RUN or PRGM	Used to control I/O status in RUN mode on bit displays. I/O bit status is refreshed each scan at which time forced status will be canceled. Bit/Word Monitor / Multibit/Word Monitor → CHG → ENT	117
HEX/BCD Data Change	RUN or PRGM	Used to change memory contents in either hexadecimal or BCD. Bit/Word Monitor / Multibit/Word Monitor → CHG → New value → ENT	118
Binary Data Change	RUN or PRGM	Used to change memory contents in binary. Use arrow keys to select bit, the 1 key to turn it ON, and the 0 key to turn it OFF. Word Monitor → CHG → ▼ ▲ → 1 / 0 → ENT	120
Error Message Read	RUN or PRGM	Used to read out and clear current error messages. The PC must be in PROGRAM mode to clear errors. CLR → FUN 6 1 → MON → MON. Error read. Error cleared and next error read.	109
LR Bit Allocation	PRGM only	Used to allocate LR bits. CLR → FUN 6 8 → 1 / 2 / 3 → ENT. 1: None 2: 64 bits (16 per PC) 3: 128 bits (32 per PC)	34
LR Allocation Read	RUN or PRGM	Used to read the number of LR bits that have been allocated. When performed with the SP10-PRO01-V1, both the PC and Programming Console settings are displayed. CLR → FUN 6 9 → MON	35

Name	Modes	Basic key sequences	Page
Bit Search	RUN or PRGM	Used to search the program for I/O bits, work bits, LR bits, HR bits and Timer/counter bits. 	61
Scan Time Read	RUN only	Used to display the maximum scan time of the program that is being executed. (SP16 and SP20 only) 	33
Program Check	PRGM only	Used to confirm that user memory contents are intact and that there is an END instruction. 	57
Card Format	PRGM only	Used to initialize memory cards. 	121
Program Transfer/ Delete	PRGM only	Used to transfer programs between the Programming Console, PCs, and memory cards and to delete program files. "UM" includes the filter values. 	58
DR Area Transfer	PRGM only	Used to transfer DR area contents to EEPROM.  Whenever this operation is executed, the contents of the DR area in RAM is transferred to EEPROM. Data backed up in EEPROM will then be automatically transferred back to RAM whenever power is turned on to the CPU. This feature can be used to ensure that the same data is set in the DR area each time power is turned on or to save DR area contents when CPU power is not turned on for an extended period of time. If the DR area is being used for other purposes, such as for holding data during power interruptions, the DR area contents of EEPROM must be cleared with the Data Clear operation to prevent RAM DR area contents from being overwritten when CPU power is turned on.	-
Program Compare	PRGM only	Used to compare the programs in the PC and in the Programming Console. 	-
Filter Value Designation	PRGM only	Used to adjust the input read filter time. Each group can be set to 0 ms, 1 ms, 5 ms, or 10 ms. Set all three groups at the same time. Factory settings are 10 ms. 	22
Filter Value Read	RUN or PRGM	Used to read the filter time. When performed with the SP10-PRO01-V1, both the PC and Programming Console settings are displayed. 	23

PC	Group 1	Group 2	Group 3
SP10	0 to 2	3 to 5	None
SP16	0 and 1	2 to 5	6 to 9
SP20	0 and 1	2 to 9	10 and 11

Name	Modes	Basic key sequences	Page
PC Designation	RUN or PRGM	Used to designate the PC being accessed when a Link Adapter is used to link multiple PCs. (Factory set to 0.) 	52
Instruction Insert	PRGM only	Used to insert an instruction at the address currently being displayed. 	61
Instruction Delete	PRGM only	Used to delete the instruction being displayed. 	61
Status Monitor	RUN only	Used to monitor status in RUN mode. 	54
Bit/Word Monitor	RUN or PRGM	Used to monitor I/O bits, other bit status, word status, or TC PV. 	115
Multibit/ Word Monitor	RUN or PRGM	Used to expand Bit/Word Monitor to up to three bits, words, and/or TC PV. 	115

Appendix E
Error and Arithmetic Flag Operation

The following table shows the instructions that affect the ER, CY, GT, LT and EQ flags. In general, ER indicates that operand data is not within requirements. CY indicates arithmetic or data shift results. GT indicates that a compared value is larger than some standard, LT that it is smaller, and EQ, that it is the same. EQ also indicates a result of zero for arithmetic operations. Refer to *Section 3 Instruction Set* for details.

Vertical arrows in the table indicate the flags that are turned ON and OFF according to the result of the instruction.

Although timer and counter instructions are executed when ER is ON, instructions with a vertical arrow under the ER column are not executed if ER is ON. All of the other flags in the following table will also not operate when ER is ON.

Instructions	0311 (ER)	0312 (CY)	0313 (LT)	0314 (EQ)	0315 (GT)
TIM	Unaffected	Unaffected	Unaffected	Unaffected	Unaffected
CNT	Unaffected	Unaffected	Unaffected	Unaffected	Unaffected
END(01)	OFF	OFF	OFF	OFF	OFF
STEP(04)	↕	Unaffected	Unaffected	Unaffected	Unaffected
SNXT(05)	↕	Unaffected	Unaffected	Unaffected	Unaffected
TIMM(20)	Unaffected	Unaffected	Unaffected	Unaffected	Unaffected
TIMH(21)	Unaffected	Unaffected	Unaffected	Unaffected	Unaffected
ATIM(22)	Unaffected	Unaffected	Unaffected	Unaffected	Unaffected
RDM(23)	↕	Unaffected	Unaffected	Unaffected	Unaffected
CNTH(24)	↕	Unaffected	Unaffected	Unaffected	Unaffected
ATM1(25)	Unaffected	Unaffected	Unaffected	Unaffected	Unaffected
ATM2(26)	Unaffected	Unaffected	Unaffected	Unaffected	Unaffected
MOV(30)	↕	Unaffected	Unaffected	↕	Unaffected
MVN(31)	↕	Unaffected	Unaffected	↕	Unaffected
CMP(32)	↕	Unaffected	↕	↕	↕
SFT(33)	↕	Unaffected	Unaffected	Unaffected	Unaffected
BCMP(34)	↕	Unaffected	Unaffected	Unaffected	Unaffected
ADD(40)	↕	↕	Unaffected	↕	Unaffected
SUB(41)	↕	↕	Unaffected	↕	Unaffected
ANDW(42)	↕	Unaffected	Unaffected	↕	Unaffected
ORW(43)	↕	Unaffected	Unaffected	↕	Unaffected
CLC(44)	Unaffected	OFF	Unaffected	Unaffected	Unaffected

Appendix F
I/O Assignment Sheets

This appendix contains sheets that can be copied by the programmer to record I/O bit allocations and terminal assignments, as well as details of work bits, data storage areas, timers, and counters.

Note Some bits appear as both I/O bits and work bits so that the I/O assignment sheets can be used for any of the SP-series PCs. Be sure that you do not assign a bit as a work bit if it is already being used as an I/O bit and that you do not assign more I/O bits than are supported by your PC.

No.: **System:**

Program: **Programmer:** **Date:**

Unit #0

Inputs

Bit	Field device	Notes
0000		
0001		
0002		
0003		
0004		
0005		
0006		
0007		
0008		
0009		
0010		
0011		

Outputs

Bit	Field device	Notes
0100		
0101		
0102		
0103		
0104		
0105		
0106		
0107		

Unit #1

Inputs

Bit	Field device	Notes
0000		
0001		
0002		
0003		
0004		
0005		
0006		
0007		
0008		
0009		
0010		
0011		

Outputs

Bit	Field device	Notes
0100		
0101		
0102		
0103		
0104		
0105		
0106		
0107		

No.: **System:**

Program: **Programmer:** **Date:**

Unit #2

Inputs

Bit	Field device	Notes
0000		
0001		
0002		
0003		
0004		
0005		
0006		
0007		
0008		
0009		
0010		
0011		

Outputs

Bit	Field device	Notes
0100		
0101		
0102		
0103		
0104		
0105		
0106		
0107		

Unit #3

Inputs

Bit	Field device	Notes
0000		
0001		
0002		
0003		
0004		
0005		
0006		
0007		
0008		
0009		
0010		
0011		

Outputs

Bit	Field device	Notes
0100		
0101		
0102		
0103		
0104		
0105		
0106		
0107		

No.: **System:**

Programmer: **Program:** **Date:** **Unit #:**

Timers and Counters

Address	T or C	Set value	Notes
00			
01			
01			
03			
04			
05			
06			
07			
08			
09			
10			
11			
12			
13			
14			
15			

Word 00

Bit	Usage	Notes
0008		
0009		
0010		
0011		
0012		
0013		
0014		
0015		

Word 01

Bit	Usage	Notes
0104		
0105		
0106		
0107		
0108		
0109		
0110		
0111		
0112		
0113		
0114		
0115		

Word 02

Bit	Usage	Notes
0200		
0201		
0202		
0203		
0204		
0205		
0206		
0207		
0208		
0209		
0210		
0211		
0212		
0213		
0214		
0215		

Word:

Bit	Usage	Notes
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

Word:

Bit	Usage	Notes
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

Word:

Bit	Usage	Notes
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

Word:

Bit	Usage	Notes
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

No.: **System:**

Programmer: **Program:** **Date:** **Unit #:**

Word	Contents	Notes

Word	Contents	Notes

Appendix G
Program Coding Sheet

The following pages can be copied for use in coding ladder diagram programs.

When coding programs, be sure to specify all function codes for instructions and data areas (or # for constant) for operands. These will be necessary when inputting programs though a Programming Console or other Peripheral Device.

No.: **System:** **Page 1**

Program: **Programmer:** **Date:**

Address	Instruction	Operand(s)
000		
001		
002		
003		
004		
005		
006		
007		
008		
009		
010		
011		
012		
013		
014		
015		
016		
017		
018		
019		
020		
021		
022		
023		
024		
025		
026		
027		
028		
029		
030		
031		
032		
033		
034		
035		
036		
037		

Address	Instruction	Operand(s)
038		
039		
040		
041		
042		
043		
044		
045		
046		
047		
048		
049		
050		
051		
052		
053		
054		
055		
056		
057		
058		
059		
060		
061		
062		
063		
064		
065		
066		
067		
068		
069		
070		
071		
072		
073		
074		
075		

No.: **System:** **Page 2**

Program: **Programmer:** **Date:**

Address	Instruction	Operand(s)
076		
077		
078		
079		
080		
081		
082		
083		
084		
085		
086		
087		
088		
089		
090		
091		
092		
093		
094		
095		
096		
097		
098		
099		
100		
101		
102		
103		
104		
105		
106		
107		
108		
109		
110		
111		
112		
113		

Address	Instruction	Operand(s)
114		
115		
116		
117		
118		
119		
120		
121		
122		
123		
124		
125		
126		
127		
128		
129		
130		
131		
132		
133		
134		
135		
136		
137		
138		
139		
140		
141		
142		
143		
144		
145		
146		
147		
148		
149		
150		
151		

No.: **System:** **Page 3**

Program: **Programmer:** **Date:**

Address	Instruction	Operand(s)
152		
153		
154		
155		
156		
157		
158		
159		
160		
161		
162		
163		
164		
165		
166		
167		
168		
169		
170		
171		
172		
173		
174		
175		
176		
177		
178		
179		
180		
181		
182		
183		
184		
185		
186		
187		
188		
189		

Address	Instruction	Operand(s)
190		
191		
192		
193		
194		
195		
196		
197		
198		
199		
200		
201		
202		
203		
204		
205		
206		
207		
208		
209		
210		
211		
212		
213		
214		
215		
216		
217		
218		
219		
220		
221		
222		
223		
224		
225		
226		
227		

No.: **System:** **Page 4**

Program: **Programmer:** **Date:**

Address	Instruction	Operand(s)
228		
229		
230		
231		
232		
233		
234		
235		
236		
237		
238		
239		
240		
241		
242		
243		
244		
245		
246		
247		
248		
249		
250		
251		
252		
253		
254		
255		
256		
257		
258		
259		
260		
261		
262		
263		
264		
265		

Address	Instruction	Operand(s)
266		
267		
268		
269		
270		
271		
272		
273		
274		
275		
276		
277		
278		
279		
280		
281		
282		
283		
284		
285		
286		
287		
288		
289		
290		
291		
292		
293		
294		
295		
296		
297		
298		
299		
300		
301		
302		
303		

No.: **System:** **Page 5**

Program: **Programmer:** **Date:**

Address	Instruction	Operand(s)
304		
305		
306		
307		
308		
309		
310		
311		
312		
313		
314		
315		
316		
317		
318		
319		
320		
321		
322		
323		
324		
325		
326		

Address	Instruction	Operand(s)
327		
328		
329		
330		
331		
332		
333		
334		
335		
336		
337		
338		
339		
340		
341		
342		
343		
344		
345		
346		
347		
348		

Glossary

address The location in memory where data is stored. For data areas, an address consists of a two-letter data area designation and a number that designates the word and/or bit location. For the UM area, an address designates the instruction location (UM area).

allocation The process by which the PC assigns certain bits or words in memory for various functions. This includes pairing I/O bits to I/O points.

AND A logic operation whereby the result is true if and only if both premises are true. In ladder-diagram programming the premises are usually ON/OFF states of bits or the logical combination of such states called execution conditions.

BCD Short for binary-coded decimal.

BCD calculation An arithmetic calculation that uses numbers expressed in binary-coded decimal.

binary A number system where all numbers are expressed to the base 2, i.e., any number can be written using only 1's or 2's. Each group of four binary bits is equivalent to one hexadecimal digit.

binary calculation An arithmetic calculation that uses numbers expressed in binary.

binary-coded decimal A system used to represent numbers so that each group of four binary bits is numerically equivalent to one decimal digit.

bit A binary digit; hence a unit of data in binary notation. The smallest unit of information that can be electronically stored in a PC. The status of a bit is either ON or OFF. Different bits at particular addresses are allocated to special purposes, such as holding the status input from external devices, while other bits are available for general use in programming.

bit address The location in memory where a bit of data is stored. A bit address must specify (sometimes by default) the data area and word that is being addressed, as well as the number of the bit.

bit number A number that indicates the location of a bit within a word. Bit 00 is the rightmost (least-significant) bit; bit 15 is the leftmost (most-significant) bit.

bus bar The line leading down the left and sometimes right side of a ladder diagram. Instruction execution proceeds down the bus bar, which is the starting point for all instruction lines.

carry flag A flag that is used with arithmetic operations to hold a carry from an addition or multiplication operation, or to indicate that the result is negative in a subtraction operation. The carry flag is also used with certain types of shift operations.

clock pulse A pulse available at a certain bit in memory for use in timing operations. Various clock pulses are available with different pulse widths.

clock pulse bit — A bit in memory that supplies a pulse that can be used to time operations. Various clock pulse bits are available with different pulse widths, and therefore different frequencies.

common data — Data that is stored in the LR Area of a PC and which is shared by other PCs in the same the same system. Each PC has a specified section of the LR Area allocated to it. This allocation is the same in each LR Area of each PC.

condition — An message placed in an instruction line to direct the way in which the terminal instructions, on the right side, are to be executed. Each condition is assigned to a bit in memory that determines its status. The status of the bit assigned to each condition determines, in turn, the execution condition for each instruction up to a terminal instruction on the right side of the ladder diagram.

constant — An operand for which the actual numeric value is specified by the user, and which is then stored in a particular address in the data memory.

control bit — A bit in a memory area that is set either through the program or via a Programming Device to achieve a specific purpose, e.g., a Restart bit is turned ON and OFF to restart a Unit.

Control System — All of the hardware and software components used to control other devices. A Control System includes the PC System, the PC programs, and all I/O devices that are used to control or obtain feedback from the controlled system.

controlled system — The devices that are being controlled by a PC System.

control signal — A signal sent from the PC to effect the operation of the controlled system.

counter — A dedicated group of digits or words in memory used to count the number of times a specific process has occurred, or a location in memory accessed through a TC bit and used to count the number of times the status of a bit or an execution condition has changed from OFF to ON.

CPU — An acronym for central processing unit. In a PC System, the CPU executes the program, processes I/O signals, communicates with external devices, etc.

data area — An area in the PC's memory that is designed to hold a specific type of data, e.g., the LR area is designed to hold common data in a PC Link System. Memory areas that hold programs are not considered data areas.

data area boundary — The highest address available within a data area. When designating an operand that requires multiple words, it is necessary to ensure that the highest address in the data area is not exceeded.

data sharing — An aspect of Data Links created through Link Adapters in which common data areas or common data words are created between two or more PCs.

debug — A process by which a draft program is corrected until it operates as intended. Debugging includes both the removal of syntax errors, as well as the fine-tuning of timing and coordination of control operations.

decimal — A number system where all numbers are expressed to the base 10. In a PC all data is ultimately stored in binary form, four binary bits are often used to represent one decimal digit, via a system called binary-coded decimal.

decrement	Decreasing a numeric value.
default	A value automatically set by the PC when the user omits to set a specific value. Many devices will assume such default conditions upon the application of power.
delay	In tracing, a value that specifies where tracing is to begin in relationship to the trigger. A delay can be either positive or negative, i.e., can designate an offset on either side of the trigger.
destination	The location where an instruction is to place the data on which it is operating, as opposed to the location from which data is taken for use in the instruction. The location from which data is taken is called the source.
differentiated instruction	An instruction used to ensure that the operand bit is never turned ON for more than one scan after the execution condition goes either from OFF to ON for a Differentiate Up instruction or from ON to OFF for a Differentiate Down instruction.
digit	A unit of storage in memory that consists of four bits.
distributed control	An automation concept in which control of each portion of an automated system is located near the devices actually being controlled, i.e., control is decentralized and 'distributed' over the system. Distributed control is one of the fundamental concepts of PC Systems.
download	The process of transferring a program or data from a higher-level computer to a lower-level computer or PC or between peripheral devices and the PC.
electrical noise	Random variations of one or more electrical characteristics such as voltage, current, and data, which might interfere with the normal operation of a device.
exection condition	The ON or OFF status under which an instruction is executed. The execution condition is determined by the logical combination of conditions on the same instruction line and up to the instruction currently being executed.
execution time	The time required for the CPU to execute either an individual instruction or an entire program.
extended counter	A counter created in a program by using two or more count instructions in succession. Such a counter is capable of counting higher than any of the standard counters provided by the individual instructions.
extended timer	A timer created in a program by using two or more timers in succession. Such a timer is capable of timing longer than any of the standard timers provided by the individual instructions.
fatal error	An error that stops PC operation and requires correction before operation can continue.
flag	A dedicated bit in memory that is set by the system to indicate some type of operating status. Some flags, such as the carry flag, can also be set by the operator or via the program.
flicker bit	A bit that is programmed to turn ON and OFF at a specific frequency.

force reset — The process of forcibly turning OFF a bit via a programming device. Bits are usually turned OFF as a result of program execution.

force set — The process of forcibly turning ON a bit via a programming device. Bits are usually turned ON as a result of program execution.

function code — A two-digit number used to input an instruction into the PC.

hardware error — An error originating in the hardware structure (electronic components) of the PC, as opposed to a software error, which originates in software (i.e., programs).

hexadecimal — A number system where all numbers are expressed to the base 16. In a PC all data is ultimately stored in binary form, however, displays and inputs on Programming Devices are often expressed in hexadecimal to simplify operation. Each group of four binary bits is numerically equivalent to one hexadecimal digit.

increment — Increasing a numeric value.

indirect address — An address whose contents indicates another address. The contents of the second address will be used as the operand. Indirect addressing is possible in the DR area only .

initialize — Part of the startup process whereby some memory areas are cleared, system setup is checked, and default values are set.

input — The signal coming from an external device into the PC. The term input is often used abstractly or collectively to refer to incoming signals.

input bit — A bit that is allocated to hold the status of an input.

input device — An external device that sends signals into the PC System.

input point — The point at which an input enters the PC System. Input points correspond physically to terminals or connector pins.

input signal — A change in the status of a connection entering the PC. Generally an input signal is said to exist when, for example, a connection point goes from low to high voltage or from a nonconductive to a conductive state.

instruction — A direction given in the program that tells the PC of an action to be carried out, and which data is to be used in carrying out the action. Instructions can be used to simply turn a bit ON or OFF, or they can perform much more complex actions, such as converting and/or transferring large blocks of data.

instruction block — A group of instructions that is logically related in a ladder-diagram program. Although any logically related group of instructions could be called an instruction block, the term is generally used to refer to blocks of instructions called logic blocks that require logic block instructions to relate them to other instructions or logic blocks.

instruction execution time — The time required to execute an instruction. The execution time for any one instruction can vary with the execution conditions for the instruction and the operands used within it.

instruction line — A group of conditions that lie together on the same horizontal line of a ladder diagram. Instruction lines can branch apart or join together to form instruction blocks.

interlock — A programming method used to treat a number of instructions as a group so that the entire group can be reset together when individual execution is not required. An interlocked program section is executed normally for an ON execution condition and partially reset for an OFF execution condition.

I/O capacity — The number of inputs and outputs that a PC is able to handle. This number ranges from around 10 for smaller PCs to two thousand for the largest ones.

I/O devices — The devices to which terminals on I/O Units or Special I/O Units, or other Units are connected. I/O devices may be either part of the Control System, if they function to help control other devices, or they may be part of the controlled system.

I/O point — The place at which an input signal enters the PC System, or at which an output signal leaves the PC System. In physical terms, I/O points correspond to terminals or connector pins on a Unit; in terms of programming, an I/O points correspond to I/O bits in memory.

I/O response time — The time required for an output signal to be sent from the PC in response to an input signal received from an external device.

JIS — Acronym for Japanese Industrial Standards.

ladder diagram (program) — A form of program arising out of relay-based control systems that uses circuit-type diagrams to represent the logic flow of programming instructions. The appearance of the program is similar to a ladder, and thus the name.

ladder diagram symbol — A symbol used in a ladder-diagram program.

ladder instruction — An instruction that represents the 'rung' portion of a ladder-diagram program. The other instructions in a ladder diagram fall along the right side of the diagram and are called terminal instructions.

leftmost (bit/word) — The highest numbered bits of a group of bits, generally of an entire word, or the highest numbered words of a group of words. These bits/words are often called most-significant bits/words.

Link Adapter — A Unit used to connect up to four CPUs to enable created data links in the LR area.

logic block — A group of instructions that is logically related in a ladder-diagram program and that requires logic block instructions to relate it to other instructions or logic blocks.

logic block instruction — An instruction used to locally combine the execution condition resulting from a logic block with a current execution condition. The current execution condition could be the result of a single condition, or of another logic block. AND Load and OR Load are the two logic block instructions.

logic instruction — Instructions used to logically combine the content of two words and output the logical results to a specified result word. The logic instructions combine

	all the same-numbered bits in the two words and output the result to the bit of the same number in the specified result word.
memory area	Any of the areas in the PC used to hold data or programs.
mnemonic code	A form of a ladder-diagram program that consists of a sequential list of the instructions without using a ladder diagram. Mnemonic code is required to input a program into a PC when using a Programming Console.
most-significant (bit/word)	See *leftmost (bit/word).*
NC input	An input that is normally closed, i.e., the input signal is considered to be present when the circuit connected to the input opens.
NO input	An input that is normally open, i.e., the input signal is considered to be present when the circuit connected to the input closes.
noise interference	Disturbances in signals caused by electrical noise.
nonfatal error	A hardware or software error that produces a warning but does not stop the PC from operating.
normally closed condition	A condition that produces an ON execution condition when the bit assigned to it is OFF, and an OFF execution condition when the bit assigned to it is ON.
normally closed condition	A condition that produces an ON execution condition when the bit assigned to it is ON, and an OFF execution condition when the bit assigned to it is OFF.
NOT	A logic operation which inverts the status of the operand. For example, AND NOT indicates an AND operation with the opposite of the actual status of the operand bit.
OFF	The status of an input or output when a signal is said not to be present. The OFF state is generally represented by a low voltage or by non-conductivity, but can be defined as the opposite of either.
OFF delay	The delay between the time when a signal is switched OFF (e.g., by an input device or PC) and the time when the signal reaches a state readable as an OFF signal (i.e., as no signal) by a receiving party (e.g., output device or PC).
ON	The status of an input or output when a signal is said to be present. The ON state is generally represented by a high voltage or by conductivity, but can be defined as the opposite of either.
ON delay	The delay between the time when an ON signal is initiated (e.g., by an input device or PC) and the time when the signal reaches a state readable as an ON signal by a receiving party (e.g., output device or PC).
one-shot bit	A bit that is turned ON or OFF for a specified interval of time which is longer than one scan.
operand	Bit(s) or word(s) designated as the data to be used for an instruction. An operand can be input as a constant expressing the actual numeric value to be

	used or as an address to express the location in memory of the data to be used.
operand bit	A bit designated as an operand for an instruction.
operand word	A word designated as an operand for an instruction.
operating error	An error that occurs during actual PC operation as opposed to an initialization error, which occurs before actual operations can begin.
OR	A logic operation whereby the result is true if either of two premises is true, or if both are true. In ladder-diagram programming the premises are usually ON/OFF states of bits or the logical combination of such states called execution conditions.
output	The signal sent from the PC to an external device. the term output is often used abstractly or collectively to refer to outgoing signals.
output bit	A bit in memory that is allocated to hold the status to be sent to an output device.
output device	An external device that receives signals from the PC System.
output point	The point at which an output leaves the PC System. Output points correspond physically to terminals or connector pins.
output signal	A signal being sent to an external device. Generally an output signal is said to exist when, for example, a connection point goes from low to high voltage or from a nonconductive to a conductive state.
overseeing	Part of the processing performed by the CPU that includes general tasks required to operate the PC.
overwrite	Changing the content of a memory location so that the previous content is lost.
PC	An acronym for Programmable Controller.
PC configuration	The arrangement and interconnections of the Units that are put together to form a functional PC.
peripheral device	Devices connected to a PC System to aid in system operation. Peripheral devices include printers, programming devices, external storage media, etc.
port	A connector on a PC or computer that serves as a connection to an external device.
present value	The current value registered in a device at any instant during its operation. Present value is abbreviated as PV.
printed circuit board	A board onto which electrical circuits are printed for mounting into a computer or electrical device.
program	The list of instructions that tells the PC the sequence of control actions to be carried out.

Programmable Controller A computerized device that can accept inputs from external devices and generate outputs to external devices according to a program held in memory. Programmable Controllers are used to automate control of external devices. Although single-component Programmable Controllers are available, building-block Programmable Controllers are constructed from separate components. Such building-block Programmable Controllers are formed only when enough of these separate components are assembled to form a functional assembly, i.e., no one individual Unit is called a PC.

programmed alarm An alarm given as a result of execution of an instruction designed to generate the alarm in the program, as opposed to one generated by the system.

programmed error An error arising as a result of the execution of an instruction designed to generate the error in the program, as opposed to one generated by the system.

programmed message A message generated as a result of execution of an instruction designed to generate the message in the program, as opposed to one generated by the system.

Programming Console The simplest form or programming device available for a PC. Programming Consoles are available both as hand-held models and, for larger PCs, as CPU-mounting models.

PROGRAM mode A mode of operation that allows inputting and debugging of programs to be carried out, but that does not permit normal execution of the program.

PV Acronym for present value.

refresh The process of updating output status sent to external devices so that it agrees with the status of output bits held in memory and of updating input bits in memory so that they agree with the status of inputs from external devices.

relay-based control The forerunner of PCs. In relay-based control, groups of relays are interconnected to form control circuits. In a PC, these are replaced by programmable circuits.

reset The process of turning a bit or signal OFF or of changing the present value of a timer or counter to its set value or to zero.

return The process by which instruction execution shifts from a subroutine back to the main program (usually the point from which the subroutine was called).

reversible counter A counter that can be both incremented and decremented depending on the specified conditions.

right-hand instruction Another term for terminal instruction.

rightmost (bit/word) The lowest numbered bits of a group of bits, generally of an entire word, or the lowest numbered words of a group of words. These bits/words are often called least-significant bits/words.

RUN mode The operating mode used by the PC for normal control operations.

scan The process used to execute a ladder-diagram program. The program is examined sequentially from start to finish and each instruction is executed in turn based on execution conditions.

scan time The time required for a single scan of the ladder-diagram program.

self diagnosis A process whereby the system checks its own operation and generates a warning or error if an abnormality is discovered.

self-maintaining bit A bit that is programmed to maintain either an OFF or ON status until set or reset by specified conditions.

set The process of turning a bit or signal ON.

set value The value from which a decrementing counter starts counting down or to which an incrementing counter counts up (i.e., the maximum count), or the time from which or for which a timer starts timing. Set value is abbreviated SV.

shift register One or more words in which data is shifted a specified number of units to the right or left in bit, digit, or word units. In a rotate register, data shifted out one end is shifted back into the other end. In other shift registers, new data (either specified data, zero(s) or one(s)) is shifted into one end and the data shifted out at the other end is lost.

slot A position on a Rack (Backplane) to which a Unit can be mounted.

software error An error that originates in a software program.

source The location from which data is taken for use in an instruction, as opposed to the location to which the result of an instruction is to be written. The latter is called the destination.

SV Abbreviation for set value.

switching capacity The maximum voltage/current that a relay can safely switch on and off.

syntax error An error in the way in which a program is written. Syntax errors can include 'spelling' mistakes (i.e., a function code that does not exist), mistakes in specifying operands within acceptable parameters (e.g., specifying reserved SR bits as a destination), and mistakes in actual application of instructions (e.g., a call to a subroutine that does not exist).

system configuration The arrangement in which Units in a system are connected.

system error An error generated by the system, as opposed to one resulting from execution of an instruction designed to generate an error.

system error message An error message generated by the system, as opposed to one resulting from execution of an instruction designed to generate a message.

TC area A data area that can be used only for timers and counters. Each bit in the TC area serves as the access point for the SV, PV, and Completion Flag for the timer or counter defined with that bit.

TC number A definer that corresponds to a bit in the TC area and used to define the bit as either a timer or a counter.

terminal instruction An instruction placed on the right side of a ladder diagram that uses the final execution conditions of an instruction line.

timer — A location in memory accessed through a TC bit and used to time down from the timer's set value. Timers are turned ON and reset according to their execution conditions.

transfer — The process of moving data from one location to another within the PC, or between the PC and external devices. When data is transferred, generally a copy of the data is sent to the destination, i.e., the content of the source of the transfer is not changed.

UM area — The memory area used to hold the active program, i.e., the program that is being currently executed.

watchdog timer — A timer within the system that ensures that the scan time stays within specified limits. When limits are reached, either warnings are given or PC operation is stopped depending on the particular limit that is reached.

word — A unit of data storage in memory that consists of 16 bits. All data areas consists of words. Some data areas can be accessed only by words; others, by either words or bits.

word address — The location in memory where a word of data is stored. A word address must specify (sometimes by default) the data area and the number of the word that is being addressed.

work bit — A bit that can be used for data calculation or other manipulation in programming, i.e., a 'work space' in memory.

Index

H

I

L

M

N

O

P

R

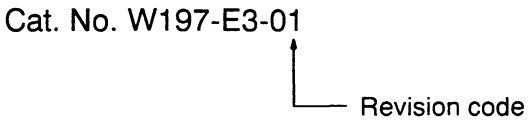
S

T

W

Revision History

A manual revision code appears as a suffix to the catalog number on the front cover of the manual.



The following table outlines the changes made to the manual during each revision. Page numbers refer to the previous version.

Revision code	Date	Revised content
1	March 1991	Original production
1A	September 1991	**Page 132:** Description of DR Area Transfer expanded.
2	March 1992	Information added for two new PCs: the SP16 and SP20. Additions include a scan time read Programming Console operation and the following instructions: HIGH-SPEED COUNTER (CNTH(24)), ANALOG TIMER 1 (ATM1(25)), ANALOG TIMER 2 (ATM2(26)), and BLOCK COMPARE (BCMP(34)). Other instructions and operations have also been updated for the new PCs. These new features are not supported by the SP10. The following mistake was also corrected. **Page 89:** Operation when a lower limit is greater than an upper limit corrected.
E3-01	October 1992	**Page 19:** Capacitive ground added. **Page 20:** Capacitive ground note added. **Page 131:** Input frequency added.

OMRON

OMRON ELECTRONICS LLC

1 Commerce Drive
Schaumburg, IL 60173 USA
800.55.OMRON (66766)

OMRON CANADA, INC.

885 Milner Avenue
Scarborough, Ontario M1B 5V8
416.286.6465

OMRON ON-LINE

Global - http://www.omron.com
USA - http://www.omron.com/oei
Canada - http://www.omron.com/oci

For Distributor Locations or Product Information, Call:

800.55.OMRON or 847.843.7900

UNITED STATES REGIONAL SALES OFFICES		
Northeast	Boston, MA	508.303.8880
East	Philadelphia, PA	610.524.1897
Southeast	Atlanta, GA	770.798.6780
Central	Cincinnati, OH	513.469.6766
Midwest	Chicago, IL	847.843.7910
Southwest	Dallas, TX	972.871.2166
West	Los Angeles, CA	714.621.3455

CANADA REGIONAL SALES OFFICES		
Ontario	Toronto	416.286.6465
	Kitchener	519.896.1144
	Kingston	613.376.3968
Quebec	Montreal	514.636.6676
	Ste-Foy	418.864.7378
British Columbia	Vancouver	604.522.8855
Alberta	Edmonton	403.440.0818
	Calgary	403.257.3095

BRAZIL SALES OFFICE	
Sao Paulo	55.11.5564.6488

ARGENTINA SALES OFFICE	
Buenos Aires	54.114.787.1129

MEXICO/LATIN AMERICA SALES OFFICE	
Florida	954.227.2121

W197-E3-1 Printed in U.S.A.