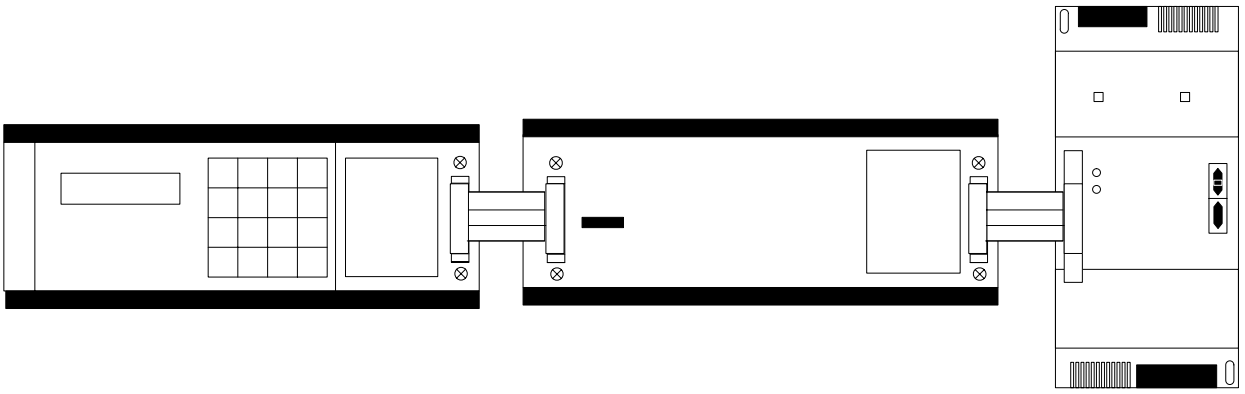
K-type Programmable Controllers

OPERATION MANUAL

Revised July 1999



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 precautions in this manual. Always heed the information provided with them. Failure to heed precautions can result in injury to people or damage to property.

⚠ DANGER Indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury.

⚠ WARNING Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury.

⚠ Caution Indicates a potentially hazardous situation which, if not avoided, may result in minor or moderate injury, or property damage.

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, often means “word” and is abbreviated “Wd” in documentation in this sense.

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... 1. Indicates lists of one sort or another, such as procedures, checklists, etc.

TABLE OF CONTENTS

PRECAUTIONS . . . . . .

1 Intended Audience . . . . . .
2 General Precautions . . . . . .
3 Safety Precautions . . . . . .
4 Operating Environment Precautions . . . . . .
5 Application Precautions . . . . . .

SECTION 1
Background . . . . . .

1-1 Introduction . . . . . .
1-2 Relay Circuits: The Roots of PC Logic . . . . . .
1-3 PC Terminology . . . . . .
1-4 OMRON Product Terminology . . . . . .
1-5 Overview of PC Operation . . . . . .
1-6 Peripheral Devices . . . . . .
1-7 Available Manuals . . . . . .

SECTION 2
Hardware Considerations . . . . . .

2-1 Introduction . . . . . .
2-2 Indicators . . . . . .
2-3 PC Configuration . . . . . .

SECTION 3
Memory Areas . . . . . .

3-1 Introduction . . . . . .
3-2 Data Area Structure . . . . . .
3-3 Internal Relay (IR) Area . . . . . .
3-4 Special Relay (SR) Area . . . . . .
3-5 Data Memory (DM) Area . . . . . .
3-6 Holding Relay (HR) Area . . . . . .
3-7 Timer/Counter (TC) Area . . . . . .
3-8 Temporary Relay (TR) Area . . . . . .

SECTION 4
Writing and Inputting the Program . . . . . .

4-1 Introduction . . . . . .
4-2 Instruction Terminology . . . . . .
4-3 The Ladder Diagram . . . . . .
4-4 The Programming Console . . . . . .
4-5 Preparation for Operation . . . . . .
4-6 Inputting, Modifying, and Checking the Program . . . . . .
4-7 Controlling Bit Status . . . . . .
4-8 Work Bits (Internal Relays) . . . . . .
4-9 Programming Precautions . . . . . .
4-10 Program Execution . . . . . .

TABLE OF CONTENTS

SECTION 5
Instruction Set . . . . . .

5-1 Introduction . . . . .
5-2 Notation . . . . .
5-3 Instruction Format . . . . .
5-4 Data Areas, Definer Values, and Flags . . . . .
5-5 Ladder Diagram Instructions . . . . .
5-6 Bit Control Instructions . . . . .
5-7 INTERLOCK and INTERLOCK CLEAR – IL(02) and ILC(03) . . . . .
5-8 JUMP and JUMP END – JMP(04) and JME(05) . . . . .
5-9 END – END(01) . . . . .
5-10 NO OPERATION – NOP(00) . . . . .
5-11 Timer and Counter Instructions . . . . .
5-12 Data Shifting . . . . .
5-13 Data Movement . . . . .
5-14 DATA COMPARE – CMP(20) . . . . .
5-15 Data Conversion . . . . .
5-16 BCD Calculations . . . . .
5-17 Subroutines . . . . .
5-18 Step Instructions . . . . .
5-19 Special Instructions . . . . .

SECTION 6
Program Execution Timing . . . . . .

6-1 Introduction . . . . .
6-2 Cycle Time . . . . .
6-3 Calculating Cycle Time . . . . .
6-4 Instruction Execution Times . . . . .
6-5 I/O Response Time . . . . .

SECTION 7
Program Debugging and Execution . . . . . .

7-1 Introduction . . . . .
7-2 Debugging . . . . .
7-3 Monitoring Operation and Modifying Data . . . . .
7-4 Program Backup and Restore Operations . . . . .

SECTION 8
Troubleshooting . . . . . .

8-1 Introduction . . . . .
8-2 Reading and Clearing Errors and Messages . . . . .
8-3 Error Messages . . . . .
8-4 Error Flags . . . . .

Appendices

A Standard Models . . . . .
B Programming Instructions and Execution Times . . . . .
C Programming Console Operations . . . . .
D Error and Arithmetic Flag Operation . . . . .
E Binary–Hexadecimal–Decimal Table . . . . .
F Word Assignment Recording Sheets . . . . .
G Program Coding Sheet . . . . .

Glossary . . . . . .

Index . . . . . .

Revision History . . . . . .

About this Manual:

The OMRON K-type Programmable Controllers offer an effective way to automate processing, manufacturing, assembly, packaging, and many other processes to save time and money. Distributed control systems can also be designed to allow centralized monitoring and supervision of several separate controlled systems. Monitoring and supervising can be done through a host computer, connecting the controlled system to a data bank. It is thus possible to have adjustments in system operation made automatically to compensate for requirement changes.

The K-type Units can utilize a number of additional Units including dedicated Special I/O Units that can be used for specific tasks and Link Units that can be used to build more highly integrated systems.

The K-types are equipped with large programming instruction sets, data areas, and other features to control processing directly. Programming utilizes ladder-diagram programming methods, which are described in detail for those unfamiliar with them.

This manual describes the characteristics and abilities of the K-types programming operations, instructions, and other aspects of operation and preparation that demand attention. Before attempting to operate the PC, thoroughly familiarize yourself with the information contained herein. Hardware information is provided in detail in the *Installation Guide*. A table of other manuals that can be used in combination with this manual is provided at the end of *Section 1 Introduction*.

Section 1 Introduction explains the background and some of the basic 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 peripheral devices used with the K-types and a table of other manuals available to use with this manual for special PC applications are also provided.

Section 2 Hardware Considerations explains basic aspects of the overall PC configuration and describes the indicators that are referred to in other sections of this manual.

Section 3 Memory Areas takes a look at the way memory is divided and allocated and explains the information provided there to aid in programming. It also explains how I/O is managed in memory and how bits in memory correspond to specific I/O points.

Section 4 Programming explains the basics of writing and inputting the ladder-diagram program, looking at the elements that make up the 'ladder' part of a ladder-diagram program and explaining how execution of this program is controlled and the methods required to input it input the PC. ***Section 5 Instruction Set*** then goes on to describe individually all of the instructions used in programming, while ***Section 6 Program Execution Timing*** explains the scanning process used to execute the program and tells how to coordinate inputs and outputs so that they occur at the proper times.

Section 7 Debugging and Execution provides the Programming Console procedures used to debug the program and to monitor and control system operation.

Finally, ***Section 8 Troubleshooting*** provides information on system error indications and other means of reducing system down time. Information in this section is also necessary when debugging a program.

The ***Appendices*** provide tables of standard OMRON products available for the K-types, reference tables of instructions and Programming Console operations, and other information helpful in PC operation.

> ⚠ **WARNING** Failure to read and understand the information provided in this manual may result in personal injury or death, damage to the product, or product failure. Please read each section in its entirety and be sure you understand the information provided in the section and related sections before attempting any of the procedures or operations given.

PRECAUTIONS

This section provides general precautions for using the K-type Programmable Controllers (PCs) and related devices.

The information contained in this section is important for the safe and reliable application of Programmable Controllers. You must read this section and understand the information contained before attempting to set up or operate a PC system.

1 Intended Audience . . . . . .
2 General Precautions . . . . . .
3 Safety Precautions . . . . . .
4 Operating Environment Precautions . . . . . .
5 Application Precautions . . . . . .

1 Intended Audience

This manual is intended for the following personnel, who must also have knowledge of electrical systems (an electrical engineer or the equivalent).

- Personnel in charge of installing FA systems.
- Personnel in charge of designing FA systems.
- Personnel in charge of managing FA systems and facilities.

2 General Precautions

The user must operate the product according to the performance specifications described in the operation manuals.

Before using the product under conditions which are not described in the manual or applying the product to nuclear control systems, railroad systems, aviation systems, vehicles, combustion systems, medical equipment, amusement machines, safety equipment, and other systems, machines, and equipment that may have a serious influence on lives and property if used improperly, consult your OMRON representative.

Make sure that the ratings and performance characteristics of the product are sufficient for the systems, machines, and equipment, and be sure to provide the systems, machines, and equipment with double safety mechanisms.

This manual provides information for programming and operating the Unit. Be sure to read this manual before attempting to use the Unit and keep this manual close at hand for reference during operation.

⚠ WARNING It is extremely important that a PC and all PC Units be used for the specified purpose and under the specified conditions, especially in applications that can directly or indirectly affect human life. You must consult with your OMRON representative before applying a PC System to the above-mentioned applications.

3 Safety Precautions

⚠ WARNING Do not attempt to take any Unit apart while the power is being supplied. Doing so may result in electric shock.

⚠ WARNING Do not touch any of the terminals or terminal blocks while the power is being supplied. Doing so may result in electric shock.

⚠ WARNING Do not attempt to disassemble, repair, or modify any Units. Any attempt to do so may result in malfunction, fire, or electric shock.

4 Operating Environment Precautions

⚠ Caution Do not operate the control system in the following locations:

- Locations subject to direct sunlight.
- Locations subject to temperatures or humidity outside the range specified in the specifications.
- Locations subject to condensation as the result of severe changes in temperature.

- Locations subject to corrosive or flammable gases.
- Locations subject to dust (especially iron dust) or salts.
- Locations subject to exposure to water, oil, or chemicals.
- Locations subject to shock or vibration.

⚠ Caution Take appropriate and sufficient countermeasures when installing systems in the following locations:

- Locations subject to static electricity or other forms of noise.
- Locations subject to strong electromagnetic fields.
- Locations subject to possible exposure to radioactivity.
- Locations close to power supplies.

⚠ Caution The operating environment of the PC System can have a large effect on the longevity and reliability of the system. Improper operating environments can lead to malfunction, failure, and other unforeseeable problems with the PC System. Be sure that the operating environment is within the specified conditions at installation and remains within the specified conditions during the life of the system.

5 Application Precautions

Observe the following precautions when using the PC System.

⚠ WARNING Always heed these precautions. Failure to abide by the following precautions could lead to serious or possibly fatal injury.

- Always ground the system to 100 Ω or less when installing the Units. Not connecting to a ground of 100 Ω or less may result in electric shock.
- Always turn OFF the power supply to the PC before attempting any of the following. Not turning OFF the power supply may result in malfunction or electric shock.
  - Mounting or dismounting I/O Units, CPU Units, Memory Cassettes, or any other Units.
  - Assembling the Units.
  - Setting DIP switches or rotary switches.
  - Connecting cables or wiring the system.
  - Connecting or disconnecting the connectors.

⚠ Caution Failure to abide by the following precautions could lead to faulty operation of the PC or the system, or could damage the PC or PC Units. Always heed these precautions.

- Fail-safe measures must be taken by the customer to ensure safety in the event of incorrect, missing, or abnormal signals caused by broken signal lines, momentary power interruptions, or other causes.
- Interlock circuits, limit circuits, and similar safety measures in external circuits (i.e., not in the Programmable Controller) must be provided by the customer.
- Always use the power supply voltages specified in the operation manuals. An incorrect voltage may result in malfunction or burning.
- Take appropriate measures to ensure that the specified power with the rated voltage and frequency is supplied. Be particularly careful in places where the power supply is unstable. An incorrect power supply may result in malfunction.
- Install external breakers and take other safety measures against short-circuiting in external wiring. Insufficient safety measures against short-circuiting may result in burning.

- Do not apply voltages to the Input Units in excess of the rated input voltage. Excess voltages may result in burning.
- Do not apply voltages or connect loads to the Output Units in excess of the maximum switching capacity. Excess voltages or loads may result in burning.
- Disconnect the functional ground terminal when performing withstand voltage tests. Not disconnecting the functional ground terminal may result in burning.
- Be sure that all the mounting screws, terminal screws, and cable connector screws are tightened to the torque specified in the relevant manuals. Incorrect tightening torque may result in malfunction.
- Leave the label attached to the Unit when wiring. Removing the label may result in malfunction if foreign matter enters the Unit.
- Remove the label after the completion of wiring to ensure proper heat dissipation. Leaving the label attached may result in malfunction.
- Use crimp terminals for wiring. Do not connect bare stranded wires directly to terminals. Connection of bare stranded wires may result in burning.
- Wire all connections correctly.
- Double-check all wiring and switch settings before turning ON the power supply. Incorrect wiring may result in burning.
- Be sure that the terminal blocks, Memory Units, expansion cables, and other items with locking devices are properly locked into place. Improper locking may result in malfunction.
- Check the user program for proper execution before actually running it on the Unit. Not checking the program may result in an unexpected operation.
- Confirm that no adverse effect will occur in the system before attempting any of the following. Not doing so may result in an unexpected operation.
    - Changing the operating mode of the PC.
    - Force-setting/force-resetting any bit in memory.
    - Changing the present value of any word or any set value in memory.
- Resume operation only after transferring to the new CPU Unit the contents of the DM Area, HR Area, and other data required for resuming operation. Not doing so may result in an unexpected operation.
- Do not pull on the cables or bend the cables beyond their natural limit. Doing either of these may break the cables.
- Do not place objects on top of the cables or other wiring lines. Doing so may break the cables.
- When replacing parts, be sure to confirm that the rating of a new part is correct. Not doing so may result in malfunction or burning.
- Before touching a Unit, be sure to first touch a grounded metallic object in order to discharge any static built-up. Not doing so may result in malfunction or damage.
- Install the Units properly as specified in the operation manuals. Improper installation of the Units may result in malfunction.

SECTION 1
Background

1-1 Introduction . . . . . .
1-2 Relay Circuits: The Roots of PC Logic . . . . . .
1-3 PC Terminology . . . . . .
1-4 OMRON Product Terminology . . . . . .
1-5 Overview of PC Operation . . . . . .
1-6 Peripheral Devices . . . . . .
1-7 Available Manuals . . . . . .

1-1 Introduction

A Programmable Controller **(PC)** is basically a central processing unit **(CPU)** containing a program and connected to input and output (I/O) devices **(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 product passing by 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.

Desired control sequences are input to the K-type PCs using a form of PC logic called ladder-diagram programming. This manual is written to explain ladder-diagram programming and to prepare the reader to program and operate the K-type PCs.

1-2 Relay Circuits: The Roots of PC Logic

PCs historically originate in relay-based control systems. And 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 consistency 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, originated as 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 PC terms used for OMRON PCs.

Relay term	PC equivalent
contact	input or condition
coil	output or work bit
NO relay	normally open condition
NC relay	normally closed condition

Actually there is not a total equivalence between these terms, because the term condition is used only to describe ladder diagram programs in general and is specifically equivalent to one of certain basic instructions. The terms input and output are not used in programming per se, except in reference to I/O bits that are assigned to input and output signals coming into and leaving the PC. Normally open conditions and normally closed conditions are explained in *4-3 The Ladder Diagram*.

1-3 PC Terminology

Although also provided in the *Glossary* at the back of this manual, the following terms are crucial to understanding PC operation and are thus explained here as well.

PC

When we refer to the PC, we are generally talking about the CPU and all of the Units directly controlled by it through the program. This does not include the I/O devices connected to PC inputs and outputs.

If you are not familiar with the terms used above to describe a PC, refer to *Section 2 Hardware Considerations* for explanations.

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 and 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 and 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 describing 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 describing the signals that enter or leave the system, reference is made to input signals and output signals, or sometimes just inputs and outputs.

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.

1-4 OMRON Product Terminology

OMRON products are divided into several functional groups that have generic names. *Appendix A Standard Models* list products by these groups. The term **Unit** is used to refer to all OMRON PC products, depending on the context.

The largest group of OMRON products is **I/O Units**. I/O Units come in a variety of point quantities and specifications.

Special I/O Units are dedicated Units that are designed to meet specific needs. These include Analog Timer Units and Analog I/O Units.

Link Units are used to create Link Systems that link more than one PC or link a single PC to remote I/O points. Link Units include I/O Link Units that are used to connect K-type PCs to Remote I/O Systems controlled by a larger PC (e.g. C1000H) and Host Link Units.

Other product groups include **Programming Devices** and **Peripheral Devices**.

1-5 Overview of PC Operation

The following are the basic steps involved in programming and operating a K-type PC. 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. This manual is written to explain steps three through six, eight, and nine. The section(s) of this manual that provide relevant information are listed with each of these steps.

1, 2, 3...

1. Determine what the controlled system must do, in what order, and at what times.
2. Determine what Units will be required. Refer to the *Installation Guide*. If a Link System is required, refer to the required *System Manual(s)*.
3. On paper, assign all input and output devices to I/O points on Units and determine which I/O bits will be allocated to each. If the PC includes Special I/O Units or Link Systems, refer to the individual *Operation Manuals* or *System Manuals* for details on I/O bit allocation. (*Section 3 Memory Areas*)
4. Using relay ladder symbols, write a program that represents the sequence of required operations and their inter-relationships. Be sure to also program appropriate responses for all possible emergency situations. (*Section 4 Writing and Inputting the Program*, *Section 5 Instruction Set*, and *Section 6 Program Execution Timing)*
5. Input the program and all required operating parameters into the PC. (*Section 4 Writing and Inputting the Program*)
6. Debug the program, first to eliminate any syntax errors and then to eliminate execution errors. (*Section 4 Writing and Inputting the Program, Section 7 Program Debugging and Execution*, and *Section 8 Troubleshooting*)
7. Wire the PC to the controlled system. This step can actually be started as soon as step 3 has been completed. Refer to the *Installation Guide* and to *Operation Manuals* and *System Manuals* for details on individual Units.
8. Test the program in an actual control situation and fine tune it if required. (*Section 7 Program Debugging and Execution* and *Section 8 Troubleshooting*)
9. Record two copies of the finished program on masters and store them safely in different locations. (*Section 7 Program Debugging and Execution*)

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 fully understood. Designing the Control System requires a thorough understanding of the system that is to be controlled. The first step in designing a Control System is thus determining the requirements of the controlled system.

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. Keep in mind that the number of I/O points available depends on the configuration of the PC. Refer to *3-3 Internal Relay (IR) Area* for details on I/O capacity and assigning I/O bits to I/O points.

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 five input signals from the photoelectric switch.

Each of the related tasks must be similarly determined, throughout the entire control operation.

Unit Requirements

The actual Units that will be mounted must be determined according to the requirements of the I/O devices. This will include actual hardware specifications, such as voltage and current levels, as well as functional considerations, such as those that require Special I/O Units or Link Systems. In many cases, Special I/O Units or Link Systems can greatly reduce the programming burden. Details on these Units and Link Systems are available in individual *Operation Manuals* and *System Manuals.*

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.

1-6 Peripheral Devices

The following peripheral devices can be used in programming, either to input/debug/monitor the PC program or to interface the PC to external devices to output the program or memory area data. Model numbers for all devices listed below are provided in *Appendix A Standard Models*. OMRON product names have been placed in bold when introduced in the following descriptions.

Programming Console

A Programming Console is the simplest form of programming device for OMRON PCs. Although a **Programming Console Adapter** is sometimes required, all Programming Consoles are connected directly to the CPU without requiring a separate interface. The Programming Console also functions as an interface to output programs to a standard cassette tape recorder.

Various types of Programming Console are available, including both CPU-mounting and Handheld models. Programming Console operations are described later in this manual.

Graphic Programming Console: GPC

A **Peripheral Interface Unit** is required to interface the GPC to the PC.

The GPC also functions as an interface to output programs directly to a standard cassette tape recorder. A **PROM Writer**, **Floppy Disk Interface Unit**, or **Printer Interface Unit** can be directly mounted to the GPC to output programs directly to an EPROM chip, floppy disk drive, or printing device.

Ladder Support Software: LSS

LSS is designed to run on IBM AT/XT compatibles to enable nearly all of the operations available on the GPC. It also offers extensive documentation capabilities.

A **Host Link Unit** is required to interface a computer running LSS to the PC. Using an Optical Host Link Unit also enables the use of optical fiber cable to connect the FIT to the PC. Wired Host Link Units are available when desired. (Although FIT does not have optical connectors, conversion to optical fiber cable is possible by using Converting Link Adapters.)

Factory Intelligent Terminal: FIT

The FIT is an OMRON computer with specially designed software that allows you to perform all of the operations that are available with the GPC or LSS. Programs can also be output directly to an EPROM chip, floppy disk drive, or printing device without any additional interface units. The FIT has an EPROM writer and two 3.5” floppy disk drives built in.

A **Peripheral Interface Unit** or **Host Link Unit** is required to interface the FIT to the PC. Using an Optical Host Link Unit also enables the use of optical fiber cable to connect the FIT to the PC. Wired Host Link Units are available when desired. (Although FIT does not have optical connectors, conversion to optical fiber cable is possible by using Converting Link Adapters.)

PROM Writer

Other than its applications described above, the PROM Writer can be mounted to the PC’s CPU to write programs to EPROM chips.

Floppy Disk Interface Unit

Other than its applications described above, the Floppy Disk Interface Unit can be mounted to the PC’s CPU to interface a floppy disk drive and write programs onto floppy disks.

Printer Interface Unit

Other than its applications described above, the Printer Interface Unit can be mounted to the PC’s CPU to interface a printer or X-Y plotter to print out programs in either mnemonic or ladder-diagram form.

1-7 Available Manuals

The following table lists other manuals that may be required to program and/or operate the K-type PCs. *Operation Manuals* and/or *Operation Guides* are also provided with individual Units and are required for wiring and other specifications.

Name	Cat. No.	Contents
Installation Guide	W147	Hardware specifications
Data Access Console Operation Guide	W173	Procedures for monitoring and manipulating data.
GPC Operation Manual	W84	Programming procedures for the GPC (Graphics Programming Console)
FIT Operation Manual	W150	Programming procedures for using the FIT (Factory Intelligent Terminal
LSS Operation Manual	W237	Programming procedures for using LSS (Ladder Support Software)
Printer Interface Unit Operation Guide	W107	Procedures for interfacing a PC to a printer
PROM Writer Operation Guide	W155	Procedures for writing programs to EPROM chips
Floppy Disk Interface Unit Operation Guide	W119	Procedures for interfacing a PC to a floppy disk drive
Optical Remote I/O System Manual	W136	Information on building an Optical Remote I/O System to enable remote I/O capability
Host Link System Manual	W143	Information on building a Host Link System to manage PCs from a ‘host’ computer
K-type Analog I/O Units Operation Guide	W122	Hardware and software information on using Analog I/O Units with the K-type PCs.

SECTION 2
Hardware Considerations

2-1 Introduction . . . . . .
2-2 Indicators . . . . . .
2-3 PC Configuration . . . . . .

2-1 Introduction

This section provides information on hardware aspects of K-type PCs that are relevant to programming and software operation. These include indicators on the CPU and basic PC configuration. This information is covered in detail in the *Installation Guide*.

2-2 Indicators

CPU indicators provide visual information on the general operation of the PC. Using the flags and other error indicators provided in the memory data areas, although not a substitute for proper error programming, provides ready confirmation of proper operation.

CPU Indicators

CPU indicators are located on the front right hand side of the PC adjacent to the I/O expansion slot and are described in the following table.

Indicator	Function
POWER	Lights when power is supplied to the CPU.
RUN	Lights when the CPU is operating normally.
ERR	Lights when an error is discovered in system error diagnosis operations. When this indicator lights, the RUN indicator will go off, CPU operation will be stopped, and all outputs from the PC will be turned OFF.
ALARM	Lights when an error is discovered in system error diagnosis operations. PC operation will continue.

2-3 PC Configuration

The system must contain a K-type CPU and may additionally contain an Expansion I/O Unit, Special I/O Units and/or I/O Link Units.

The Expansion I/O Units are not a required part of the basic system and are used to increase the number of I/O points available. Special I/O Units and I/O Link Units are used to reduce programming complexity.

SECTION 3
Memory Areas

3-1 Introduction . . . . .
3-2 Data Area Structure . . . . .
3-3 Internal Relay (IR) Area . . . . .
3-4 Special Relay (SR) Area . . . . .
3-4-1 Battery Alarm Flag . . . . .
3-4-2 Cycle Time Error Flag . . . . .
3-4-3 High-speed Drum Counter Reset . . . . .
3-4-4 Clock Pulse Bits . . . . .
3-4-5 Error Flag (ER) . . . . .
3-4-6 Step Flag . . . . .
3-4-7 Always OFF, Always ON Flags . . . . .
3-4-8 First Cycle Flag . . . . .
3-4-9 Arithmetic Flags . . . . .
3-5 Data Memory (DM) Area . . . . .
3-6 Holding Relay (HR) Area . . . . .
3-7 Timer/Counter (TC) Area . . . . .
3-8 Temporary Relay (TR) Area . . . . .

3-1 Introduction

Various types of data are required to achieve effective and correct control. To facilitate managing this data, the PC is provided with various **memory areas** for data, each of which performs a different function. The areas generally accessible by the user for use in programming are classified as **data areas**. The other memory area is the Program Memory, where the user's program is actually stored.

This section describes these areas individually and provides information that will be necessary to use them. The name, acronym, range, and function of each area are summarized in the following table. All but the last one of these are data areas. All memory areas are normally referred to by their acronyms.

Area	Acronym	Range	Function
Internal Relay area	IR	Words: 00 to 18 (bits 00 to 07) Bits: 0000 to 1807	Used to manage I/O points, control other bits, timers, and counters, to temporarily store data.
Special Relay area	SR	Words: 18 (bits 08 to 15) and 19 (bits 00 to 07) Bits: 1808 to 1907	Contains system clocks, flags, control bits, and status information.
Data Memory area	DM	DM 00 to DM 63 (words only)	Used for internal data storage and manipulation.
Holding Relay area	HR	Words: HR 0 to HR 9 Bits: HR 000 to HR 915	Used to store data and to retain the data values when the power to the PC is turned off.
Timer/Counter area	TC	TC 00 to TC 47 (TC numbers are used to access other information)	Used to define timers and counters and to access completion flags, PV, and SV for them.
Temporary Relay area	TR	TR 00 to TR 07 (bits only)	Used to temporarily store execution conditions.
Program Memory	UM	UM: 1,194 words.	Contains the program executed by the CPU.

Work Bits and Words

When some bits and words in certain data areas are not used for their intended purpose, they can be used in programming as required to control other bits. Words and bits available for use in this fashion are called work bits and work words. Most, but not all, unused bits can be used as work bits. Those that can be are specified by area in the remainder of this section. Actual application of work bits and work words is described in *Section 4 Writing and Inputting the Program*.

Flags and Control Bits

Some data areas contain flags and/or control bits. Flags are bits that are automatically turned ON and OFF to indicate status of one form or another. Although some flags can be turned ON and OFF by the user, most flags can be read only; they cannot be controlled directly.

Control bits are bits turned ON and OFF by the user to control specific aspects of operation. Any bit given a name using the word bit rather than the word flag is a control bit, e.g., Restart Bits are control bits.

3-2 Data Area Structure

When designating a data area, the acronym for the area is always required for any but the IR and SR areas. Although the acronyms for the IR and SR areas are often given for clarity, they are not required and not input when programming. Any data area designation without an acronym is assumed to be in either the IR and SR area. Because IR and SR addresses run consecutively, the word or bit addresses are sufficient to differentiate these two areas.

An actual data location within any data area but the TC area is designated by its address. The address designates the bit and/or word within the area where the desired data is located. The TR area consists of individual bits

used to store execution conditions at branching points in ladder diagrams. The use of TR bits is described in *Section 4 Writing and Inputting the Program.* The TC area consists of TC numbers, each of which is used for a specific timer or counter defined in the program. Refer to *3-7 Timer/Counter (TC) Area* for more details on TC numbers and to *5-11 Timer and Counter Instructions* for information on actual application.

The rest of the data areas (i.e., the IR, SR, HR and DM areas) consist of words, each of which consists of 16 bits numbered 00 through 15 from right to left. IR words 00 and 01 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.

Bit number	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IR word 00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IR word 01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note The term least significant is often used for rightmost; the term most significant, for leftmost. These terms have not been used in this manual because a single word is often split into two or more parts, with each part used for different parameters or operands, sometimes even with bits in another word. When this is done, the rightmost bits in a word may actually be the most significant bits, i.e., the leftmost bits, of a value with other bits, i.e., the least significant bits, contained in another word.

The DM area is accessible by word only; you cannot designate an individual bit within a DM word. Data in the IR, SR and HR areas is accessible either by bit or by word, depending on the instruction in which the data is being used.

To designate one of these areas by word, all that is necessary is the acronym (if required) and the one or two-digit word address. To designate an area by bit, the word address is combined with the bit number as a single three- or four-digit address. The examples in the following table should make this clear. The two rightmost digits of a bit designation must indicate a bit between 00 and 15.

The same TC number can be used to designate either a word containing the present value (PV) of the timer or counter or a bit that functions as the completion flag for the timer or counter. This is explained in more detail in *3-7 Timer/Counter (TC) Area*.

Area	Word designation	Bit designation
IR	00	0015 (leftmost bit in word 00)
SR	19	1900 (rightmost bit in word 19)
DM	DM 10	Not possible
TC	TC 46 (designates PV)	TC 46 (designates completion flag)

Data Structure

Word data input as decimal values is stored in binary-coded decimal (BCD) code; word data input as hexadecimal is stored in binary form. Because each word contains 16 bits, each four bits of a word represents one digit: either a hexadecimal digit equivalent numerically to the binary bits or decimal. 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. *Section 5 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 x 5 + 16^2 x 15 + 16 x 5 + 15).

Decimal and BCD can also be easily converted back and forth. In this case, each BCD digit (i.e., each 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 x 5 + 16^2 x 7 + 16 x 5 + 7).

Because the numeric equivalent of each four BCD binary bits must be 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 and they are equivalent to the hexadecimal digit C.

There are instructions provided to convert data in either direction between BCD and hexadecimal. Refer to *5-15 Data Conversion* for details. Tables of binary equivalents to hexadecimal and BCD digits are provided in the appendices for reference.

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.

3-3 Internal Relay (IR) Area

The IR area is used both to control I/O points and as work bits to manipulate and store data internally. It is accessible both by bit and by word. Those words that can be used to control I/O points are called I/O words. Bits in I/O words are called I/O bits.

The number of I/O words varies between the K-type PCs. As shown, the IR area is comprised of three main sections. These are input words, output words and work words (work bits). Work bits are used in programming to manipulate data and control other bits. IR area work bits are reset when power is interrupted or PC operation is stopped.

I/O Words

The maximum number of available I/O bits is 16 (bits/word) times the number of I/O words. I/O bits are assigned to input or output points as described in *Word Allocations*.

If a Unit brings inputs into the PC, the bit assigned to it is an input bit; if the Unit sends an output from the PC, the bit is an output bit. To turn on an output, the output bit assigned to it must be turned ON. When an input turns on, the input bit assigned to it also turns ON. These facts can be used in the program to access input status and control output status through I/O bits.

I/O bits that are not assigned to I/O points can be used as work bits, unless otherwise specified in *Word Allocations*.

Input Bit Usage

Input bits can directly input external signals to the PC and can be used in any order in programming. Each input bit can also be used in as many instructions as required to achieve effective and proper control. They cannot be used in instructions that control bit status, e.g., the OUTPUT, DIFFERENTIATION UP, and KEEP instructions.

Output Bit Usage

Output bits are used to output program execution results and can be used in any order in programming. Because outputs are refreshed only once during each cycle (i.e. once each time the program is executed), any output bit can be used in only one instruction that controls its status, including OUT, OUT NOT, KEEP(11), DIFU(13), DIFD(14), and SFT(10). If an output bit is used in more than one such instruction, only the status determined by the last instruction will actually be output from the PC. See *5-12-1 SHIFT REGISTER - SFT(10)* for an example of an output bit controlled by two instructions.

Word Allocations

The maximum number of words available for I/O within the IR area is 10, numbered 00 through 09. The remaining words (10 through 18) are to be used for work bits. (Note that with word 18, only the bits 00 through 07 are available for work bits although some of the remaining bits are required for special purposes when RDM is used).

The actual number of bits that can be used as I/O bits is determined by the model of the CPU and the PC configuration. There are different models of Expansion I/O Units and Special I/O Units and I/O Link Units which can be connected to any of the CPUs. Each CPU model provides a particular number of I/O bits and each Expansion I/O Unit, Special I/O Unit or I/O Link Unit provides a particular number of I/O bits. Configuration charts for the possible combinations of CPUs and Units are included later in this section. Refer to those to determine the actual available I/O bits.

Within CPUs the I/O input words are always even numbered and the output words are always odd numbered. The general rule when connecting Expansion I/O Units to CPUs is that the first available word for the Expansion I/O Unit (whether input or output or a combination) is one more than the last I/O word of the CPU. If the Expansion I/O Unit is only either input or output (and not both) then the I/O words provided by the Expansion I/O Unit are allocated consecutively and the remaining words up to word 09 may be used for work bits. If the Expansion I/O Unit provides both input and output words then the words are allocated alternatively (input words always having even numbers) until all I/O words provided by the Expansion I/O Unit are allocated. The remaining words up to word 09 may then be used for work bits. Note that when a portion of an input word is not allocated to an input point then that portion may be used for work bits.

I/O Bits Available in CPUs

The following table shows which bits can be used as I/O bits in each of the K-type CPUs. Bits in the shaded areas can be used as work bits but not as output bits.

Model	Input bits				Output bits			
C20K	Word 00				Word 01			
	00	08			00	08		
	01	09			01	09		
	02	10			02	10		
	03	11			03	11		
	04	Cannot be used.			04	12		
	05				05	13		
	06				06	14		
	07				07	15		
C28K	Word 00				Word 01			
	00	08			00	08		
	01	09			01	09		
	02	10			02	10		
	03	11			03	11		
	04	12			04	12		
	05	13			05	13		
	06	14			06	14		
	07	15			07	15		
C40K	Word 00		Word 02		Word 01		Word 03	
	00	08	00	Cannot be used.	00	08	00	08
	01	09	01		01	09	01	09
	02	10	02		02	10	02	10
	03	11	03		03	11	03	11
	04	12	04		04	12	04	12
	05	13	05		05	13	05	13
	06	14	06		06	14	06	14
	07	15	07		07	15	07	15
C60K	Word 00		Word 02		Word 01		Word 03	
	00	08	00	08	00	08	00	08
	01	09	01	09	01	09	01	09
	02	10	02	10	02	10	02	10
	03	11	03	11	03	11	03	11
	04	12	04	12	04	12	04	12
	05	13	05	13	05	13	05	13
	06	14	06	14	06	14	06	14
	07	15	07	15	07	15	07	15

indicates words that cannot be used for I/O, but can be used as work bits.

I/O Bits Available in Expansion I/O Units

The following table shows which bits can be used as I/O bits in each of the Expansion I/O Units. Bits in the shaded areas can be used as work bits but not as output bits. The word addresses depend on the CPU that the Expansion I/O Unit is coupled to. In all cases the first Expansion I/O Unit address for input and output words is one more than the last CPU address for input and output words. For example, the last CPU word address for a C40K CPU is 03 and hence the first input or output word address for any of the Expansion I/O Units coupled to a C40K CPU will be 04. In the tables below "n" is the last CPU word allocated as an input or output word.

There are several models for some of the Units listed below. A blank space (_) in the model number indicates that any of the applicable model numbers could be inserted here.

C20P

Input bits: Word (n+1)		Output bits: Word (n + 2)	
00	08	00	08
01	09	01	09
02	10	02	10
03	11	03	11
04	Cannot be used.	04	12
05		05	13
06		06	14
07		07	15

C28P

Input bits: Word (n+1)		Output bits: Word (n + 2)	
00	08	00	08
01	09	01	09
02	10	02	10
03	11	03	11
04	12	04	12
05	13	05	13
06	14	06	14
07	15	07	15

C40P

Input bits: Word (n+1)		Word (n + 3)		Output bits: Word (n + 2)		Word (n + 4)	
00	08	00	Cannot be used.	00	08	00	08
01	09	01		01	09	01	09
02	10	02		02	10	02	10
03	11	03		03	11	03	11
04	12	04		04	12	04	12
05	13	05		05	13	05	13
06	14	06		06	14	06	14
07	15	07		07	15	07	15

C60P

Input bits: Word (n+1)		Word (n + 3)		Output bits: Word (n + 2)		Word (n + 4)	
00	08	00	08	00	08	00	08
01	09	01	09	01	09	01	09
02	10	02	10	02	10	02	10
03	11	03	11	03	11	03	11
04	12	04	12	04	12	04	12
05	13	05	13	05	13	05	13
06	14	06	14	06	14	06	14
07	15	07	15	07	15	07	15

indicates words that cannot be used for I/O, but can be used as work bits.

C16P -I□-□

Input bits: Word (n + 1)		Output bits
00	08	
01	09	
02	10	
03	11	
04	12	
05	13	
06	14	
07	15	

C16P -O□-□

Output bits: Word (n + 1)	
00	08
01	09
02	10
03	11
04	12
05	13
06	14
07	15

C4K-I□

Input bits: Word (n + 1)
00
01
02
03
Cannot be used

C4K-O□

Output bits: Word (n + 1)	
00	08
01	09
02	10
03	11
04	12
05	13
06	14
07	15

C4K-TM

Input bits: Word (n + 1)	Output bits: Word (n + 2)	
00	00	08
01	01	09
02	02	10
03	03	11
Cannot be used	04	12
	05	13
	06	14
	07	15

PC Configuration

A K-type PC can be configured with a CPU Unit and one or more of the following Units: Expansion I/O Units, Analog Timer Units, or an I/O Link Unit. All of these Units are connected in series with the CPU Unit at one end. An I/O Link Unit, if included, must be on the other end (meaning only one I/O Link Unit can be used) and an Analog Timer Unit cannot be used. The rest of the Units can be in any order desired.

There is also a restriction in the number of Units which can be included. To compute the number of Units for this restriction, add up all of the Units counting the C40K CPU Unit, C60K CPU Unit, C40K Expansion I/O Unit and C60K Expansion I/O Unit as two Units each and any other Units as one Unit each. This total must be no more than five.

The following table shows some of the combinations that can be used to achieve specific numbers of I/O points. The numbers in the table indicate the number of Units of that size to be used as either the CPU or Expansion I/O Unit; any one of the Units can be the CPU Unit. This table does not include the C4P or C16P Expansion I/O Units, the Analog Timer Unit, or the I/O Link Unit, which can be used for greater system versatility or special applications. Refer to the remaining tables in this section for other combinations.

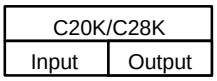
I/O points			Count as 2 each		Count as 1 each	
Total	In	Out	C60□ (32/28)	C40□ (24/16)	C28□ (16/12)	C20□ (12/8)
20	12	8	---	---	---	1
28	16	12	---	---	1	---
40	24	16	---	---	---	2
			---	1	---	---
48	28	20	---	---	1	1
56	32	24	---	---	2	---
60	32	28	1	---	---	---
	36	24	---	---	---	3
			---	1	---	1
68	40	28	---	---	1	2
			---	1	1	---
76	44	32	---	---	2	1
80	48	32	---	---	---	4
			---	1	---	2
			---	2	---	---
			1	---	---	2
84	48	36	---	---	3	---
88	48	40	1	---	1	---
	52	36	---	---	1	3
			---	1	1	1
96	56	40	---	---	2	2
			---	1	2	---
100	56	44	1	---	---	2
			1	1	---	---

I/O points			Count as 2 each		Count as 1 each	
Total	In	Out	C60□ (32/28)	C40□ (24/16)	C28□ (16/12)	C20□ (12/8)
100	60	40	---	---	---	5
			---	1	---	3
			---	2	---	1
104	60	44	---	---	3	1
108	60	48	1	---	1	1
	64	44	---	---	1	4
			---	1	1	2
			---	2	1	---
112	64	48	---	---	4	---
116	64	52	1	---	2	---
	68	48	---	---	2	3
			---	1	2	1
120	64	56	2	---	---	---
	68	52	1	---	---	3
			1	1	---	1
124	72	52	---	---	3	2
			---	1	3	---
128	72	56	1	---	1	2
			1	1	1	---
132	76	56	---	---	4	1
136	76	60	1	---	2	1
140	76	64	2	---	---	1
	80	60	---	---	5	---
144	80	64	1	---	3	---
148	80	68	2	---	1	---

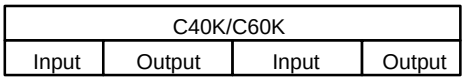
The tables on the following pages show the possible configurations for a K-type PC. Although the tables branch to show the various possibilities at any one point, there can be no branching in the actual PC connections. You can choose either branch at any point and go as far as required, i.e., you can break off at any point to create a smaller PC System.When implementing a system there is a physical restriction on the total cable length allowable. The sum of the lengths of all cables in the system must be limited to less than 1.2 meters.

The tables also show which words will be input words and which words will be output words. All of these are determined by the position of the Unit in the configuration except for the C4P and C16P Expansion I/O Units, in which case the model of the Unit determines whether the words are input or output.

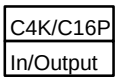
The symbols used in the table represent the following:



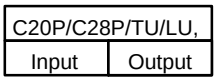
C20K or C28K CPU Unit



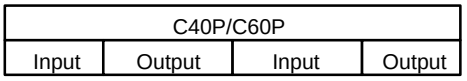
C40K or C60K CPU Unit



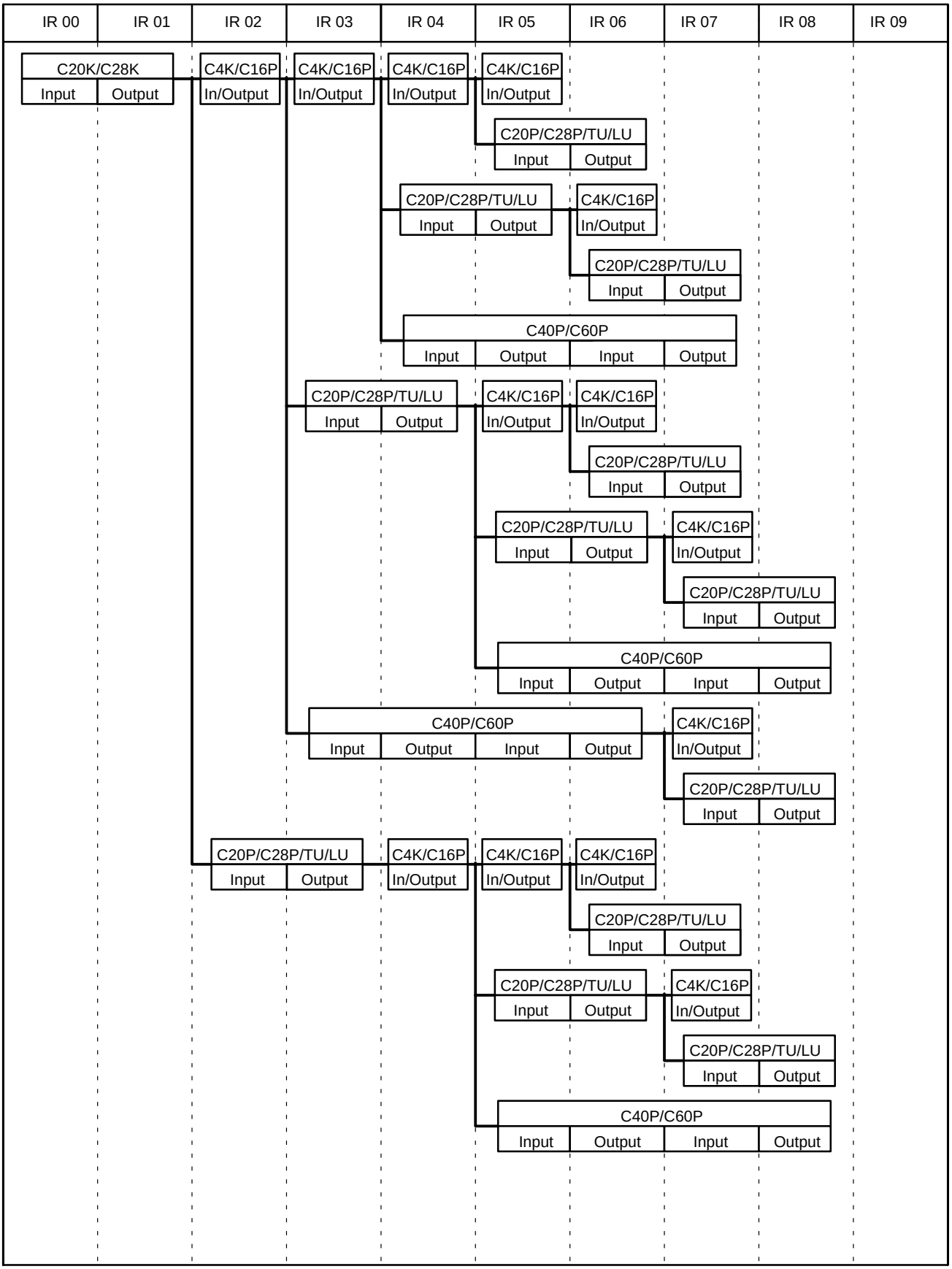
C4P or C16P Expansion I/O Unit

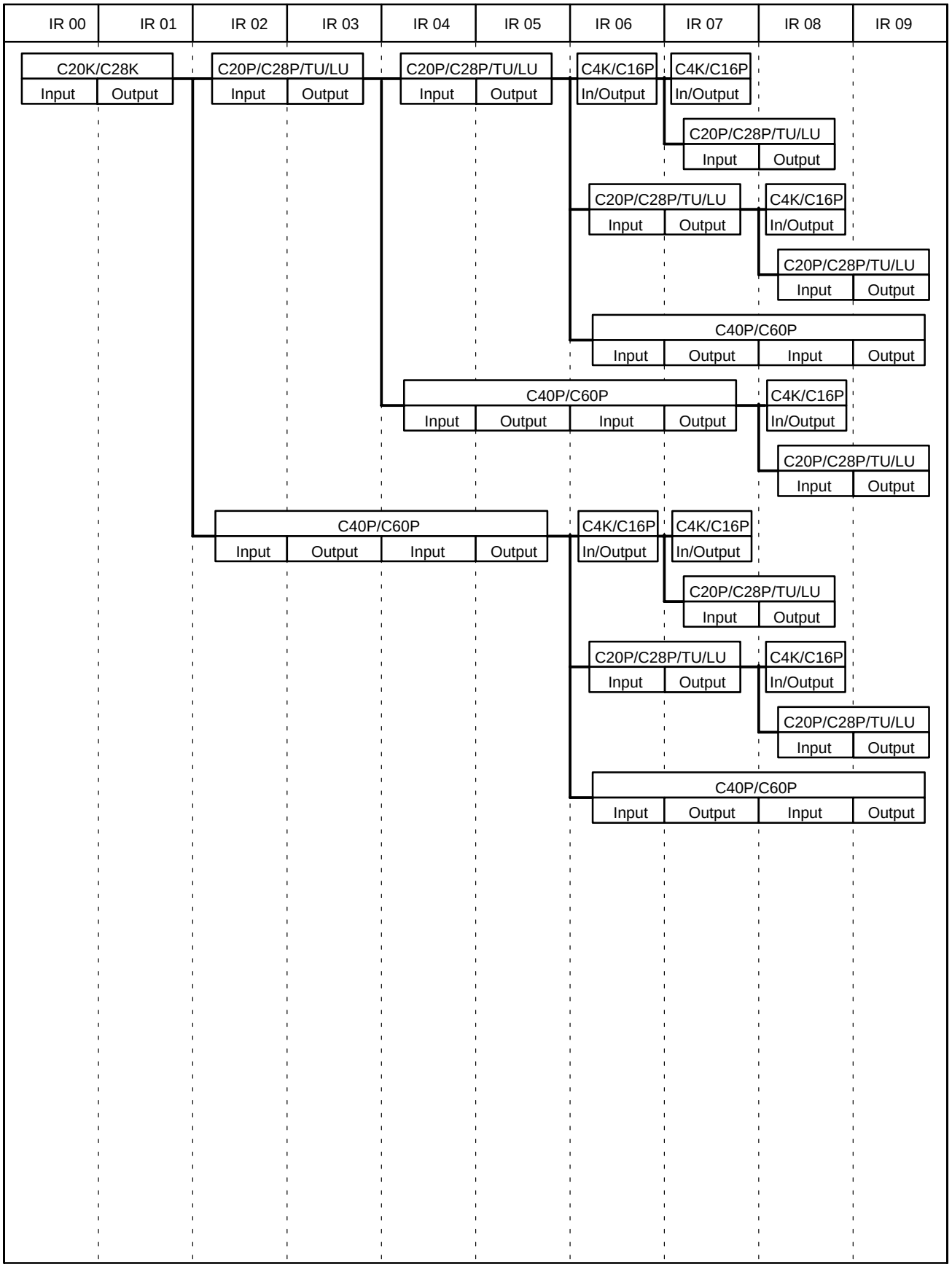


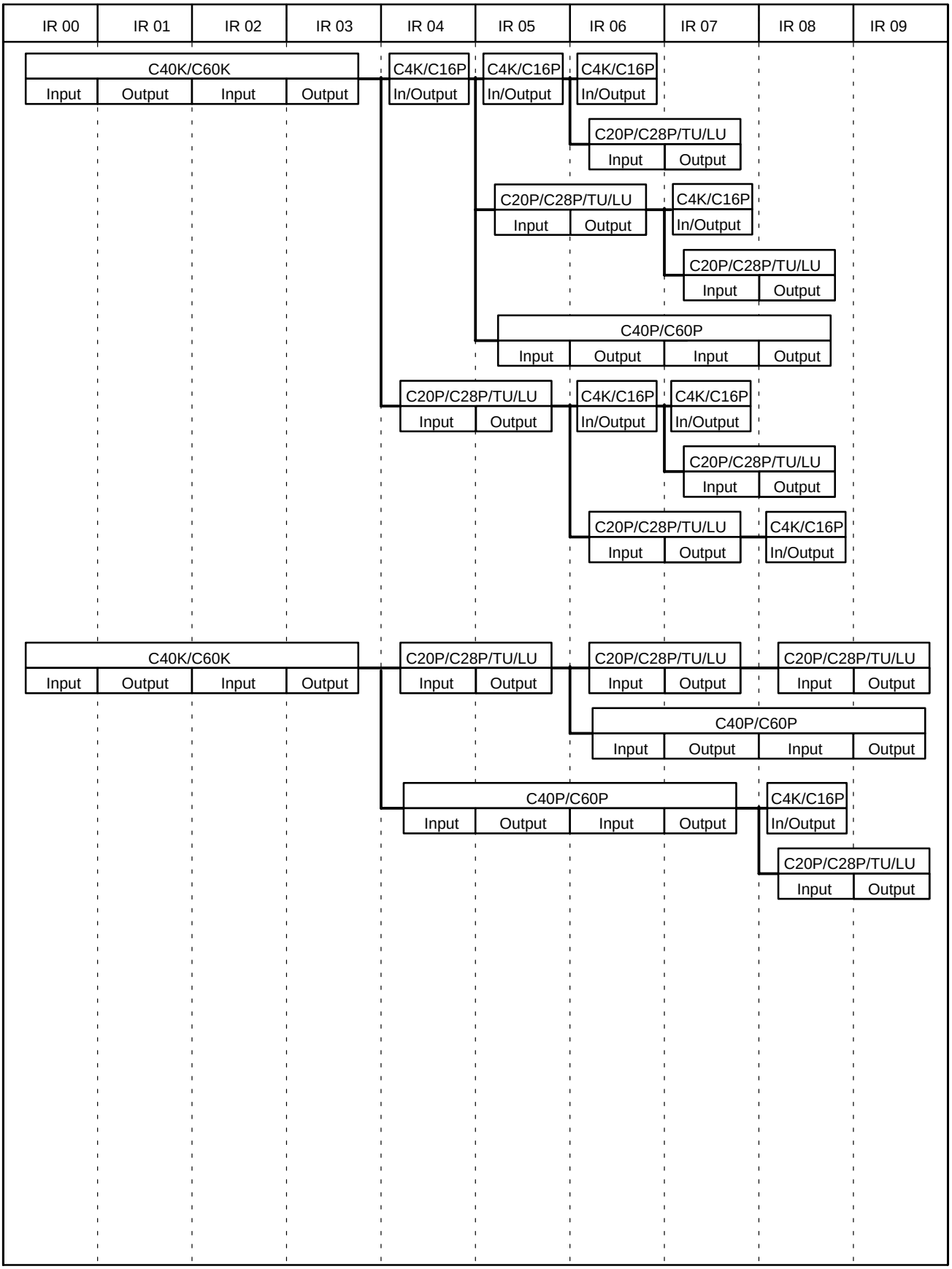
C20P Expansion I/O Unit, C28K Expansion I/O Unit, Analog Timer Unit, or I/O Link Unit



C40P or C60P Expansion I/O Unit


IR 00
IR 01
IR 02
IR 03
IR 04
IR 05
IR 06
IR 07
IR 08
IR 09
C20K/C28K
Input
Output
C4K/C16P
In/Output
C20P/C28P/TU/LU
C40P/C60P


IR 00
IR 01
IR 02
IR 03
IR 04
IR 05
IR 06
IR 07
IR 08
IR 09
C20K/C28K
Input
Output
C20P/C28P/TU/LU
Input
Output
C20P/C28P/TU/LU
Input
Output
C4K/C16P
In/Output
C4K/C16P
In/Output
C20P/C28P/TU/LU
Input
Output
C20P/C28P/TU/LU
Input
Output
C4K/C16P
In/Output
C20P/C28P/TU/LU
Input
Output
C40P/C60P
Input
Output
Input
Output
C40P/C60P
Input
Output
Input
Output
C4K/C16P
In/Output
C20P/C28P/TU/LU
Input
Output
C40P/C60P
Input
Output
Input
Output
C4K/C16P
In/Output
C4K/C16P
In/Output
C20P/C28P/TU/LU
Input
Output
C20P/C28P/TU/LU
Input
Output
C4K/C16P
In/Output
C20P/C28P/TU/LU
Input
Output
C40P/C60P
Input
Output
Input
Output


IR 00
IR 01
IR 02
IR 03
IR 04
IR 05
IR 06
IR 07
IR 08
IR 09
C40K/C60K
Input
Output
Input
Output
C4K/C16P
In/Output
C4K/C16P
In/Output
C4K/C16P
In/Output
C20P/C28P/TU/LU
Input
Output
C20P/C28P/TU/LU
Input
Output
C4K/C16P
In/Output
C20P/C28P/TU/LU
Input
Output
C40P/C60P
Input
Output
Input
Output
C20P/C28P/TU/LU
Input
Output
C4K/C16P
In/Output
C4K/C16P
In/Output
C20P/C28P/TU/LU
Input
Output
C20P/C28P/TU/LU
Input
Output
C4K/C16P
In/Output
C40K/C60K
Input
Output
Input
Output
C20P/C28P/TU/LU
Input
Output
C20P/C28P/TU/LU
Input
Output
C20P/C28P/TU/LU
Input
Output
C40P/C60P
Input
Output
Input
Output
C40P/C60P
Input
Output
Input
Output
C4K/C16P
In/Output
C20P/C28P/TU/LU
Input
Output

3-4 Special Relay (SR) Area

The SR area contains flags and control bits used for monitoring system operation, accessing clock pulses, and signalling errors. SR area word addresses range from 18 through 19; bit addresses, from 1804 through 1907.

The following table lists the functions of SR area flags and control bits. 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 1903 to 1907 are turned OFF when END is executed at the end of each program cycle, and thus cannot be monitored on the Programming Console. Other control bits are OFF until set by the user.

Word	Bit	Function
18	04	RDM(60) Reset Bit
	05	RDM(60) Count Input Bit
	06	RDM(60) Up/Down Selection Bit
	07	HDM(61) Reset Bit
	08	Battery Alarm flag
	09	Cycle Time Error flag
	10	High Speed Counter Reset
	11	Step flag
	12	Always OFF flag
	13	Always ON flag
	14	Always OFF flag
	15	First Cycle flag
19	00	0.1-second Clock Pulse
	01	0.2-second Clock Pulse
	02	1-second Clock Pulse
	03	Error (ER) flag
	04	Carry (CY) flag
	05	Greater Than (GR) flag
	06	Equals (EQ) flag
	07	Less Than (LE) flag

3-4-1 Battery Alarm Flag

SR bit 1808 turns ON if the voltage of the CPU backup battery drops. A voltage drop can be indicated by connecting the output of this bit to an external indicating device such as a LED. This bit can be used in programming to activate an external warning for a low battery.

3-4-2 Cycle Time Error Flag

SR bit 1809 turns ON if the cycle time exceeds 100 ms. This bit is turned ON when the cycle time is between 100 and 130 ms. The PC will still operate but timing may become inaccurate. The PC will stop operating if the execution time exceeds 130 ms.

3-4-3 High-speed Drum Counter Reset

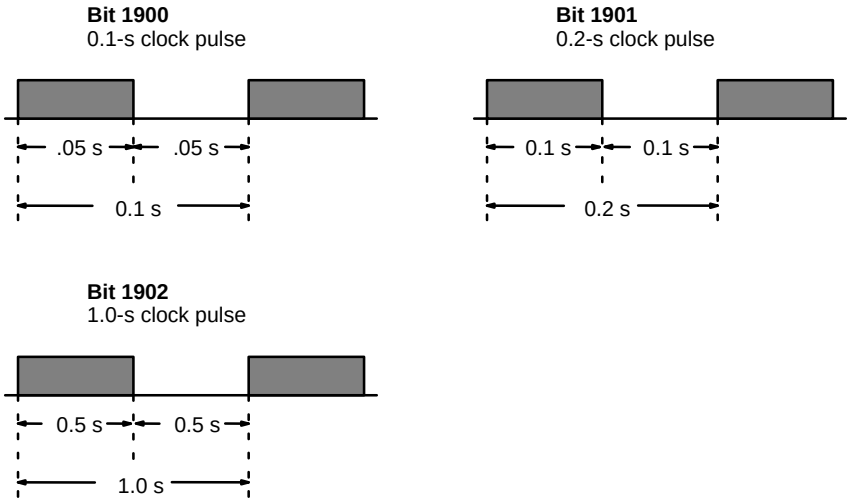
SR bit 1810 turns ON for one cycle time when the hard reset signal (input 0001) is turned ON.

3-4-4 Clock Pulse Bits

Three clock pulses are available to control program timing. Each clock pulse bit is ON for the first half of the rated pulse time, then OFF for the second half. In other words, each clock pulse has a duty factor of 50%.

These clock pulse bits are often used with counter instructions to create timers. Refer to *5-11 Timer and Counter Instructions* for an example of this.

Pulse width	0.1 s	0.2 s	1.0 s
Bit	1900	1901	1902



Caution Because the 0.1-second clock pulse bit has an ON time of 50 ms, the CPU may not be able to accurately read the pulses if program execution time is too long.

3-4-5 Error Flag (ER)

SR bit 1903 turns ON when the results of an arithmetic operation is not output in BCD or the value of the BIN data processed by the BIN to BCD or BCD to BIN conversion instruction exceeds 9999. When the ER flag is ON the current instruction is not executed.

3-4-6 Step Flag

SR bit 1811 turns ON for one cycle when single-step execution is started with the STEP instruction.

3-4-7 Always OFF, Always ON Flags

SR bits 1812 and 1814 are always OFF and 1813 is always ON. By connecting these bits to external indicating devices such as a LED they can be used to monitor the PC's operating status.

3-4-8 First Cycle Flag

SR bit 1815 turns ON when program execution starts and turns OFF after one cycle.

3-4-9 Arithmetic Flags

The following flags are used in data shifting, arithmetic calculation, and comparison instructions. They are generally referred to only by their two-letter abbreviations. Refer to *5-12 Data Shifting*, *5-14 DATA COMPARE - CMP(20)* and *5-16 BCD Calculations* for details.

Caution These flags are all reset when END is executed, and therefore cannot be monitored from a Programming Device.

Carry Flag, CY SR bit 1904 turns ON when there is a carry in the result of an arithmetic operation. The content of CY is also used in some arithmetic operations, e.g., it is added or subtracted along with other operands. This flag can be set and cleared from the program using the SET CARRY and CLEAR CARRY instructions. Use CLC before any instruction using CY unless the current content of CY is required.

Greater Than Flag, GR SR bit 1905 turns ON when the result of a comparison shows the second of two 4-digit operands to be greater than the first.

Equal Flag, EQ SR bit 1906 turns ON when the result of a comparison shows two operands to be equal or when the result of an arithmetic operation is zero.

Less Than Flag, LE SR bit 1907 turns ON when the result of a comparison shows the second of two 4-digit operands to be less than the first.

Note Remember that the previous four flags, CY, GR, EQ, and LE, are cleared by the END instruction.

3-5 Data Memory (DM) Area

The DM area is used for internal data storage and manipulation and is accessible only by word. Addresses range from DM 00 through DM 63.

Although composed of 16 bits just like any other word in memory, DM words cannot be specified by bit for use in instructions with bit-size operands, such as LD, OUT, AND, and OR.

When the RDM (REVERSIBLE DRUM COUNTER) is used the DM area words 00 to 31 are used as the area where the upper and lower limits of the counter are preset and as such these words cannot be used for any other purposes.

When the HDM (HIGH-SPEED DRUM COUNTER) is used the DM area words 32 to 63 are used as the area where the upper and lower limits of the counter are preset and as such these words cannot be used for any other purposes.

The DM area retains status during power interruptions.

3-6 Holding Relay (HR) Area

The HR area is used to store and manipulate various kinds of data and can be accessed either by word or by bit. Word addresses range from HR 0 through HR 9; bit addresses, from HR 000 through HR 915. HR bits can be used in any order required and can be programmed as often as required.

The HR area retains status when the system operating mode is changed, or when power is interrupted.

3-7 Timer/Counter (TC) 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 47. Each TC number is defined as either a timer or counter using one of the following instructions: TIM, TIMH, CNT or CNTR. No prefix is required when using a TC number 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 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 during the program check. There are no restrictions on the order in which TC numbers can be used.

Once defined, a TC number can be designated as an operand in one or more instructions other than those listed above. 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.

TC numbers are also used to access the SV of timers and counters from a Programming Device. The procedures for doing so from the Programming Console are provided in *7-3 Monitoring Operation and Modifying Data.*

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 *5-7 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 00” 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. In explanations of ladder diagrams, the completion flag and PV accessed through a TC number are generally called the completion flag or the PV of the instruction (e.g., the completion flag of TIM 00 is the completion flag accessed through TC number 00, which has been defined using TIM).

When the RDM (REVERSIBLE DRUM COUNTER) is used, TC 46 is used as the present value storage area of the counter and thus cannot be used for any other purpose.

When the HDM (HIGH-SPEED DRUM COUNTER) is used, TC 47 is used as the present value storage area of the counter and thus cannot be used for any other purpose.

3-8 Temporary Relay (TR) Area

The TR area provides eight bits that are used only with the LD and OUT instructions to enable certain types of branching ladder diagram programming. The use of TR bits is described in *Section 4 Writing and Inputting the Program.*

TR addresses range from TR 0 though TR 7. Each of these bits can be used as many times as required and in any order required as long as the same TR bit is not used twice in the same instruction block.

SECTION 4
Writing and Inputting the Program

4-1 Introduction
4-2 Instruction Terminology
4-3 The Ladder Diagram
- 4-3-1 Basic Terms
- 4-3-2 Mnemonic Code
- 4-3-3 Ladder Instructions
- 4-3-4 OUT and OUT NOT
- 4-3-5 The END Instruction
- 4-3-6 Logic Block Instructions
- 4-3-7 Coding Multiple Right-hand Instructions
- 4-3-8 Branching Instruction Lines
- 4-3-9 Jumps

4-4 The Programming Console
- 4-4-1 The Keyboard
- 4-4-2 PC Modes

4-5 Preparation for Operation
- 4-5-1 Entering the Password
- 4-5-2 Clearing Memory
- 4-5-3 Clearing Error Messages

4-6 Inputting, Modifying, and Checking the Program
- 4-6-1 Setting and Reading from Program Memory Address
- 4-6-2 Inputting or Overwriting Programs
- 4-6-3 Checking the Program
- 4-6-4 Displaying the Cycle Time
- 4-6-5 Program Searches
- 4-6-6 Inserting and Deleting Instructions

4-7 Controlling Bit Status
- 4-7-1 DIFFERENTIATE UP and DIFFERENTIATE DOWN
- 4-7-2 KEEP
- 4-7-3 Self-maintaining Bits (Seal)

4-8 Work Bits (Internal Relays)
4-9 Programming Precautions
4-10 Program Execution

4-1 Introduction

This section explains how to convert ladder diagrams to mnemonic code and input them into the PC. It then describes the basic steps and concepts involved in programming and introduces the instructions used to build the basic structure of the ladder diagram and control its execution. The entire set of instructions used in programming is described in *Section 5 Instruction Set*.

There are several basic steps involved in writing a program.

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. If the PC has any Units, i.e. Analog Timer Units, Host Link Units , and I/O Link Units that are allocated words in data areas other than the IR area or are allocated IR words in which the function of each bit is specified by the Unit, prepare similar tables to show what words are used for which Units and what function is served by each bit within the words.
3. Determine what words are available for work bits and prepare a table in which you can allocate these as you use them.
4. Also prepare tables of TC numbers and jump 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; jump numbers 01 through 08 can be used only once each. (TC numbers are described in *5-11 Timer and Counter Instructions*; jump numbers are described later in this section.)
5. Draw the ladder diagram.
6. Input the program into the CPU. When using the Programming Console, this will involve converting the program to mnemonic form.
7. Check the program for syntax errors and correct these.
8. Execute the program to check for execution errors and correct these.
9. After the entire Control System has been installed and is ready for use, execute the program and fine tune it if required.

The basics of writing the ladder diagram and inputting it into memory are described in the rest of this section. Debugging and monitoring operation of the program are described in *Section 7 Program Debugging and Execution*. *Section 8 Troubleshooting* also provides information required for debugging.

This section provides the procedures for inputting and debugging a program and monitoring and controlling the PC through a Programming Console. The Programming Console is the most commonly used Programming Device for the K-type PCs. It is compact and available both in handheld models or CPU-mounted models. Refer to *Appendix A Standard Models* for model numbers and other details.

If you are using a GPC, FIT, or a computer running LSS, refer to the *Operation Manual* for corresponding procedures on these.

4-2 Instruction Terminology

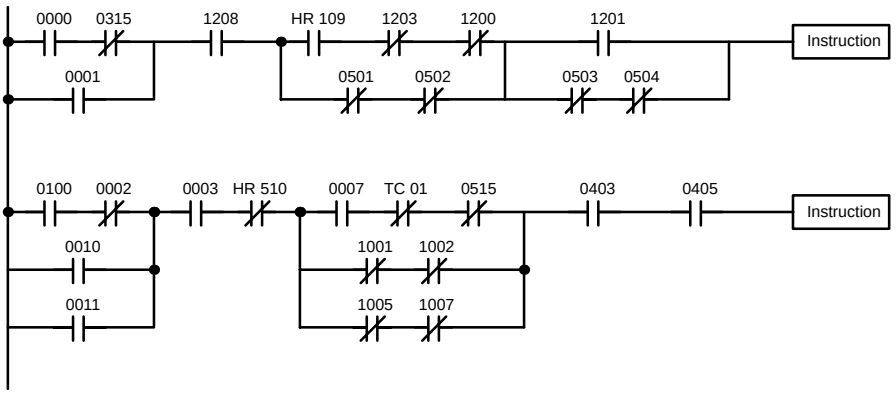
There are basically two types of instructions used in ladder diagram programming: instructions that correspond to conditions on the ladder diagram and are used in instruction form only when converting a program to mnemonic code and instructions that are used on the right side of the ladder diagram and are executed according to the conditions on the instruction lines leading to them.

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 data area words or bits that contain the data to be used. For instance, a MOVE instruction that has IR 00 designated as the source operand will move the contents of IR 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 word.

Other terms used in describing instructions are introduced in *Section 5 Instruction Set*.

4-3 The 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. Along the instruction lines are placed conditions that lead to other instructions on the right side. The logical combinations of these conditions determine when and how the instructions at the right are executed. A simple ladder diagram is shown below.



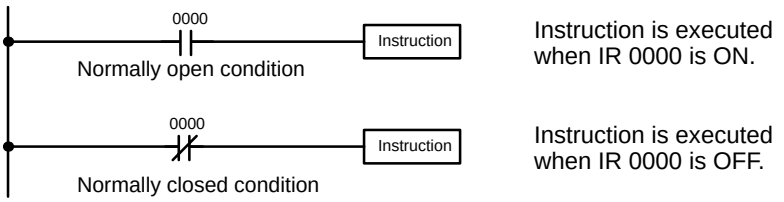
As shown in the diagram above, instruction lines can branch apart and they can join back together. The vertical pairs of lines are called conditions. Conditions without diagonal lines through them are called normally open conditions and correspond to a LOAD, AND, or OR instruction. The conditions with 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 instruction. It is the status of the bit associated with each condition that determine the execution condition for following instructions. The function of each of the instructions that correspond to a condition is described below. Before we consider these, however, there are some basic terms that must be explained.

Note When displaying ladder diagrams with a GPC, a FIT, or LSS, a second bus bar will be shown on the right side of the ladder diagram and will be connected to all instructions on the right side. This does not change the ladder diagram program in any functional sense. No conditions can be placed between the instructions on the right side and the right bus bar, i.e., all instructions on the right must be connected directly to the right bus bar. Refer to the *GPC*, *FIT*, or *LSS Operation Manual* for details.

4-3-1 Basic Terms

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. An 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 an 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 except for LOAD instructions have execution conditions.

Operand Bits

The operands designated for any of the ladder instructions can be any bit in the IR, SR, HR or TC area. This means that the conditions in a ladder diagram can be determined by I/O bits, flags, work bits, timers/counters, etc. LOAD and OUTPUT instructions can also use TR area bits, but they do so only in special applications.

Logic Blocks

What conditions correspond to what instructions is determined by the relationship between the conditions established by 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.

4-3-2 Mnemonic Code

The ladder diagram cannot be directly input into the PC via a Programming Console; a GPC, a FIT, or LSS is required. 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, regardless of the Programming Device used, the program is stored in memory in mnemonic form, making it important to understand mnemonic code.

Because of the importance of the Programming Console as a peripheral device and because of the importance of mnemonic code in complete understanding of a program, we will introduce and describe the mnemonic code along with the ladder diagram. Remember, you will not need to use the mnemonic code if you are inputting via a GPC, a FIT, or LSS (although you can use it with these devices too, if you prefer).

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 no operands, while others require up to three operands, Program Memory addresses can be from one to four words long.

Program Memory addresses start at 0000 and run until the capacity of Program Memory has been exhausted. 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	
0000	LD	HR	001
0001	AND		0001
0002	OR		0002
0003	LD NOT		0200
0004	AND		0201
0005	AND LD		0102
0006	MOV(21)		
			00
		DM	00
0007	CMP(20)		
		DM	00
		HR	0
0008	LD		0205
0009	OUT		0101
0010	MOV(21)		
		DM	00
		DM	05
0011	DIFU(13)		0002
0012	AND		0005
0013	OUT		0103

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 cycled to see if any addresses have been left out.

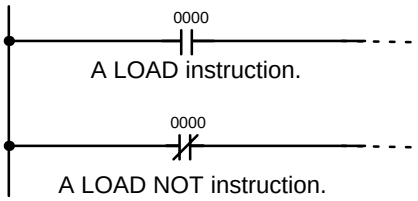
When programming, addresses are automatically displayed and do not have to be input unless for some reason a different location is desired for the instruction. When converting to mnemonic code, it is best to start at Program Memory address 0000 unless there is a specific reason for starting elsewhere.

4-3-3 Ladder Instructions

The ladder instructions are those 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 all other instructions are executed.

LOAD and LOAD NOT

The first condition that starts any logic block within a ladder diagram corresponds to a LOAD or LOAD NOT instruction.

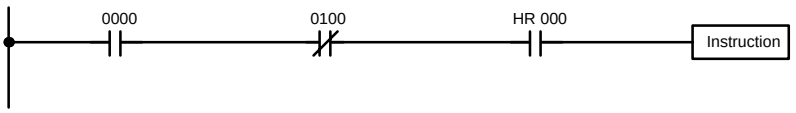


Address	Instruction	Operands
0000	LD	0000
0001	Instruction	
0002	LD NOT	0000
0003	Instruction	

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 IR 0000 was ON; for the LOAD NOT instruction (i.e., an normally closed condition), it would be ON when IR 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; 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.



Address	Instruction	Operands
0000	LD	0000
0001	AND NOT	0100
0002	AND	HR 000
0003	Instruction	

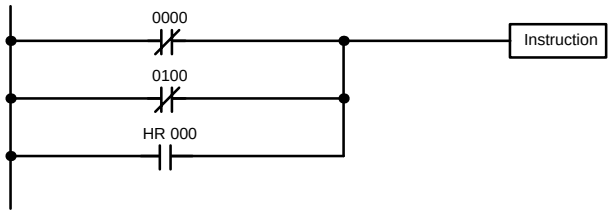
The instruction at the right would have an ON execution condition only when all three conditions are ON, i.e., when IR 0000 was ON, IR 0100 was OFF, and HR 000 was ON.

Actually, AND instructions can be considered individually in series, each of which would take the logical AND between the execution condition (i.e., the sum of all conditions up to that point) and the status of the AND instruction's operand bit. If both of these were ON, an ON execution condition would be produced for the next instruction. The execution condition for the first AND instruction in a series would be 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.



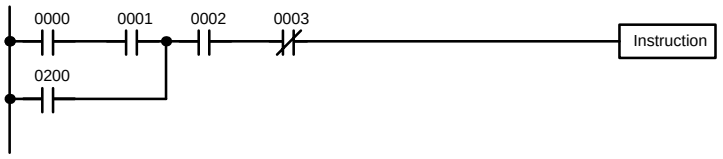
Address	Instruction	Operands
0000	LD	0000
0001	OR NOT	0100
0002	OR	HR 000
0003	Instruction	

The instruction at the right would have an ON execution condition when any one of the three conditions was ON, i.e., when IR 0000 was OFF, when IR 0100 was OFF, or when HR 000 was ON.

OR and OR NOT instructions can also 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.

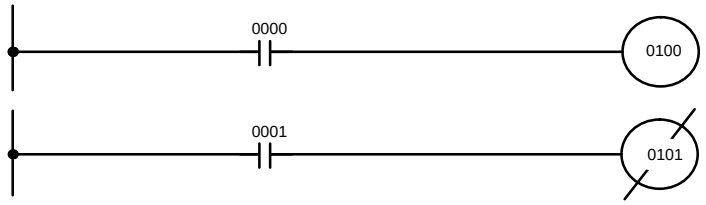


Address	Instruction	Operands
0000	LD	0000
0001	AND	0001
0002	OR	0200
0003	AND	0002
0004	AND NOT	0003
0005	Instruction	

Here, an AND is taken between the status of 0000 and that of 0001 to determine the execution condition for an OR with the status of 0200. The result of this operation determines the execution condition for an AND with the status of 0002, which in turn determines the execution condition for an AND with the inverse of the status of 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.

4-3-4 OUT and OUT NOT

The OUT and OUT NOT instructions are used to control the status of the designated operand bit according to the execution condition. With the OUT 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 OUT 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 follows:



Address	Instruction	Operands
0000	LD	0000
0001	OUT	0100

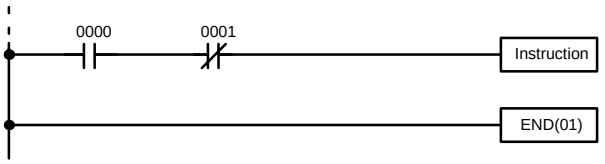
Address	Instruction	Operands
0000	LD	0001
0001	OUT NOT	0101

In the above examples, bit 0100 will be ON as long as 0000 is ON and bit 0101 will be OFF as long as 0001 is ON. Here, 0000 and 0001 would be input bits and 0100 and 0101 output bits assigned to the Units controlled by the PC, i.e., the signals coming in through the input points assigned 0000 and 0001 are controlling the output points assigned 0100 and 0101, respectively.

The length of time that a bit is ON or OFF can be controlled by combining the OUT or OUT NOT instruction with timer instructions. Refer to Examples under *5-11-1 TIMER - TIM* for details.

4-3-5 The END Instruction

The last instruction in any program must be the END instruction. When the CPU cycles the program, it executes all instructions 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.



Address	Instruction	Operands
0500	LD	0000
0501	AND NOT	0001
0502	Instruction	
0503	END(01)	---

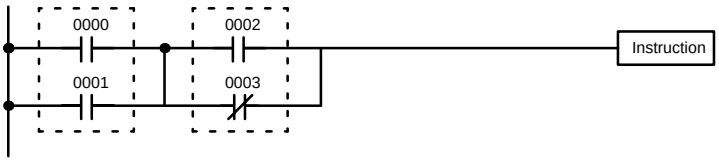
If there is no END instruction anywhere in the program, the program will not be executed at all.

4-3-6 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.



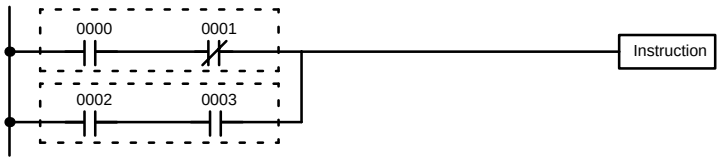
Address	Instruction	Operands
0000	LD	0000
0001	OR	0001
0002	LD	0002
0003	OR NOT	0003
0004	AND LD	---

The two logic blocks are indicated by dotted lines. Studying this example shows that an ON execution condition would be produced when both 1) either of the conditions in the left logic block was ON (i.e., when either 0000 or 0001 was ON) and 2) either of the conditions in the right logic block was ON (i.e., when either 0002 was ON or 0003 was OFF).

Analyzing the diagram in terms of instructions, the condition at 0000 would be a LOAD instruction and the condition below it would be an OR instruction between the status of 0000 and that of 0001. The condition at 0002 would be another LOAD instruction and the condition below this would be an OR NOT instruction, i.e., an OR between the status or 0002 and the inverse of the status of 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 allows us to do this. AND LOAD always takes an AND between the current execution condition and 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.

OR LOAD

Although we'll not describe it in detail, the following diagram would require 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 0000 was ON and 0001 was OFF or when 0002 and 0003 were both ON.



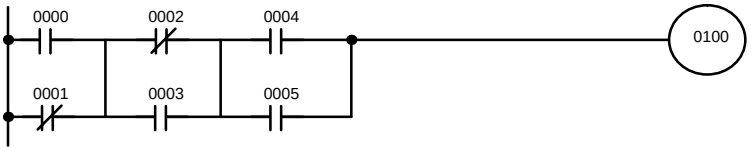
Address	Instruction	Operands
0000	LD	0000
0001	AND NOT	0001
0002	LD	0002
0003	AND	0003
0004	OR LD	---

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 using a LOAD instruction to code the first condition, and then AND LOAD or OR LOAD is used to logically combine the blocks. With both AND LOAD and OR LOAD there are two ways to achieve this. One is to code the logic block instruction after the first two blocks and then after each additional block. The other is to code all of the blocks to be combined, starting each block with LOAD or LOAD NOT, and then to code the logic block instructions which combine them. In this case, the instructions for the last pair of blocks should be combined first, and then each preceding block should be combined, working progressively back to the first block. Although either of these methods will produce exactly the same result, the second method, that of coding all logic block instructions together, can be used only if eight or fewer blocks are being combined, i.e., if seven or fewer logic block instructions are required.

The following diagram requires AND LOAD to be converted to mnemonic code because three pairs of parallel conditions lie in series. The two means of coding the programs are also shown.



Address	Instruction	Operands
0000	LD	0000
0001	OR NOT	0001
0002	LD NOT	0002
0003	OR	0003
0004	AND LD	---
0005	LD	0004
0006	OR	0005
0007	AND LD	---
0008	OUT	0100

Address	Instruction	Operands
0000	LD	0000
0001	OR NOT	0001
0002	LD NOT	0002
0003	OR	0003
0004	LD	0004
0005	OR	0005
0006	AND LD	---
0007	AND LD	---
0008	OUT	0100

Again, with the method on the right, a maximum of eight blocks can be combined. There is no limit to the number of blocks that can be combined with the first method.

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 can be coded first, followed by OR LOAD, the last block, and another OR LOAD, or the three blocks can be coded first followed by two OR LOADs. The mnemonic code for both methods is shown below.

Address	Instruction	Operands
0000	LD	0000
0001	AND NOT	0001
0002	LD NOT	0002
0003	AND NOT	0003
0004	OR LD	---
0005	LD	0004
0006	AND	0005
0007	OR LD	---
0008	OUT	0101

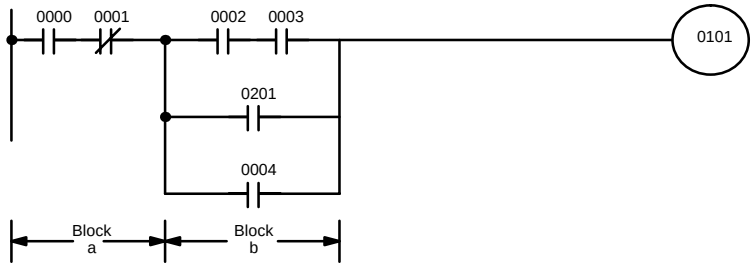
Address	Instruction	Operands
0000	LD	0000
0001	AND NOT	0001
0002	LD NOT	0002
0003	AND NOT	0003
0004	LD	0004
0005	AND	0005
0006	OR LD	---
0007	OR LD	---
0008	OUT	0101

Again, with the method on the right, a maximum of eight blocks can be combined. There is no limit to the number of blocks that can be combined with the first method.

Combining AND LD and OR LD

Both of the coding methods described above can also be used when using both AND LD and OR LD, as long as the number of blocks being combined does not exceed eight.

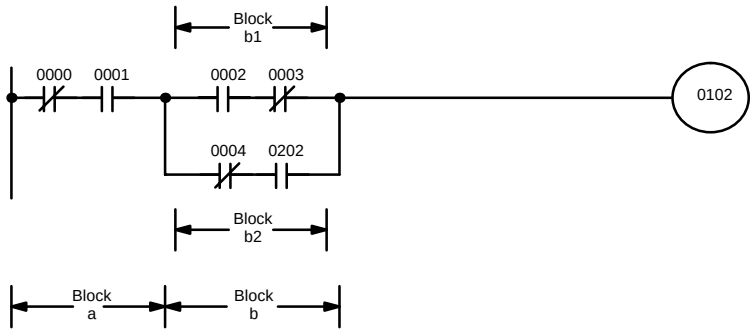
The following diagram contains only two logic blocks as shown. It is not necessary to break block b down further, because it can coded directly using only AND and OR.



Address	Instruction	Operands
0000	LD	0000
0001	AND NOT	0001
0002	LD	0002
0003	AND	0003
0004	OR	0201
0005	OR	0004
0006	AND LD	---
0007	OUT	0101

Although the following diagram is similar to the one above, block b in the diagram below cannot be coded without being broken down into two blocks combined with OR LD. In this example, the three blocks have been coded first and then OR LD has been used to combine the last two blocks followed by AND LD to combine the execution condition produced by the OR LD with the execution condition of block a.

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.



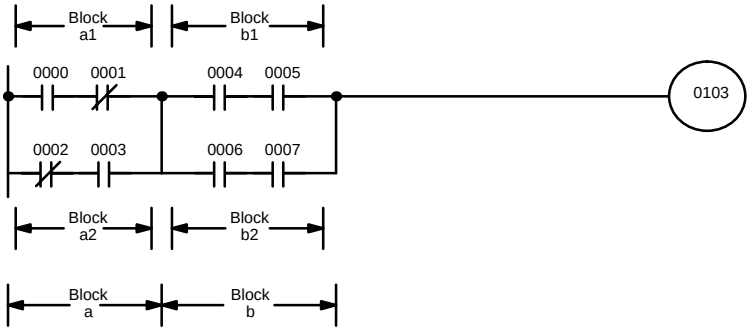
Address	Instruction	Operands
0000	LD NOT	0000
0001	AND	0001
0002	LD	0002
0003	AND NOT	0003
0004	LD NOT	0004
0005	AND	0202
0006	OR LD	---
0007	AND LD	---
0008	OUT	0102

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. AND LD and OR LD is used to combine either, i.e., AND LD or OR LD 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.

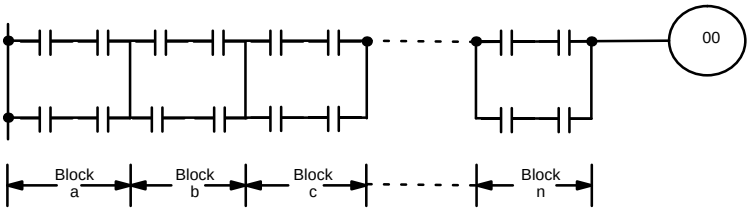
When working with complicated diagrams, blocks will ultimately be coded starting at the top left and moving down before moving across. This will generally mean that, when there might be a choice, OR LD will be coded before AND LD.

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 LD. Before AND LD can be used, however, OR LD must be used to combine the top and bottom blocks on both sides, i.e., to combine a1 and a2; b1 and b2.

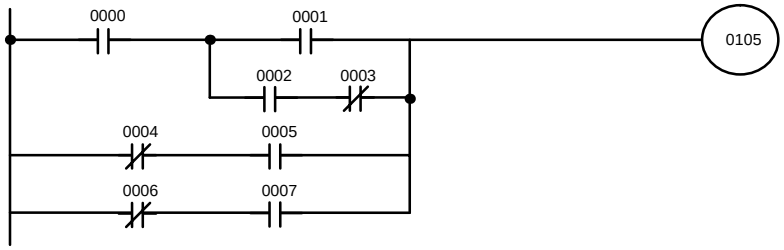


Address	Instruction	Operands	
0000	LD	0000	
0001	AND NOT	0001	
0002	LD NOT	0002	
0003	AND	0003	
0004	OR LD	---	Blocks a1 and a2
0005	LD	0004	
0006	AND	0005	
0007	LD	0006	
0008	AND	0007	
0009	OR LD	---	Blocks b1 and b2
0010	AND LD	---	Blocks a and b
0011	OUT	0103	

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

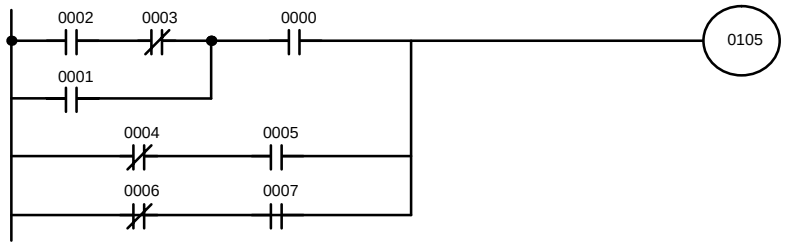


The following diagram requires first an OR LD and an AND LD to code the top of the three blocks, and then two more OR LDs to complete the mnemonic code.



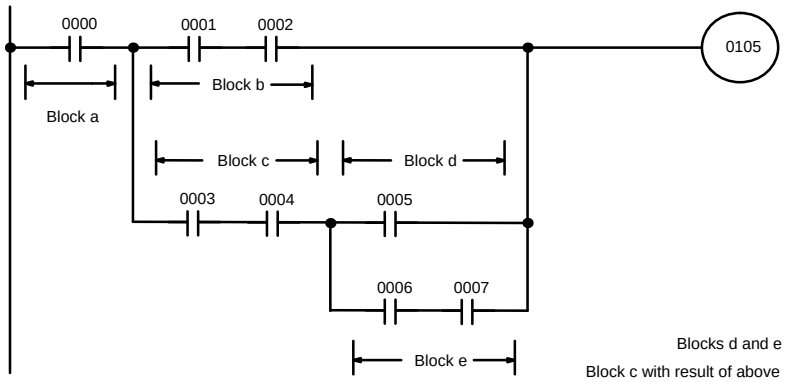
Address	Instruction	Operands
0000	LD	0000
0001	LD	0001
0002	LD	0002
0003	AND NOT	0003
0004	OR LD	---
0005	AND LD	---
0006	LD NOT	0004
0007	AND	0005
0008	OR LD	---
0009	LD NOT	0006
0010	AND	0007
0011	OR LD	---
0012	OUT	0105

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



Address	Instruction	Operands
0000	LD	0002
0001	AND NOT	0003
0002	OR	0001
0003	AND	0000
0004	LD NOT	0004
0005	AND	0005
0006	OR LD	---
0007	LD NOT	0006
0008	AND	0007
0009	OR LD	---
0010	OUT	0105

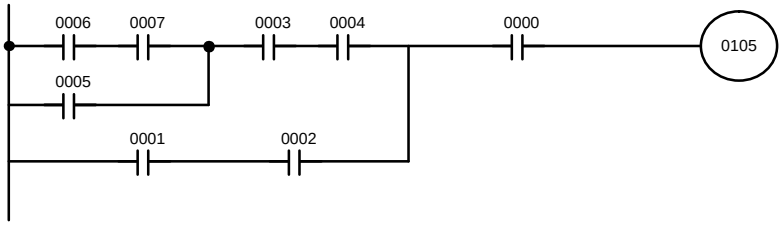
The following diagram requires five blocks, which here are coded in order before using OR LD and AND LD to combine them starting from the last two blocks and working forward. The OR LD at address 0008 combines blocks d and e, the following AND LD combines the resulting execution condition with that of block c, etc.



Blocks d and e
Block c with result of above
Block b with result of above
Block a with result of above

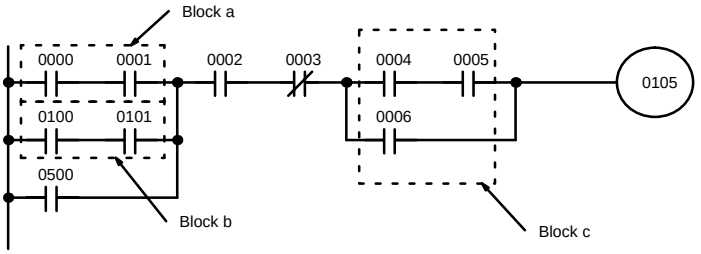
Address	Instruction	Operands
0000	LD	0000
0001	LD	0001
0002	AND	0002
0003	LD	0003
0004	AND	0004
0005	LD	0005
0006	LD	0006
0007	AND	0007
0008	OR LD	---
0009	AND LD	---
0010	OR LD	---
0011	AND LD	---
0012	OUT	0105

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

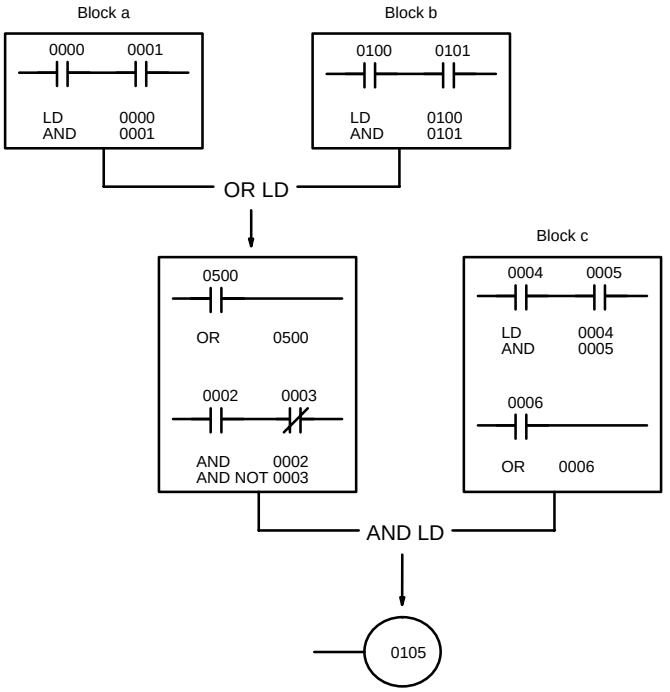


Address	Instruction	Operands
0000	LD	0006
0001	AND	0007
0002	OR	0005
0003	AND	0003
0004	AND	0004
0005	LD	0001
0006	AND	0002
0007	OR LD	---
0008	AND	0000
0009	OUT	0105

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



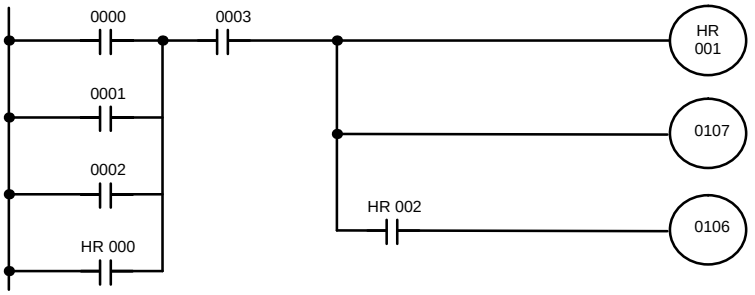
The first logic block instruction is used to combine the execution conditions resulting from blocks a and b, and the second one is used to combine the execution condition of block c with the execution condition resulting from the normally closed condition assigned 0003. The rest of the diagram can be coded with ladder instructions. The logical flow for this and the resulting code are shown below.



Address	Instruction	Operands
0000	LD	0000
0001	AND	0001
0002	LD	0100
0003	AND	0101
0004	OR LD	---
0005	OR	0500
0006	AND	0002
0007	AND NOT	0003
0008	LD	0004
0009	AND	0005
0010	OR	0006
0011	AND LD	---
0012	OUT	0105

4-3-7 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.



Address	Instruction	Operands
0000	LD	0000
0001	OR	0001
0002	OR	0002
0003	OR	HR 000
0004	AND	0003
0005	OUT	HR 001
0006	OUT	0107
0007	AND	HR 002
0008	OUT	0106

4-3-8 Branching Instruction Lines

When an instruction line branches into two or more lines, it is sometimes necessary to use either interlocks or TR bits to maintain the execution condition that existed at a branching point. This is because instruction lines are executed across to a terminal instruction on the right before returning to branching points to execute instructions on the branch lines. If the execution condition has changed during this time, the previous execution condition is lost and proper execution will not be possible without some means of preserving the previous condition. 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.

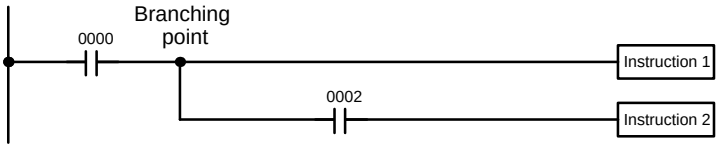


Diagram A: OK

Address	Instruction	Operands
0000	LD	0000
0001	Instruction 1	
0002	AND	0002
0003	Instruction 2	

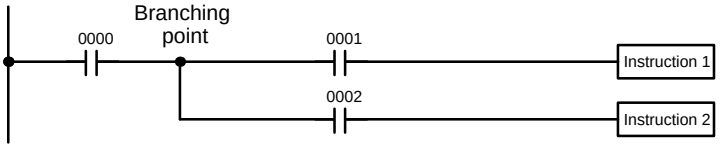


Diagram B: Needs Correction

Address	Instruction	Operands
0000	LD	0000
0001	AND	0001
0002	Instruction 1	
0003	AND	0002
0004	Instruction 2	

If, as shown in diagram A, the execution condition that existed at the branching point is not 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 at the end of the top line will sometimes be different, making it impossible to ensure correct execution of the branch line. The system remembers only the current execution condition (i.e., the logical sum for an entire line) and does not remember partial logical sums at points within a line.

There are two means of programming branching programs to preserve the execution conditions. One is to use TR bits; the other, to use interlocks (IL(02)/ILC(03)).

TR Bits

The TR area provides eight bits, TR 0 through TR 7, that can be used to temporarily preserve execution conditions. If a TR bit is used as the operand of the OUTPUT instruction placed at a branching point, the current execution condition will be stored at the designated TR bit. Storing execution conditions is a special application of the OUTPUT instruction. When returning to the branching point, the same TR bit is then used as the operand of the LOAD instruction to restore the execution condition that existed when the branching point was first reached in program execution.

The above diagram B can be written as shown below to ensure correct execution.

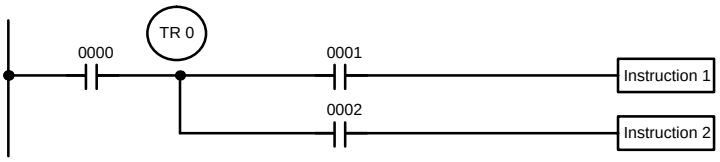
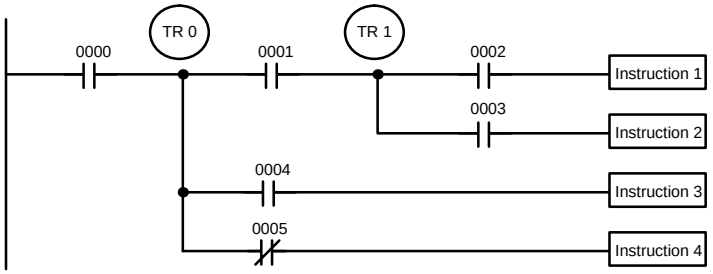


Diagram B: Corrected Using a TR bit

Address	Instruction	Operands	
0000	LD		0000
0001	OUT	TR	0
0002	AND		0001
0003	Instruction 1		
0004	LD	TR	0
0005	AND		0002
0006	Instruction 2		

In terms of actual instructions the above diagram would be as follows: The status of 0000 is loaded (a LOAD instruction) to establish the initial execution condition. This execution condition is then output using an OUTPUT instruction to TR 0 to store the execution condition at the branching point. The execution condition is then ANDed with the status of 0001 and instruction 1 is executed accordingly. The execution condition that was stored at the branching point is then loaded back in (a LOAD instruction with TR 0 as the operand) and instruction 2 is executed accordingly.

The following example shows an application using two TR bits.

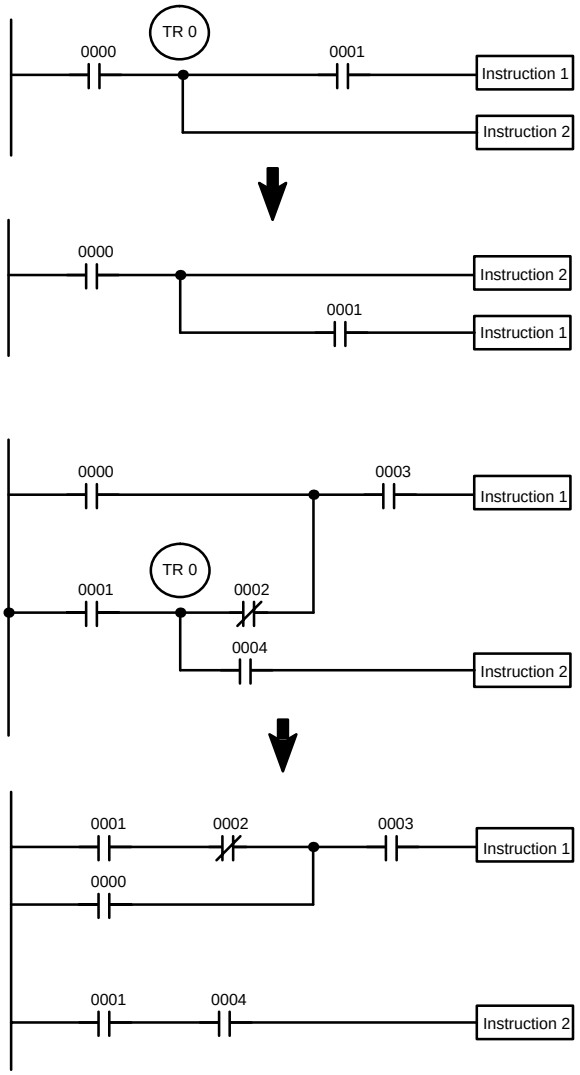


Address	Instruction	Operands	
0000	LD		0000
0001	OUT	TR	0
0002	AND		0001
0003	OUT	TR	1
0004	AND		0002
0005	OUT		0500
0006	LD	TR	1
0007	AND		0003
0008	OUT		0501
0009	LD	TR	0
0010	AND		0004
0011	OUT		0502
0012	LD	TR	0
0013	AND NOT		0005
0014	OUT		0503

In this example, TR 0 and TR 1 are used to store the execution conditions at the branching points. After executing instruction 1, the execution condition stored in TR 1 is loaded for an AND with the status 0003. The execution condition stored in TR 0 is loaded twice, the first time for an AND with the status of 0004 and the second time for an AND with the inverse of the status of 0005.

TR bits can be used as many times as required as long as the same TR bit is not used more than once in the same instruction block. Here, a new instruction block is begun each time execution returns to the bus bar. If more than eight branching points requiring that the execution condition be saved are necessary in a single instruction block, interlocks, which are described next, must be used.

When drawing a ladder diagram, be careful not to use TR bits unless necessary. Often the number of instructions required for a program can be reduced and ease of understanding a program increased by redrawing a diagram that would otherwise required TR bits. With both of the following pairs of diagrams, the versions on the top require fewer instructions and do not require TR bits. The first example achieves this by merely reorganizing the parts of the instruction block; the second, by separating the second OUTPUT instruction and using another LOAD instruction to create the proper execution condition for it.



Note TR bits are only used when programming using mnemonic code and are not necessary when inputting ladder diagrams directly, as is possible from a GPC. The above limitations on the number of branching points requiring TR bits and considerations on methods to reduce the number of programming instructions still hold.

Interlocks

The problem of storing execution conditions at branching points can also be handled by using the INTERLOCK (IL(02)) and INTERLOCK CLEAR (ILC(03)) instructions. The branching point and all the conditions leading to it are placed on a separate line followed by all of the lines from the branching point. Each branch line is thus established as an new instruction line, with the first condition on each branch line corresponding to a LOAD or LOAD NOT instruction. If the execution condition for the INTERLOCK instruction is OFF, all instructions on the right side of the branch lines leading from the branching point receive an OFF execution condition through the first INTERLOCK CLEAR instruction. The effect that this has on particular instructions is described in *5-7 INTERLOCK and INTERLOCK CLEAR - IL(02) and ILC(03)*.

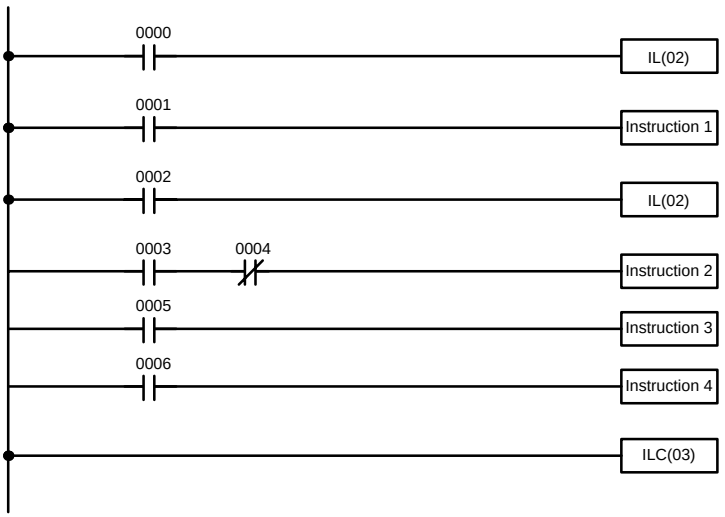
Diagram B from the initial example can also be corrected with an interlock. As shown below, this requires two more instruction lines for the interlock instructions.

Diagram B: Corrected with an Interlock

Address	Instruction	Operands
0000	LD	0000
0001	IL(02)	---
0002	LD	0001
0003	Instruction 1	
0004	LD	0002
0005	Instruction 2	
0006	ILC(03)	---

If 0000 is ON in the revised version of diagram B, above, the status of 0001 and that of 0002 would determine the execution conditions for instructions 1 and 2, respectively, on independent instruction lines. Because here 0000 is ON, this would produce the same results as ANDing the status of each of these bits, as would occur if the interlock was not used, i.e., the INTERLOCK and INTERLOCK CLEAR instructions would not affect execution. If 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.



Address	Instruction	Operands
0000	LD	0000
0001	IL(02)	---
0002	LD	0001
0003	Instruction 1	
0004	LD	0002
0005	IL(02)	---
0006	LD	0003
0007	AND NOT	0004
0008	Instruction 2	
0009	LD	0005
0010	Instruction 3	
0011	LD	0006
0012	Instruction 4	
0013	ILC(03)	---

If 0000 in the above diagram was OFF (i.e., if the execution condition for the first INTERLOCK instruction was OFF), instructions 1 through 4 would be executed with OFF execution conditions and execution would move to the instruction following the INTERLOCK CLEAR instruction. If 0000 was ON, the status of 0001 would be loaded to form the execution condition for instruction 1 and then the status of 0002 would be loaded to form the first execution status for that instruction line, i.e., the execution condition for the second INTERLOCK instruction. If 0002 was OFF, instructions 2 through 4 would be executed with OFF execution conditions. If 0002 was ON, 0003, 0005, and 0006 would be executed as written.

4-3-9 Jumps

A specific section of a program can be skipped according to a designated execution condition. Although this is similar to what happens when the execution condition for an INTERLOCK instruction is OFF, with jumps, the operands for all instructions maintain status. Jumps can therefore be used to control devices that require a sustained output, e.g., pneumatics and hydraulics, whereas interlocks can be used to control devices that do not required a sustained output, e.g., electronic instruments.

Jumps are created using the JUMP (JMP(04)) and JUMP END (JME(05)) instructions. If the execution condition for a JUMP instruction is ON, the program is executed normally as if the jump did not exist. If the execution condition for the JUMP instruction is OFF, program execution moves immediately to a JUMP END instruction without changing the status of anything between the JUMP and JUMP END instruction. Actually there are two types of jumps.

All JUMP and JUMP END instructions are assigned jump numbers ranging between 00 and 08. The jump number used determines the type of jump.

A jump can be defined using jump numbers 01 through 08 only once, i.e., each of these numbers can be used once in a JUMP instruction and once in a JUMP END instruction. When a JUMP instruction assigned one of these numbers is executed, execution moves immediately to the JUMP END instruction that has the same number as if all of the instruction between them did not exist. Diagram B from the TR bit and interlock example could be redrawn as shown below using a jump. Although 01 has been used as the jump number, any number between 01 and 08 could be used as long as it has not already been used in a different part of the program.

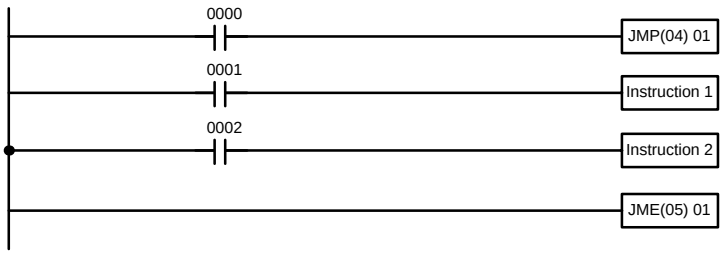


Diagram B: Corrected with a Jump

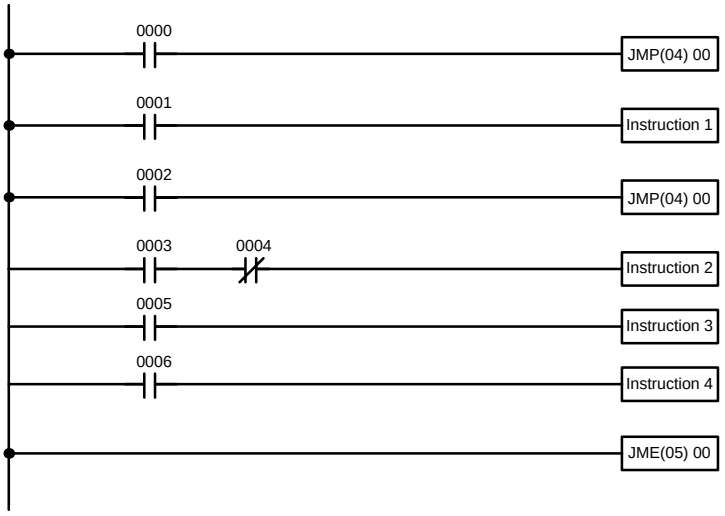
Address	Instruction	Operands
0000	LD	0000
0001	JMP(04)	01
0002	LD	0001
0003	Instruction 1	
0004	LD	0002
0005	Instruction 2	
0006	JME(05)	01

This version of diagram B would have a shorter execution time when 0000 was OFF than any of the other versions.

The other type of jump is created with a jump number of 00. As many jumps as desired can be created using jump number 00 and JUMP instructions using 00 can be used consecutively without a JUMP END using 00 between them. In the extreme, only one JUMP END 00 instruction is required for all JUMP 00 instructions. When 00 is used as the jump number for a JUMP instruction, program execution moves to the instruction following the next

JUMP END instruction with a jump number of 00. Although, as in all jumps, no status is changed and no instructions are executed between the JUMP 00 and JUMP END 00 instructions, the program must search for the next JUMP END 00 instruction, producing a slightly longer execution time.

Execution of programs containing multiple JUMP 00 instructions for one JUMP END 00 instruction resembles that of similar interlocked sections. The following diagram is the same as that used for the interlock example above, except redrawn with jumps. This diagram, however, would not execute the same, as has already be described, i.e., interlocks would reset certain parts of the interlocked section but jumps would not affect any status between the JUMP and JUMP END instructions.



Address	Instruction	Operands
0000	LD	0000
0001	JMP(04)	00
0002	LD	0001
0003	Instruction 1	
0004	LD	0002
0005	JMP(04)	00
0006	LD	0003
0007	AND NOT	0004
0008	Instruction 2	
0009	LD	0005
0010	Instruction 3	
0011	LD	0006
0012	Instruction 4	
0013	JME(05)	00

Jump diagrams can also be drawn as branching instruction lines if desired and would look exactly like their interlock equivalents. The non-branching form, which is the form displayed on the GPC, will be used in this manual.

4-4 The Programming Console

Depending on the model of Programming Console used, it is either connected to the CPU via a Programming Console Adapter and Connecting Cable or it is mounted directly to the CPU.

4-4-1 The Keyboard

The keyboard of the Programming Console is functionally divided by key color into the following four areas:

White Numeric Keys

The ten white keys are used to input numeric program data such as program addresses, data area addresses, and operand values. The numeric keys are also used in combination with the function key (FUN) to enter instructions with function codes.

Red CLR Key

The CLR key clears the display and cancels current Programming Console operations. It is also used when you key in the password at the beginning of programming operations. Any Programming Console operation can be cancelled by pressing the CLR key, although the CLR key may have to be pressed two or three times to cancel the operation and clear the display.

Yellow Operation Keys

The yellow keys are used for writing and correcting programs. Detailed explanations of their functions are given later in this section.

Gray Instruction and Data Area Keys

Except for the SHIFT key on the upper right, the gray keys are used to input instructions and designate data area prefixes when inputting or changing a program. The SHIFT key is similar to the shift key of a typewriter, and is used to alter the function of the next key pressed. (It is not necessary to hold the SHIFT key down; just press it once and then press the key to be used with it.)

The gray keys other than the SHIFT key have either the mnemonic name of the instruction or the abbreviation of the data area written on them. The functions of these keys are described below.

FUN — Pressed before the function code when inputting an instruction via its function code.

SFT — Pressed to enter SFT (the Shift Register instruction).

NOT — Input after a ladder instruction to designate an normally closed condition.

AND — Pressed to enter AND (the AND instruction) or used with NOT to enter AND NOT.

OR — Pressed to enter OR (the OR instruction) or used with NOT to enter OR NOT.

CNT — Pressed to enter CNT (the Counter instruction) or to designate a TC number that has already been defined as a counter.

LD — Pressed to enter LD (the Load instruction) or used with NOT to enter LD NOT. Also pressed to indicate an input bit.

OUT — Pressed to enter OUT (the Output instruction) or used with NOT to enter OUT NOT. Also pressed to indicate an output bit.

TIM — Pressed to enter TIM (the Timer instruction) or to designate a TC number that has already been defined as a timer.

TR — Pressed before designating an address in the TR area.

LR — Pressed before designating an address in the LR area. Cannot be used with the K-type PCs.

HR — Pressed before designating an address in the HR area.

DM — Pressed before designating an address in the DM area.

CH / * — Pressed before designating an indirect DM address. Cannot be used with the K-type PCs.

SHIFT + CH / * — Pressed before designating a word address.

CONT / # — Pressed before designating an operand as a constant.

SHIFT + CONT / # — Pressed before designating a bit address.

4-4-2 PC Modes

The Programming Console is equipped with a switch to control the PC mode. To select one of three operating modes—RUN, MONITOR, or PROGRAM—use the mode switch. The mode that you select will determine PC operation as well as the procedures that are possible from the Programming Console.

RUN mode is the mode used for normal program execution. When the switch is set to RUN and the START input on the CPU Power Supply Unit is ON, the CPU will begin executing the program according to the program written in its Program Memory. Although monitoring PC operation from the Programming Console is possible in RUN mode, no data in any of the memory areas can be input or changed.

MONITOR mode allows you to visually monitor in-progress program execution while controlling I/O status, changing PV (present values) or SV (set values), etc. In MONITOR mode, I/O processing is handled in the same way as in RUN mode. MONITOR mode is generally used for trial system operation and final program adjustments.

In PROGRAM mode, the PC does not execute the program. PROGRAM mode is for creating and changing programs, clearing memory areas, and registering and changing the I/O table. A special Debug operation is also available within PROGRAM mode that enables checking a program for correct execution before trial operation of the system.

⚠ **DANGER** Do not leave the Programming Console connected to the PC by an extension cable when in RUN mode. Noise entering via the extension cable can affect the program in the PC and thus the controlled system.

Mode Changes

When the PC is turned on, the mode it is in will depend on what Peripheral Device, if any, is connected or mounted to the CPU.

- **No Peripheral Device Connected**
  When power is applied to the PC without a Peripheral Device connected, the PC is automatically set to RUN mode. Program execution is then controlled through the CPU Power Supply Unit's START terminal.
- **Programming Console Connected**
  If the Programming Console is connected to the PC when PC power is applied, the PC is set to the mode set on the Programming Console's mode switch.
- **Other Peripheral Connected**
  If a Peripheral Interface Unit, PROM Writer, Printer Interface Unit, or a Floppy Disk Interface Unit is attached to the PC when PC power is turned on, the PC is automatically set to PROGRAM mode.

If the PC power supply is already turned on when a peripheral device is attached to the PC, the PC will stay in the same mode it was in before the peripheral device was attached. The mode can be changed with the mode switch on the Programming Console once the password has been entered. If it is necessary to have the PC in PROGRAM mode, (for the PROM Writer, Floppy Disk Interface Unit, etc.), be sure to select this mode before connecting the peripheral device, or alternatively, apply power to the PC after the peripheral device is connected.

The mode will also not change when a Peripheral Device is removed from the PC after PC power is turned on.

⚠ **DANGER** 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 any PC-controlled system to begin operation. Also be sure that starting operation is safe and appropriate whenever turning on the PC without a device mounted to the CPU when the START input on the CPU Power Supply Unit is ON.

4-5 Preparation for Operation

This section describes the procedures required to begin Programming Console operation. These include password entry, clearing memory, and error message clearing.

The following sequence of operations must be performed before beginning initial program input.

1, 2, 3...

1. Confirm that all wiring for the PC has been installed and checked properly.
2. Confirm that a RAM Unit is mounted as the Memory Unit and that the write-protect switch is OFF.

3. Connect the Programming Console to the PC. Make sure that the Programming Console is securely connected or mounted to the CPU; improper connection may inhibit operation.
4. Set the mode switch to PROGRAM mode.
5. Turn on PC power.
6. Enter the password.
7. Clear memory.

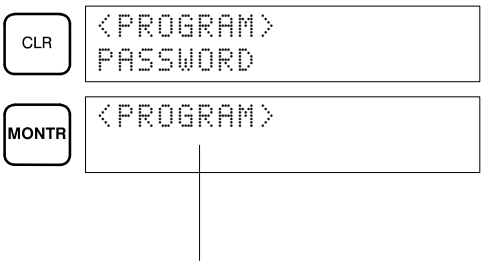
Each of these operations from entering the password on is described in detail in the following subsections. All operations should be done in PROGRAM mode unless otherwise noted.

4-5-1 Entering the 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 MONTR. 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. **Be sure 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 or MONITOR. You can change the mode to RUN or MONITOR with the mode switch after entering the password.



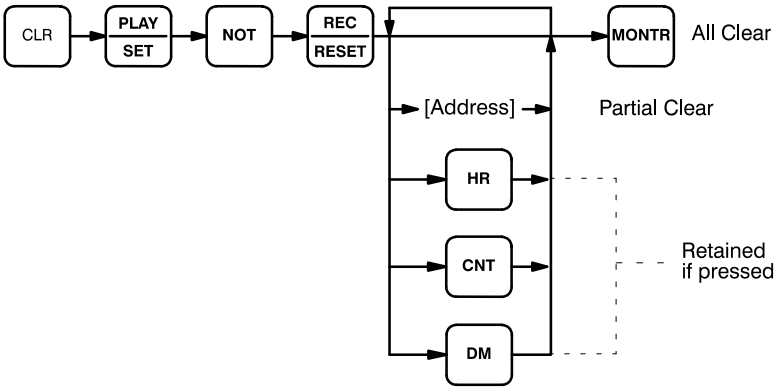
Indicates the mode set by the mode selector switch.

4-5-2 Clearing Memory

Using the Memory Clear operation it is possible to clear all or part of the Program Memory, and the IR, HR, DM and TC areas. Unless otherwise specified, the clear operation will clear all memory areas above provided that the Memory Unit attached to the PC is a RAM Unit or an EEPROM Unit and the write-protect switch is OFF. If the write-protect switch is ON, or the Memory Unit is an EPROM Unit, Program Memory cannot be cleared.

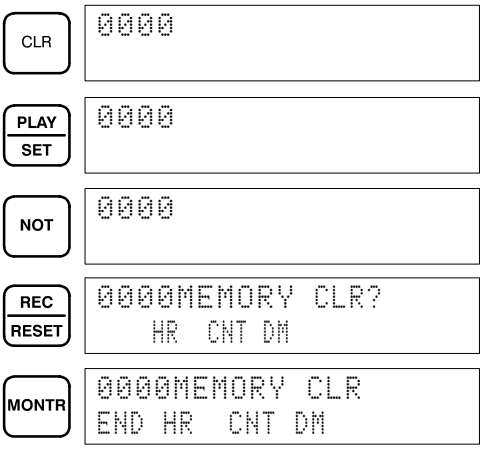
Before beginning to programming for the first time or when installing a new program, all areas should normally be cleared. Before clearing memory, check to see if a program is already loaded that you need. If you need the program, clear only the memory areas that you do not need, and be sure to check the existing program with the program check key sequence before using it. The check sequence is provided later in this section. Further debugging methods are provided in *Section 7 Program Debugging and Execution*. To clear all memory areas, press CLR until all zeros are displayed and then the top line of the following sequence. The branch lines in the sequence are used when clearing only part of the memory areas, which is described below. Memory can be cleared in PROGRAM mode only.

Key Sequence



All Clear

The following procedure is used to clear memory completely.

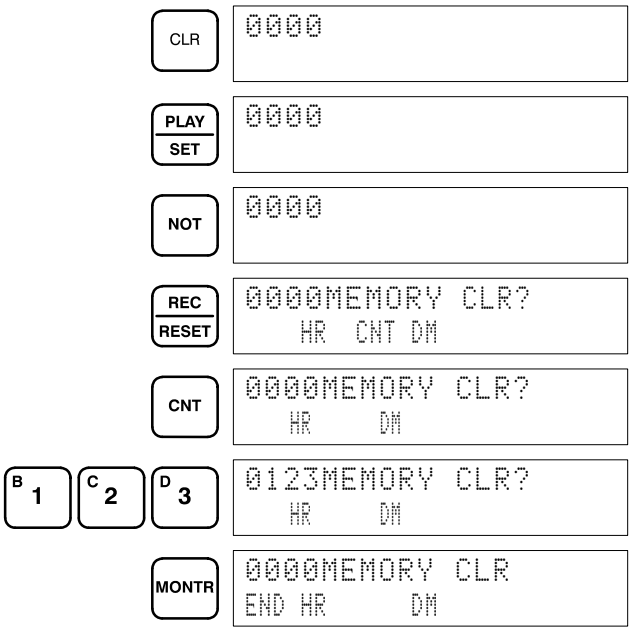


Partial Clear

It is possible to retain the data in specified areas and/or part of the Program Memory. To retain the data in the HR and TC, and/or DM areas, press the appropriate key after entering REC/RESET. The CNT key 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 beginning to a specified address. After designating the data areas to be retained, specify the first Program Memory address to be cleared. For example, to leave addresses 0000 to 0122 untouched, but to clear addresses from 0123 to the end of Program Memory, input 0123.

For example, to leave the TC area uncleared and retaining Program Memory addresses 0000 through 0122, input as follows:



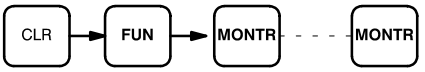
4-5-3 Clearing Error Messages

Any error messages recorded in memory should also 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 beeper 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 8 Troubleshooting*).

To display any recorded error messages, press CLR, FUN, and then MONTR. The first message will appear. Pressing MONTR again will clear the present message and display the next error message. Continue pressing MONTR until all messages have been cleared.

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

Key Sequence



4-6 Inputting, Modifying, and Checking 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 before trial execution and finally correction under actual conditions can begin.

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 cycle 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. Further debugging methods are provided in *Section 7 Program Debugging and Execution*.

4-6-1 Setting and Reading from Program Memory Address

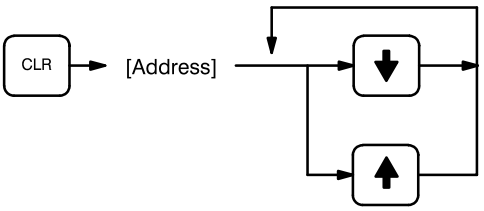
When inputting a program for the first time, it is generally input from Program Memory address 0000. As this address appears when the display is cleared, it is not necessary to input it.

When inputting a program starting from other than 0000 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. Leading zeros of the address need not be input, i.e., when specifying an address such as 0053 you need to enter only 53. The contents of the designated address will not be displayed until the down key is pressed.

Once the down key has been pressed to display the contents of the designated address, the up and down keys can 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.

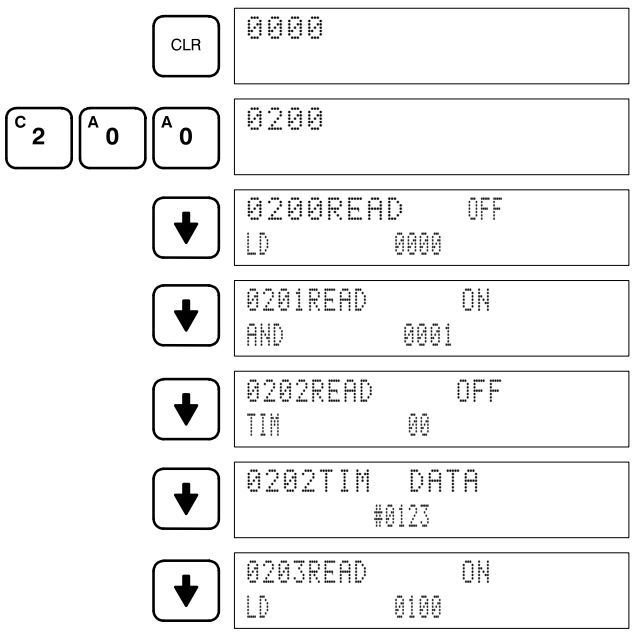
If Program Memory is read in RUN or MONITOR mode, the ON/OFF status of any bit displayed will also be shown.

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
0200	LD	0000
0201	AND	0001
0202	TIM	00
		# 0123
0203	LD	0100

4-6-2 Inputting or Overwriting Programs

Programs can be input or overwritten only in PROGRAM mode.

The same procedure is used to either input a program for the first time or to overwrite a program that already exists. In either case, the current contents of Program Memory are overwritten, i.e., if there is no previous program, the NOP(00) instruction, which will be written at every address, will be overwritten.

To input a program, just follow the mnemonic code that was produced from the ladder diagram, making sure that the proper address is set before starting. Once the proper address is displayed, input the first instruction word, press WRITE, and then input any operands required, pressing WRITE after each, i.e., WRITE is pressed at the end of each line of the mnemonic code. When WRITE is pressed, the designated instruction will be input 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 input.

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 but IR and SR addresses by pressing the corresponding data area key or to designate a constant by pressing CONT/#. CONT/# is not required for counter or timer SV (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.

Inputting SV for Counters and Timers

The SV (set value) for a timer or counter is generally input 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 WRITE. To designate a word, press CLR and then input the word address as described above.

Designating Instructions

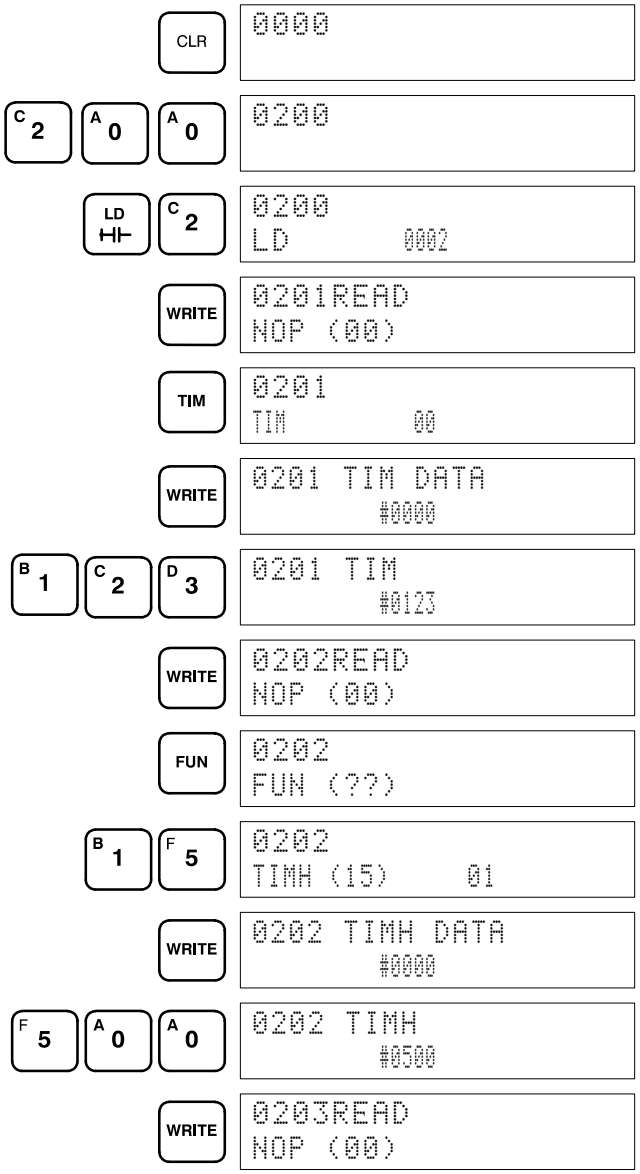
The most basic instructions are input using the Programming Console keys provided for them. All other instructions are input using function codes. These function codes are always written after the instruction's mnemonic. If no function code is given, there should be a Programming Console key for that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

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

⚠ Caution Enter function codes with care.

Example

The following ladder diagram can be input using the key inputs shown below. Displays will appear as indicated.



Address	Instruction	Operands
0200	LD	0002
0201	TIM	00
		# 0123
0202	TIMH(15)	01
		# 0500

Error Messages

The following error messages may appear when inputting a program. Correct the error as indicated and continue with the input operation. The asterisks in the displays shown below will be replaced with numeric data, normally an address, in the actual display.

Message	Cause and correction
****REPL ROM	An attempt was made to write to ROM or to write-protected RAM. Be sure a RAM Unit is mounted and that its write-protect switch is set to OFF.
****PROG OVER	The instruction at the last address in memory is not NOP(00). Erase all unnecessary instructions at the end of the program or use a larger Memory Unit.
****ADDR OVER	An address was set that is larger than the highest memory in Program Memory. Input a smaller address
****SETDATA ERR	Data has been input in the wrong format or beyond defined limits, e.g., a hexadecimal value has been input for BCD. Reinput the data.
****I/O NO. ERR	A data area address has been designated that exceeds the limit of the data area, e.g., an address is too large. Confirm the requirements for the instruction and reinput the address.

4-6-3 Checking the Program

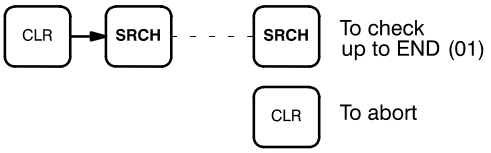
Once a program has been input, it should be checked for syntax to be sure that no programming rules have been violated. This check should also be performed if the program has been changed in any way that might create a syntax error.

To check the program, input the key sequence shown below. If an error is discovered, the check will stop and a display indicating the error will appear. Press SRCH to continue the check. If an error is not found, the program will be checked through the first END(01), with a display indicating when each 64 instructions have been checked (e.g., display #1 below).

CLR can be pressed to cancel the check after it has been started, and a display like display #2, in the example, will appear. When the check has reached the first END, a display like display #3 will appear.

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

Key Sequence



Error Messages

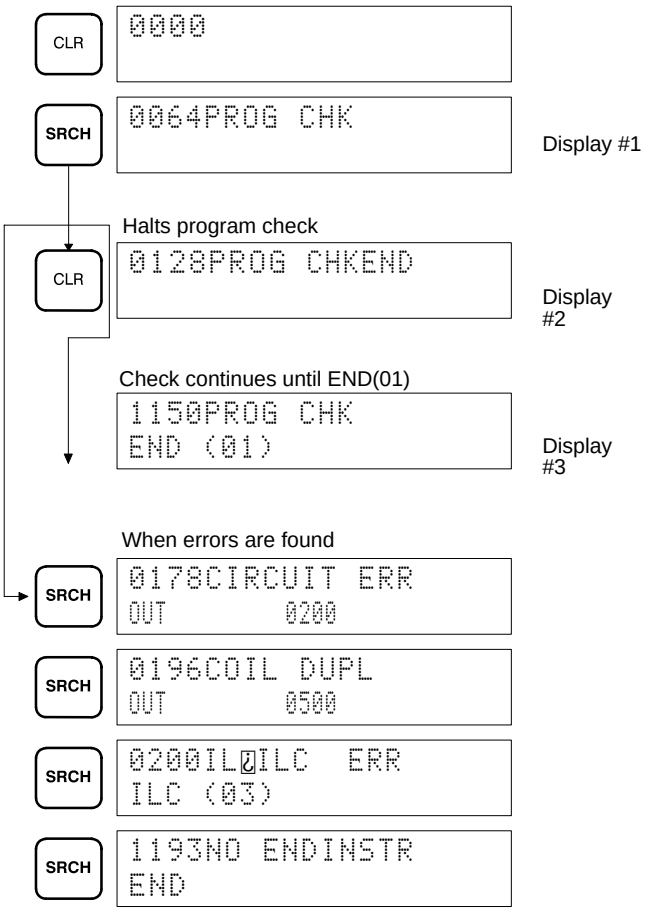
The following table provides the error types, displays, and explanations of all syntax errors. The address where the error was generated will also be displayed.

Many of the following errors are for instructions that have not been introduced yet. Refer to *4-7 Controlling Bit Status* or to *Section 5 Instruction Set* for details on these.

Message	Meaning and appropriate response
?????	The program has been destroyed. Reinput the program.
NO END INSTR	There is no END(01) in the program. Write END(01) at the final address in the program.
CIRCUIT ERR	The number of logic blocks and logic block instructions does not agree, i.e., either LD or LD NOT has been used to start a logic block whose execution condition has not been used by another instruction or a logic block instruction has been used that does not have the required number of logic blocks (i.e., unused execution conditions). Check your program.
IL-ILC ERR	IL(02) and ILC(03) are not used in pairs. Correct the program so that each IL(02) has a unique ILC(03). Although this error message will appear if more than one IL(02) is used with the same ILC(03), the program will be executed as written. Make sure your program is written as desired before proceeding.
JMP-JME ERR	JMP(04) and JME(05) are not used in pairs. Match each JMP(04) to a JME(05).
COIL DUPL	The same bit is being controlled (i.e., turned ON and/or OFF) by more than one instruction (e.g., OUT, OUT NOT, DIFU(13), DIFD(14), KEEP(11), SFT(10)). Although this is allowed for certain instructions, check instruction requirements to confirm that the program is correct or rewrite the program so that each bit is controlled by only one instruction.
DIF OVER	More than 48 DIFU and DIFDs are used in the program. Reduce the number of DIFU(13) and DIFD(14) used to 48 or less.
LOCN ERR	The instruction currently displayed is in the wrong area. Correct the program.
JME UNDEFD	The corresponding JME for a given JMP does not exist. Correct the program.
JMP UNDEFD	The corresponding JMP for a given JME does not exist. Correct the program.
DUPL	The number of the currently displayed instruction has already been programmed. Correct the program.
SBN-RET ERR	Incorrect usage of the displayed instruction (SBN or RET). Incorrect SBN usage is caused by more than one SBN having the same subroutine number. Correct the program.
SBN UNDEFD	The subroutine called by SBS does not exist. Correct the program.

Message	Meaning and appropriate response
SBS UNDEFD	A defined subroutine is not called by the main program. When this message is displayed because of interrupt routine definition, there is no problem. In all other cases, correct the program.
STEP OVER	STEP is used for more than 16 program sections. Correct the program to decrease the number of sections to 16 or less. When the GPC is used the message “CPU WAITG” is displayed.
SNXT OVER	More than 48 SNTXs are used in the program. Correct the program to decrease the number to 48 or less.
STEP ERR	STEP and SNXT are not correctly used. Correct the program.

Example The following examples shows some of the displays that can appear as a result of a program check.



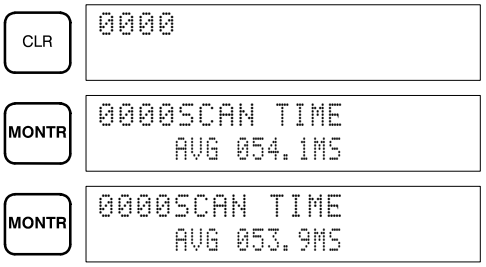
4-6-4 Displaying the Cycle Time

Once the program has been cleared of syntax errors, the cycle time should be checked. This is possible only in RUN or MONITOR mode while the program is being executed. See *Section 6 Program Execution Timing* for details on the cycle time.

To display the current average cycle time, press CLR then MONTR. The time displayed by this operation is an average cycle time. The differences in displayed values depend on the execution conditions that exist when MONTR is pressed.

Note Cycle time is displayed as scan time.

Example



4-6-5 Program Searches

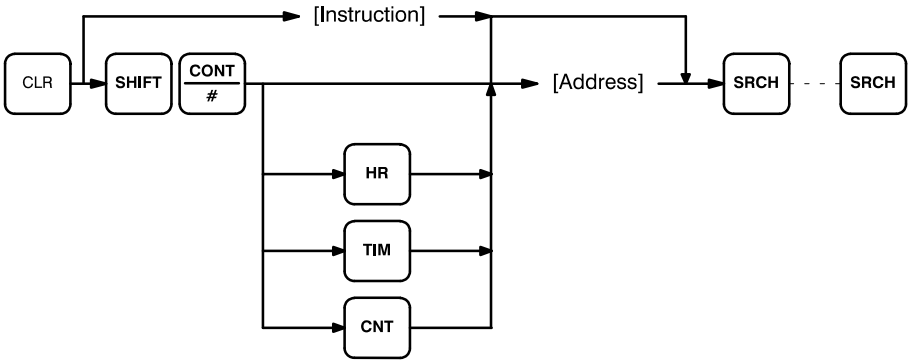
The program can be searched for occurrences of any designated instruction or data area bit address used in an instruction. Searches can be performed from any currently displayed address or from a cleared display.

To designate a bit address, press SHIFT, press CONT/#, then input the address, including any data area designation required, and press SRCH. To designate an instruction, input the instruction just as when inputting the program and press SRCH. 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 SRCH again. SRCHG will be displayed while a search is in progress.

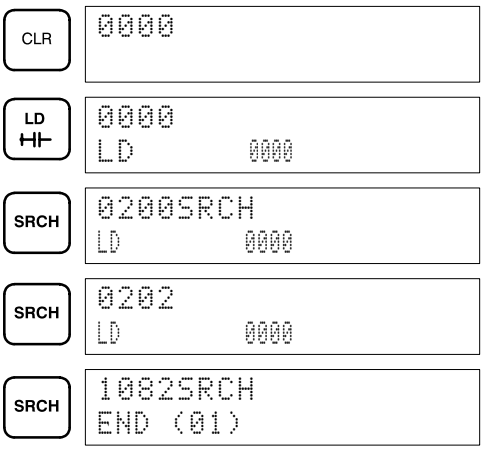
When the first word of a multiword instruction is displayed for a search operation, the other words of the instruction can be displayed by pressing the down key before continuing the search.

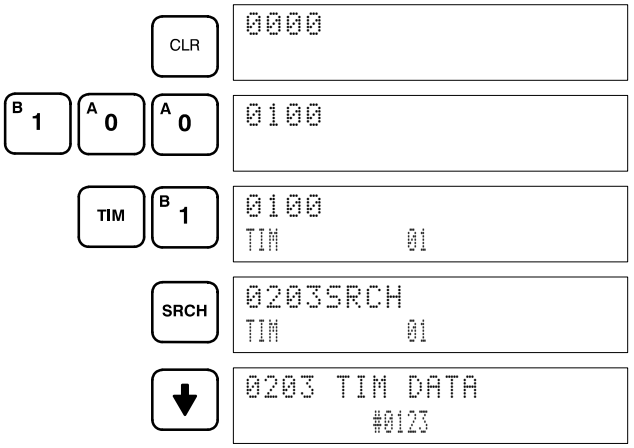
If Program Memory is read in RUN or MONITOR mode, the ON/OFF status of any bit displayed will also be shown.

Key Sequence

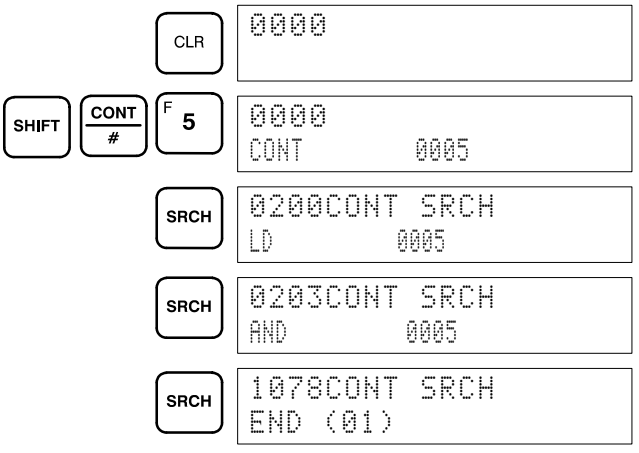


Example: Instruction Search





Example: Bit Search



4-6-6 Inserting and Deleting Instructions

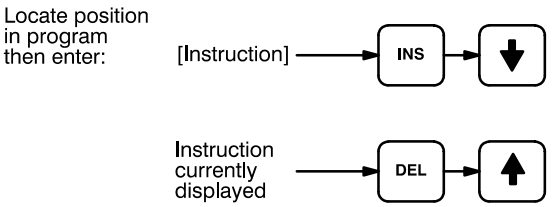
In PROGRAM mode, any instruction that is currently displayed can be deleted or another instruction can be inserted before it. These are not possible in RUN or MONITOR modes.

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 a 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 Sequence



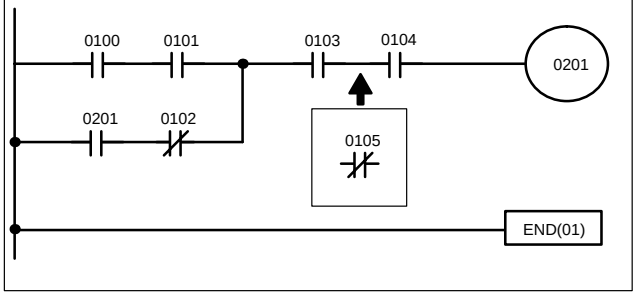
When an instruction is inserted or deleted, all addresses in Program Memory following the operation are adjusted automatically so that there are no blank addresses and no unaddressed instructions.

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

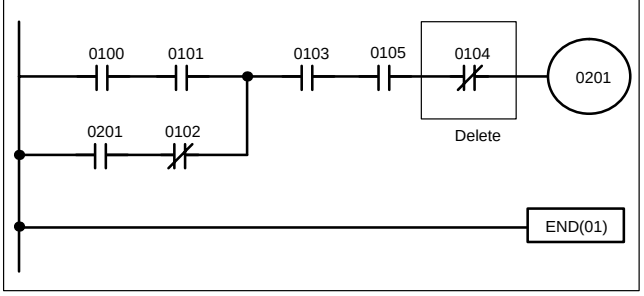
Original Program

Address	Instruction	Operands
0000	LD	0100
0001	AND	0101
0002	LD	0201
0003	AND NOT	0102
0004	OR LD	---
0005	AND	0103
0006	AND NOT	0104
0007	OUT	0201
0008	END(01)	---

Before Insertion:

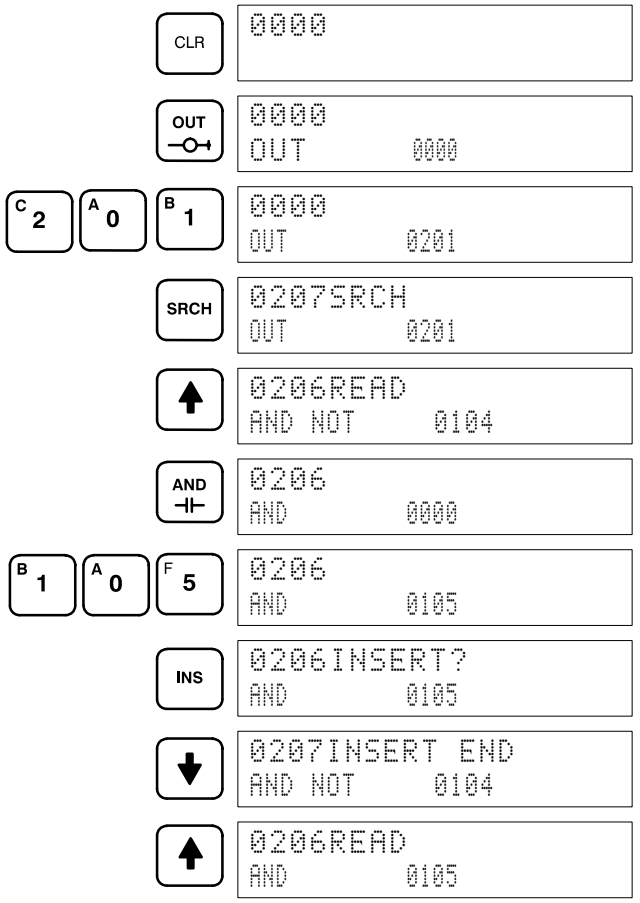


Before Deletion:



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

Inserting an Instruction



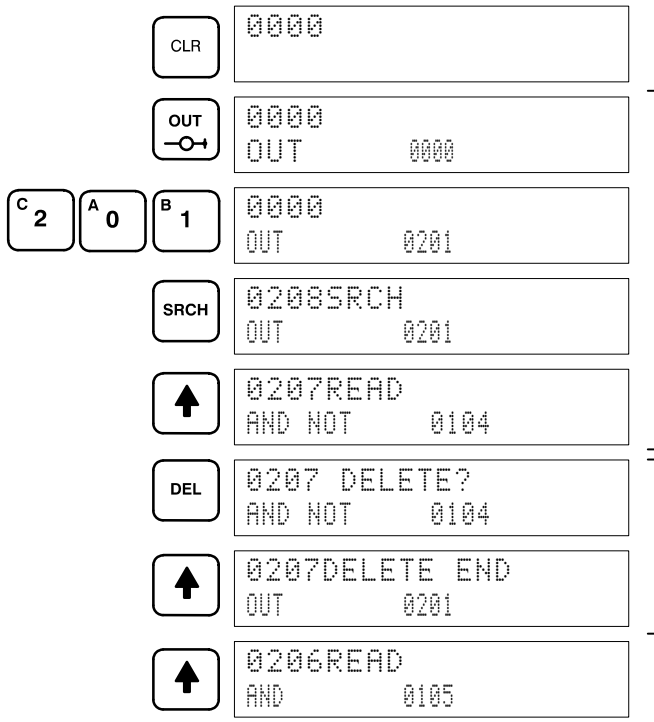
Find the address prior to the insertion point

Insert the instruction

Program After Insertion

Address	Instruction	Operands
0000	LD	0100
0001	AND	0101
0002	LD	0201
0003	AND NOT	0102
0004	OR LD	---
0005	AND	0103
0006	AND	0105
0007	AND NOT	0104
0008	OUT	0201
0009	END(01)	---

Deleting an Instruction



Find the instruction that requires deletion.

Program After Deletion

Address	Instruction	Operands
0000	LD	0100
0001	AND NOT	0101
0002	LD	0201
0003	AND NOT	0102
0004	OR LD	---
0005	AND	0103
0006	AND	0105
0007	AND NOT	0104
0008	OUT	0201

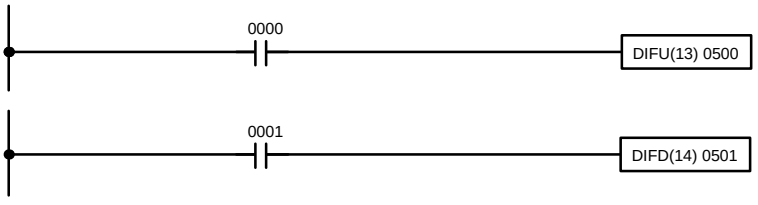
Confirm that this is the instruction to be deleted.

4-7 Controlling Bit Status

There are five instructions that can be used generally to control individual bit status. These are the OUTPUT or OUT, OUTPUT NOT or OUT NOT, DIFFERENTIATE UP, DIFFERENTIATE DOWN, and KEEP instructions. All of these instruction appear as the last instruction in an instruction line and take a bit address for an operand. Although details are provided in *5-6 Bit Control Instructions*, these instructions are described here because of their importance in most programs. Although these instructions are used to turn ON and OFF output bits in the IR area (i.e., to send or stop output signals to external devices), they are also used to control the status of other bits in the IR area or in other data areas.

4-7-1 DIFFERENTIATE UP and DIFFERENTIATE DOWN

DIFFERENTIATE UP (DIFU(13)) and DIFFERENTIATE DOWN (DIFD(14)) instructions are used to turn the operand bit ON for one cycle at a time. The DIFFERENTIATE UP turns ON the operand bit for one cycle after the execution condition when it goes from OFF to ON; the DIFFERENTIATE DOWN instruction turns ON the operand bit for one cycle after the execution condition when it goes from ON to OFF.



Address	Instruction	Operands
0000	LD	0000
0001	DIFU(13)	0500

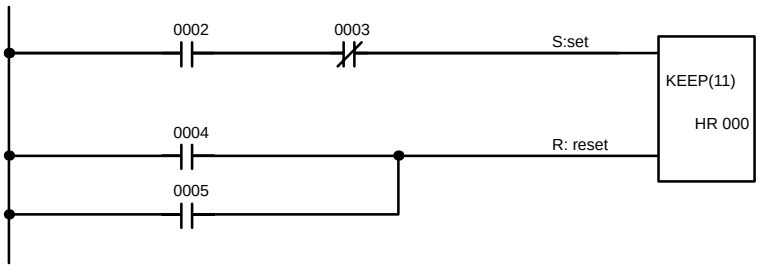
Address	Instruction	Operands
0000	LD	0001
0001	DIFD(14)	0501

Here, 0500 will be turned ON for one cycle after 0000 goes ON. The next time DIFU(13) 0500 is executed, 0500 will be turned OFF, regardless of the status of 0000. With the DIFFERENTIATE DOWN instruction, 0501 will be turned ON for one cycle after 0001 goes OFF (0501 will be kept OFF until then) and will be turned ON the next time DIFD(14) is executed.

4-7-2 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. 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 and the execution condition for the INTERLOCK instruction is ON.

In the following example, HR 000 will be turned ON when 0002 is ON and 0003 is OFF. HR 000 will then remain ON until either 0004 or 0005 turns ON.



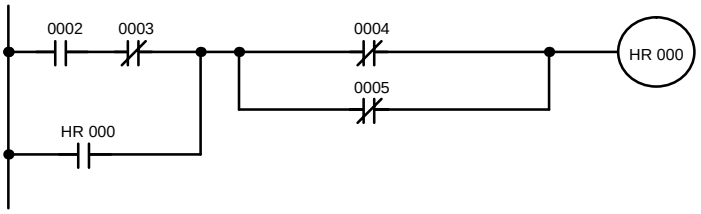
Address	Instruction	Operands
0000	LD	0002
0001	AND NOT	0003
0002	LD	0004
0003	OR	0005
0004	KEEP(11)	HR 000

4-7-3 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 in other bits occur. 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., HR 000 is turned OFF by turning ON both 0004 and 0005.



Address	Instruction	Operands
0000	LD	0002
0001	AND NOT	0003
0002	OR	HR 000
0003	AND NOT	0004
0004	OR NOT	0005
0005	OUT	HR 000

4-8 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 IR area that are not allocated as I/O bits, and certain unused bits in the AR 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

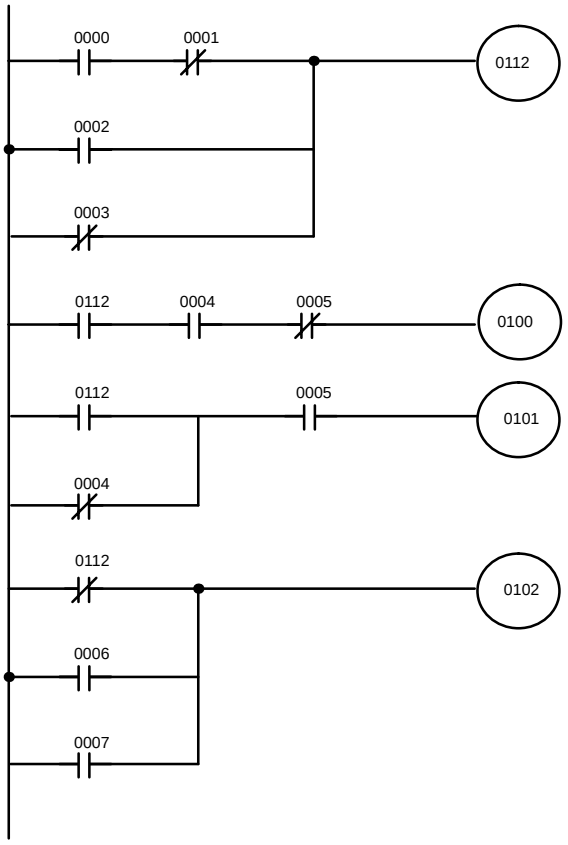
Examples given later in this subsection show two of the most common ways to employ work bits. These should act as a guide to the almost limitless number of ways in which the work bits can be used. Whenever difficulties arise in programming a control action, consideration should be given to work bits 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(10)). An example of the use of work words and bits with the SHIFT REGISTER instruction is provided in *5-12-1 SHIFT REGISTER - SFT(10)*.

Although they are not always specifically referred to as work bits, many of the bits used in the examples in *Section 5 Instruction Set* 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, IR 0000, IR 0001, IR 0002, and IR 0003 are combined in a logic block that stores the resulting execution condition as the status of IR 0112. IR 0112 is then combined with various other conditions to determine output conditions for IR 0100, IR 0101, and IR 0102, i.e., to turn the outputs allocated to these bits ON or OFF.

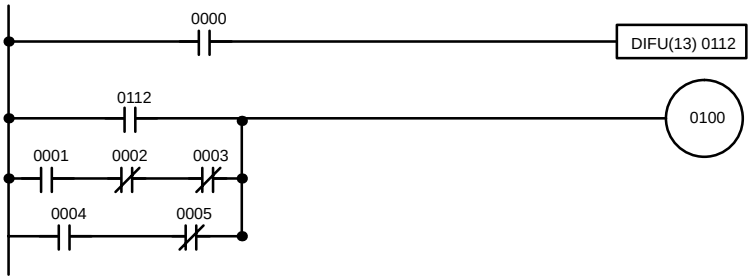


Address	Instruction	Operands
0000	LD	0000
0001	AND NOT	0001
0002	OR	0002
0003	OR NOT	0003
0004	OUT	0112
0005	LD	0112
0006	AND	0004
0007	AND NOT	0005
0008	OUT	0100
0009	LD	0112
0010	OR NOT	0004
0011	AND	0005
0012	OUT	0101
0013	LD NOT	0112
0014	OR	0006
0015	OR	0007
0016	OUT	0102

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, IR 0100 must be left on continuously as long as IR 0001 is ON and both IR 0002 and IR 0003 are OFF, or as long as IR 0004 is ON and IR 0005 is OFF. It must be turned ON for only one cycle each time IR 0000 turns ON (unless one of the preceding conditions is keeping it ON continuously).

This action is easily programmed by using IR 0112 as a work bit as the operand of the DIFFERENTIATE UP instruction (DIFU(13)). When IR 0000 turns ON, IR 0112 will be turned ON for one cycle and then be turned OFF the next cycle by DIFU(13). Assuming the other conditions controlling IR 0100 are not keeping it ON, the work bit IR 0112 will turn IR 0100 ON for one cycle only.



Address	Instruction	Operands
0000	LD	0000
0001	DIFU(13)	0112
0002	LD	0112
0003	LD	0001
0004	AND NOT	0002
0005	AND NOT	0003
0006	OR LD	---
0007	LD	0004
0008	AND NOT	0005
0009	OR LD	---
0010	OUT	0100

4-9 Programming Precautions

The number of conditions that can be used in series or parallel is unlimited. Therefore, use as many conditions as required to draw a clear diagram. Although very complicated diagrams can be drawn with instruction lines almost forming mazes, there must not be any conditions on instruction lines running vertically between two other instruction lines. Diagram A shown below, for example, is not possible, and should be redrawn as diagram B.

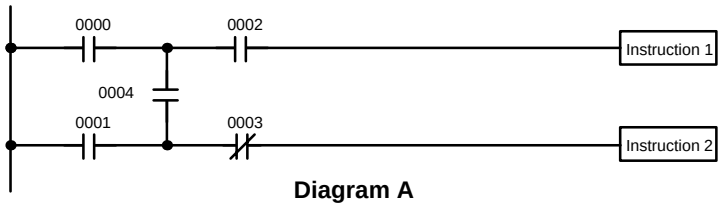


Diagram A

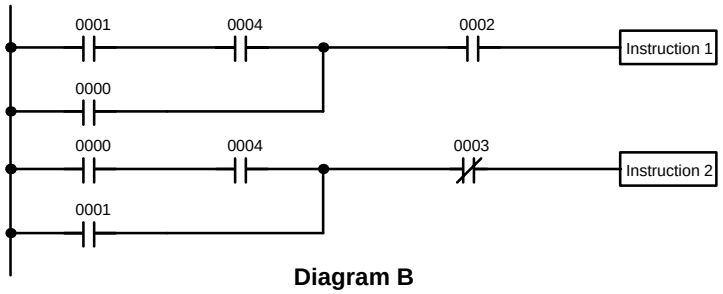


Diagram B

Address	Instruction	Operands
0000	LD	0001
0001	AND	0004
0002	OR	0000
0003	AND	0002
0004	Instruction 1	
0005	LD	0000
0006	AND	0004
0007	OR	0001
0008	AND NOT	0003
0009	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.

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 redrawn as diagram B. If an instruction must always be executed (e.g., if an output must always be kept ON while the program is being executed), the Always ON Flag (1813) in the SR area can be used.

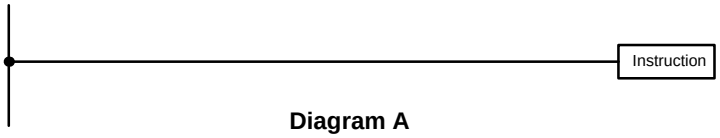


Diagram A

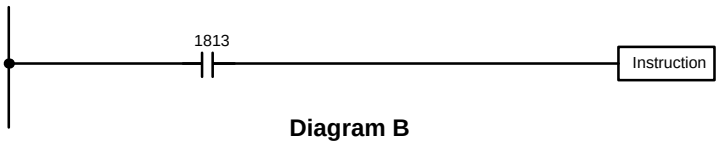


Diagram B

Address	Instruction	Operands
0000	LD	1813
0001	Instruction	

There are, however, a few exceptions to this rule, including the INTERLOCK CLEAR, JUMP END, 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. Refer to *Section 5 Instruction Set* for details.

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 *5-5-2 AND LOAD and OR LOAD* for more details and *4-6 Inputting, Modifying and Checking the Program* for further examples.

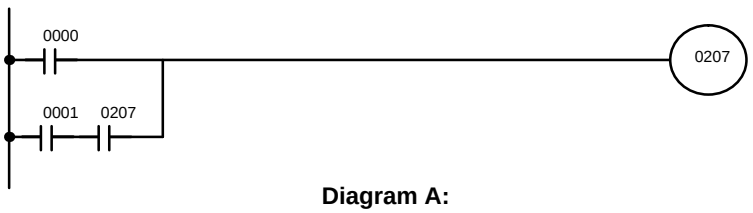


Diagram A:

Address	Instruction	Operands
0000	LD	0000
0001	LD	0001
0002	AND	0207
0003	OR LD	---
0004	OUT	0207

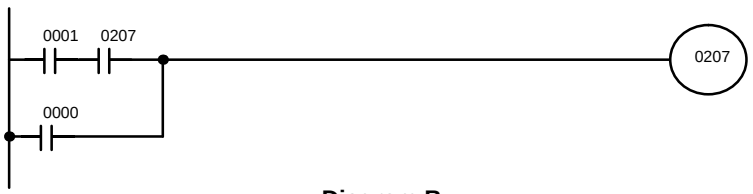


Diagram B:

Address	Instruction	Operands
0000	LD	0001
0001	AND	0207
0002	OR	0000
0003	OUT	0207

4-10 Program Execution

When program execution is started, the CPU cycles 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 any instruction lines branching from the first instruction line to other terminal instructions at the right.

Program execution is only one of the tasks carried out by the CPU as part of the cycle time. Refer to *Section 6 Program Execution Timing* for details.

SECTION 5
Instruction Set

5-1 Introduction
5-2 Notation
5-3 Instruction Format
5-4 Data Areas, Definer Values, and Flags
5-4-1 Coding Other Instructions
5-5 Ladder Diagram Instructions
5-5-1 LOAD, LOAD NOT, AND, AND NOT, OR, and OR NOT
5-5-2 AND LOAD and OR LOAD
5-6 Bit Control Instructions
5-6-1 OUTPUT and OUTPUT NOT – OUT and OUT NOT
5-6-2 DIFFERENTIATE UP and DIFFERENTIATE DOWN – DIFU(13) and DIFD(14)
5-6-3 KEEP – KEEP(11)
5-7 INTERLOCK and INTERLOCK CLEAR – IL(02) and ILC(03)
5-8 JUMP and JUMP END – JMP(04) and JME(05)
5-9 END – END(01)
5-10 NO OPERATION – NOP(00)
5-11 Timer and Counter Instructions
5-11-1 TIMER – TIM
5-11-2 HIGH-SPEED TIMER – TIMH(15)
5-11-3 Analog Timer Unit
5-11-4 COUNTER – CNT
5-11-5 REVERSIBLE COUNTER – CNTR(12)
5-11-6 HIGH-SPEED DRUM COUNTER – HDM(61)
5-11-7 REVERSIBLE DRUM COUNTER – RDM(60)
5-12 Data Shifting
5-12-1 SHIFT REGISTER – SFT(10)
5-12-2 REVERSIBLE SHIFT REGISTER – SFTR(84)
5-12-3 WORD SHIFT – WSFT(16)
5-13 Data Movement
5-13-1 MOVE – MOV(21)
5-13-2 MOVE NOT – MVN(22)
5-14 DATA COMPARE – CMP(20)
5-15 Data Conversion
5-15-1 BCD-TO- BINARY – BIN(23)
5-15-2 BINARY-TO-BCD – BCD(24)
5-15-3 4-TO-16 DECODER – MLPX(76)
5-15-4 16-TO-4 ENCODER – DMPX(77)
5-16 BCD Calculations
5-16-1 BCD ADD – ADD(30)
5-16-2 BCD SUBTRACT – SUB(31)
5-16-3 BCD MULTIPLY – MUL(32)
5-16-4 BCD DIVIDE – DIV(33)
5-16-5 SET CARRY – STC(40)
5-16-6 CLEAR CARRY – CLC(41)
5-17 Subroutines
5-17-1 SUBROUTINE DEFINE and SUBROUTINE RETURN SBN(92)/RET(93)
5-17-2 SUBROUTINE ENTRY – SBS(91)
5-18 Step Instructions
5-18-1 STEP DEFINE and STEP START – STEP(08)/SNXT(09)
5-19 Special Instructions
5-19-1 I/O REFRESH – IORF(97)
5-19-2 END WAIT – ENDW(62)
5-19-3 NOTATION INSERT – NETW(63)

5-1 Introduction

The K-type PCs have large programming instruction sets that allow for easy programming of complicated control processes. This section explains each instruction individually and provides the ladder diagram symbol, data areas, and flags used with each. Basic application examples are also provided as required in describing the instructions.

The many instructions provided by the K-type PCs are described in following subsections by instruction group. These groups include Ladder Diagram Instructions, Bit Control Instructions, Timer and Counter Instructions, Data Shifting, Data Movement, Data Comparison, Data Conversion, BCD Calculations, Subroutines, Step Instructions, and Special Instructions.

Some instructions, such as timer and counter instructions, are used to control execution of other instructions, e.g., a TIM completion flag might be used to turn ON a bit when the time period set for the timer has expired. Although these other instructions are often used to control output bits through the OUTPUT instruction, they can be used to control execution of other instructions as well. The OUTPUT instructions used in examples in this manual can therefore generally be replaced by other instructions to modify the program for specific applications other than controlling output bits directly.

5-2 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 NOT instruction, AND NOT. If you're not sure of what instruction a mnemonic is used for, refer to *Appendix B 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. A table of instructions listed in order of function codes is also provided in *Appendix B Programming Instructions and Execution Times*.

5-3 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 performed. 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 four 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 be used. Examples of definers are TC numbers, which are used in timer and counter instructions to create timer and counters, and jump numbers, which define which JUMP instruction is

paired with which JUMP END instruction. Bit operands are also contained in the same word as the instruction itself, although these are not considered definers.

5-4 Data Areas, Definer Values, and Flags

Each instruction is introduced with the ladder diagram symbol(s), the data areas that can be used with any operand(s), and the values that can be used for definers. With the data areas is also specified 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 a specified data area are necessarily allowed in an operand, e.g., if an operand requires two words, the last word in a data area cannot be designated because all words for a single operand must be in the same data area. Unless a limit is specified, any bit/word in the area can be used. Specific limitations for operands and definers are specified in a *Limitations* subsection. Refer to *Section 3 Memory Areas* for addressing conventions and the addresses of flags and control bits.

⚠ Caution The IR and SR areas are considered as separate areas and both are not necessarily allowed for an operand just because one of them is. The border between the IR and SR area can, however, be crossed for a single operand, i.e., the last bit in the IR area may be specified for an operand that requires more than one word as long as the SR area is also allowed for that operand.

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

Abbreviation	Name	Bit
ER	Instruction Execution Error flag	1903
CY	Carry flag	1904
EQ	Equals flag	1906
GR	Greater Than flag	1905
LE	Less Than flag	1907

ER is the flag most often 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 for any instruction if operands are not input within established parameters. Instructions are not executed when ER is ON. A table of instructions and the flags they affect is provided in *Appendix D Error and Arithmetic Flag Operation*.

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

5-4-1 Coding Other Instructions

When combining other right-hand instructions with ladder diagram instructions, they would appear in the same place as the OUTs used in the example in the preceding section. Many of these instructions, however, require more than one word to code.

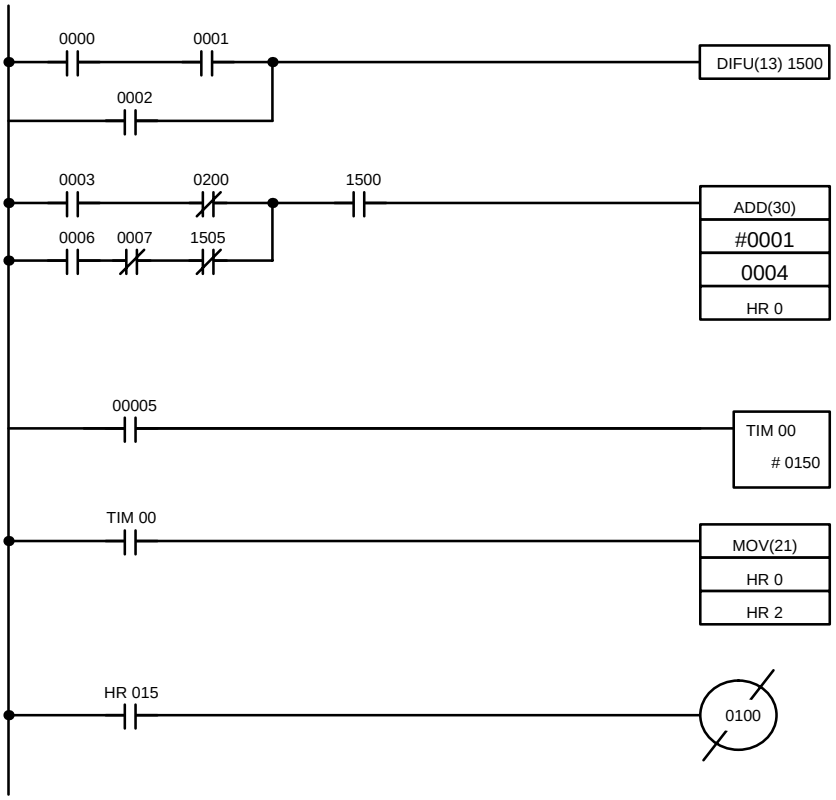
The first word of any instruction defines the instruction and provides any definers and sometimes bit operands required by the instruction. All other operands (i.e., operand words) are placed in words after the instruction word, one operand to a word, in the same order as these appear in the ladder symbol for the instruction. Although the SV for TIM and CNT are written to the left of the symbol on the same line as the instruction, these are the only instructions for which one line in the ladder symbol must be coded as two words (i.e., two lines) in the mnemonic code. Also the TC number for TIMH(15) is placed on a second line even though it is part of the instruction word. For all other instructions, each line of the ladder diagram will go into one word of mnemonic code.

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

If an IR or SR address is used in the data column, the left side of the column is left blank. If any other data area is used, the data area abbreviation is placed on the left side and the address is place on the right side. If a constant is 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. Remember, TR 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.

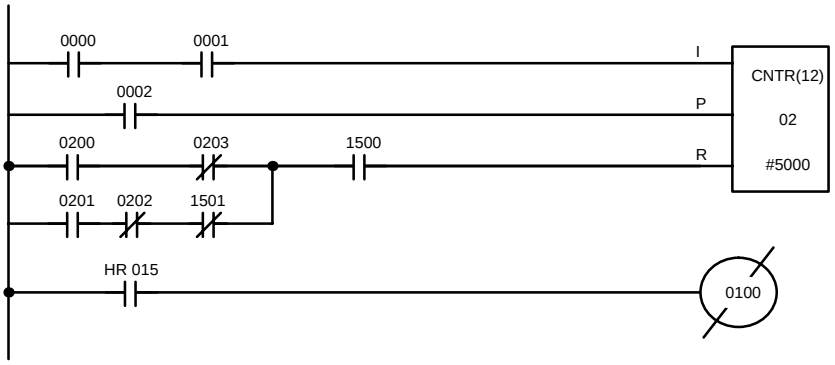
The following diagram and corresponding mnemonic code illustrate the points described above.



Address	Instruction	Operands	
0000	LD		0000
0001	AND		0001
0002	OR		0002
0003	DIFU(13)		1500
0004	LD		0003
0005	AND NOT		0200
0006	LD		0006
0007	AND NOT		0007
0008	AND NOT		1505
0009	OR LD		
0010	AND		1500
0011	ADD(30)		
		#	0001
			0004
		HR	0
0012	LD		0005
0013	TIM		00
		#	0150
0014	LD	TIM	00
0015	MOV(21)		
		HR	0
		HR	2
0016	LD	HR	015
0017	OUT NOT		0100

Multiple Instruction Lines

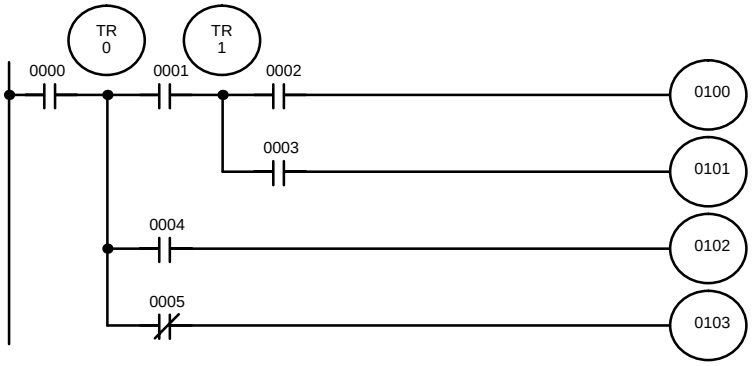
If a right-hand instruction requires multiple instruction lines, all of the lines for the instruction are coded before the right-hand instruction. Each of the lines for the instruction are coded starting with LD or LD NOT to form 'logic blocks' that are combined by the right-hand instruction. An example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



Address	Instruction	Operands
0000	LD	0000
0001	AND	0001
0002	LD	0002
0003	LD	0200
0004	AND NOT	0203
0005	LD	0201
0006	AND NOT	0202
0007	AND NOT	1501
0008	OR LD	
0009	AND	1500
0010	CNTR(12)	
		02
		# 5000
0011	LD	HR 015
0012	OUT NOT	0100

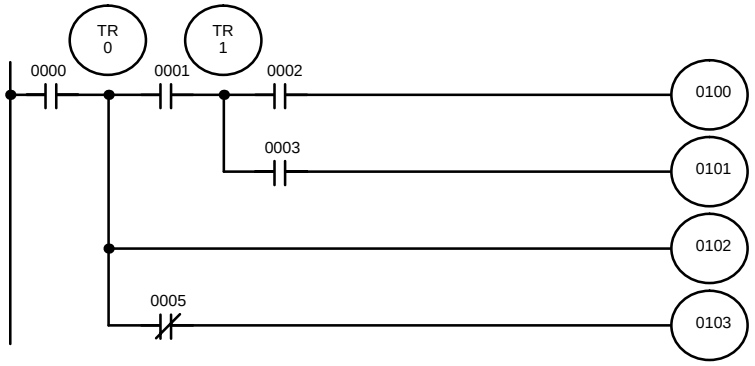
TR Bits

TR bits in a program are used to output (OUT) the execution condition at the branching point and then to load back (LD) the execution condition when it is required after returning to the branch lines. Within any one instruction block, OUT cannot be used with the same TR address. The same TR address can, however, be used with LD as many times as required. The following example shows an instruction block using two TR bits. TR 1 is used in LD once; TR 0, twice.



Address	Instruction	Operands
0000	LD	0000
0001	OUT	TR 0
0002	AND	0001
0003	OUT	TR 1
0004	AND	0002
0005	OUT	0100
0006	LD	TR 1
0007	AND	0003
0008	OUT	0101
0009	LD	TR 0
0010	AND	0004
0011	OUT	0102
0012	LD	TR 0
0013	AND NOT	0005
0014	OUT	0103

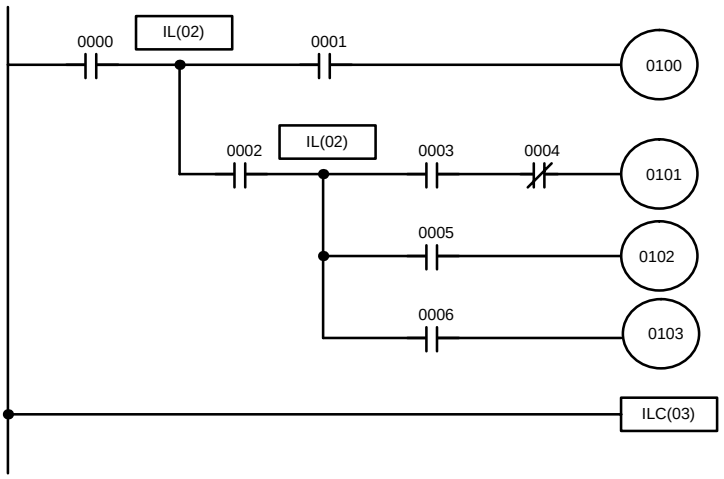
If the condition assigned 0004 was not in the diagram, the second LD using TR 0 would not be necessary because OUT with 0102 and the AND NOT with 0005 both require the same execution condition, i.e., the execution condition stored in TR 0. The diagram and mnemonic code for this program are shown below.



Address	Instruction	Operands
0000	LD	0000
0001	OUT	TR 0
0002	AND	0001
0003	OUT	TR 1
0004	AND	0002
0005	OUT	0100
0006	LD	TR 1
0007	AND	0003
0008	OUT	0101
0009	LD	TR 0
0010	OUT	0102
0011	AND NOT	0005
0012	OUT	0103

Interlocks

When coding IL(02) and ILC(03), the mnemonic code will be the same regardless of whether the instruction is drawn as branching instruction lines or whether IL(02) is placed on its own instruction line. If drawn as branching instruction lines, each branch line is coded as if it were connected to the bus bar, i.e., the first condition on each branch line corresponds to a LD or LD NOT instruction.



Address	Instruction	Operands
0000	LD	0000
0001	IL(02)	
0002	LD	0001
0003	OUT	0100
0004	LD	0002
0005	IL(02)	
0006	LD	0003
0007	AND NOT	0004
0008	OUT	0101
0009	LD	0005
0010	OUT	0102
0011	LD	0006
0012	OUT	0103
0013	ILC(03)	

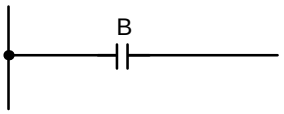
5-5 Ladder Diagram Instructions

Ladder diagram instructions include ladder instructions and logic block instructions. Ladder instructions correspond to the conditions on the ladder diagram. Logic block instructions are used to relate more complex parts of the diagram that cannot be programmed with ladder instructions alone.

5-5-1 LOAD, LOAD NOT, AND, AND NOT, OR, and OR NOT

LOAD – LD

Ladder Symbol

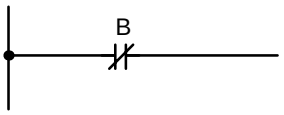


Operand Data Areas

B: Bit
IR, SR, HR, TC, TR

LOAD NOT – LD NOT

Ladder Symbol

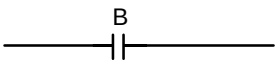


Operand Data Areas

B: Bit
IR, SR, HR, TC, TR

AND – AND

Ladder Symbol

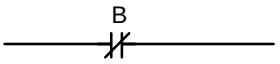


Operand Data Areas

B: Bit
IR, SR, HR, TC, TR

AND NOT – AND NOT

Ladder Symbol

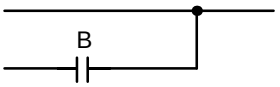


Operand Data Areas

B: Bit
IR, SR, HR, TC, TR

OR – OR

Ladder Symbol

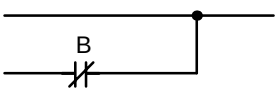


Operand Data Areas

B: Bit
IR, SR, HR, TC, TR

OR NOT – OR NOT

Ladder Symbol



Operand Data Areas

B: Bit
IR, SR, HR, TC, TR

Limitations

There is no limit in the number of any of these instructions or 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 *Section 4 Writing and Inputting the Program*, the status of the bits assigned to each instruction determines the execution conditions for all other instructions. Each of these instructions can be used as many times and a bit address 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 *Section 4 Writing and Inputting the Program*.

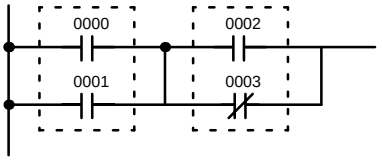
Flags

There are no flags affected by these instructions.

5-5-2 AND LOAD and OR LOAD

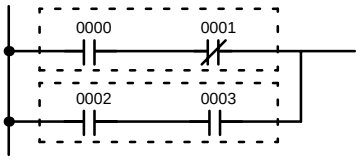
AND LOAD – AND LD

Ladder Symbol



OR LOAD – OR LD

Ladder Symbol



Description

When the above 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.

AND LD and OR LD instruction are not necessary to draw ladder diagrams, nor are they necessary when inputting ladder diagrams directly, as is possible from the GPC. They are required, however, to convert the program to and input it in mnemonic form.

In order to reduce the number of programming instruction required, a basic understanding of logic block instructions is required.

Flags

There are no flags affected by these instructions.

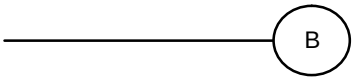
5-6 Bit Control Instructions

There are five instructions that can be used generally to control individual bit status. These are OUT, OUT NOT, DIFU(13), DIFD(14), and KEEP(11). These instructions are used to turn bits ON and OFF in different ways.

5-6-1 OUTPUT and OUTPUT NOT – OUT and OUT NOT

OUTPUT – OUT

Ladder Symbol

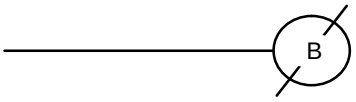


Operand Data Areas

B: Bit
IR, HR, TR

OUTPUT NOT – OUT NOT

Ladder Symbol



Operand Data Areas

B: Bit
IR, HR, TR

Limitations

Any output bit can be used in only one instruction that controls its status. See *3-3 Internal Relay (IR) Area* 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 a ON execution condition, and turns OFF the designated bit for an OFF execution condition. OUT with a TR bit appears at a branching point rather than at the end of an instruction line.

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 when a complex set of conditions can be used to control the status of a single work bit, and then that work bit can be used to control other instructions.

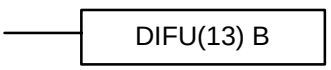
The length of time that a bit is ON or OFF can be controlled by combining the OUT or OUT NOT with TIM. Refer to Examples under *5-11-1 TIMER – TIM* for details.

Flags

There are no flags affected by these instructions.

5-6-2 DIFFERENTIATE UP and DIFFERENTIATE DOWN – DIFU(13) and DIFD(14)

Ladder Symbol



Operand Data Areas

B: Bit
IR, HR

Ladder Symbol

Operand Data Areas

B: Bit
IR, HR

Limitations

Any output bit can be used in only one instruction that controls its status. See *3-3 Internal Relay (IR) Area* for details.

Description

DIFU(13) and DIFD(14) are used to turn the designated bit ON for one cycle only.

Whenever executed, DIFU(13) compares its current execution with the previous execution condition. If the previous execution condition was OFF and and current one is ON, DIFU(13) will turn ON the designated bit. If the previous execution condition was ON and the current execution condition is either ON or OFF, DIFU(13) will turn the designated bit OFF or do nothing (i.e., if the designated bit is already OFF). The designated bit will thus never be ON for longer than one cycle assuming it is executed each cycle (see Precautions, below).

Whenever executed, DIFD(14) compares its current execution with the previous execution condition. If the previous execution condition was ON and the current one is OFF, DIFD(14) will turn ON the designated bit. If the previous execution condition was OFF and the current execution condition is either ON or OFF, DIFD(14) will turn the designated bit OFF or do nothing (i.e., if the designated bit is already OFF). The designated bit will thus never be ON for longer than one cycle.

These instructions are used when a single-cycle execution of a particular instruction is desired. Examples of these are shown below.

DIFU(13) and DIFD(14) operation can be tricky when used in programming between IL and ILC, between JMP and JME, or in subroutines. Refer to *5-7 INTERLOCK and INTERLOCK CLEAR – IL(02) and ILC(03)* and *5-8 JUMP and JUMP END – JMP(04)/JME(05)* for details. A total of 48 DIFU(13)/DIFD(14) can be used in a program. If more than 48 are used in a program only the first 48 will be executed and all others will be ignored. DIFU(13)/DIFD(14) are useful when used in conjunction with CMP(20) or MOV(21), see *Example* below.

Flags

There are no flags affected by these instructions.

Example

In diagram A, below, CMP(20) will compare the contents of the two operand words (HR 1 and DM 00) whenever it is executed with an ON execution condition and set the arithmetic flags (GR, EQ, and LE) accordingly. If the execution condition remains ON, flag status may be changed each cycle if the contents of one or both operands change. Diagram B, however, shows how DIFU(13) can be used to ensure that CMP(20) is executed only once each time the desired execution condition goes ON.

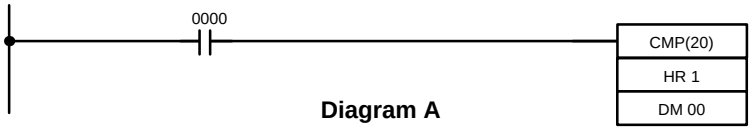


Diagram A

Address	Instruction	Operands
0000	LD	0000
0001	CMP(20)	
		HR 1
		DM 00

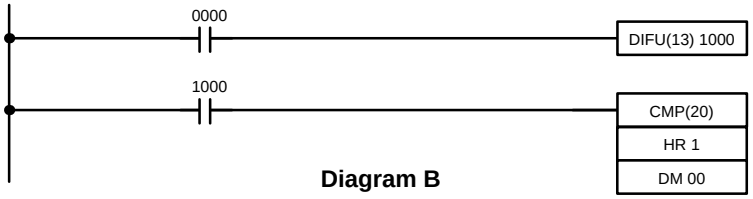
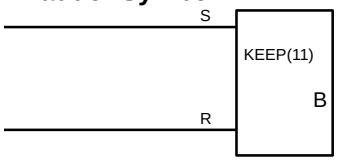


Diagram B

Address	Instruction	Operands
0000	LD	0000
0001	DIFU(13)	1000
0002	LD	1000
0003	CMP(20)	
		HR 1
		DM 00

5-6-3 KEEP – KEEP(11)

Ladder Symbol



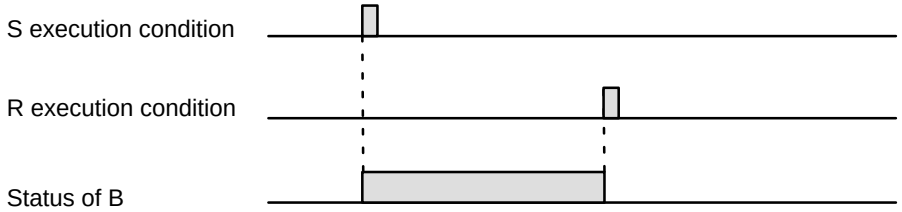
Operand Data Areas

B: Bit
IR, HR

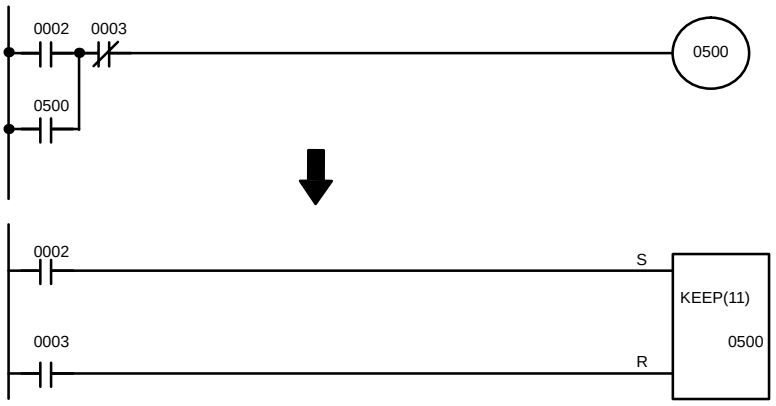
Description

KEEP(11) 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(11) 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(11) bit status is shown below.



Notice that KEEP(11) operates like a self-maintaining bit. The following two diagrams would function identically, though the one using KEEP(11) requires one less instruction to program and would maintain status even in an interlocked program section.



Address	Instruction	Operands
0000	LD	0002
0001	OR	0500
0002	AND NOT	0003
0003	OUT	0500

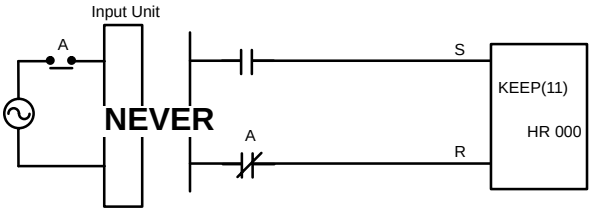
Address	Instruction	Operands
0000	LD	0002
0001	LD	0003
0002	KEEP(11)	0500

Flags

There are no flags affected by this instruction.

Precautions

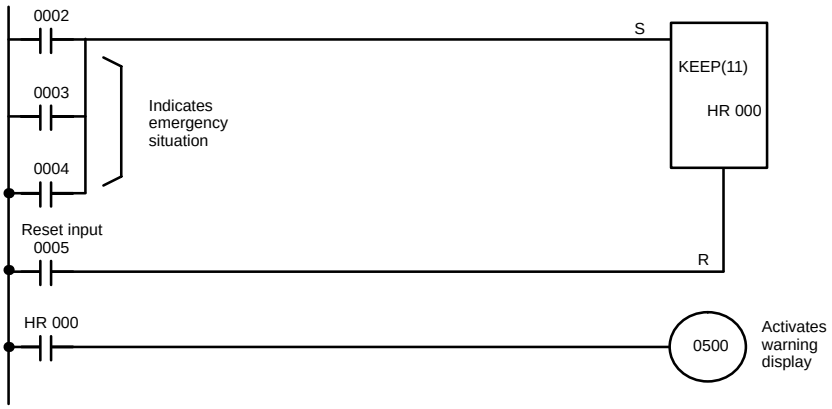
Never use an input bit in an normally closed condition on the reset (R) for KEEP(11) when the input device uses an AC power supply. The delay in shutting down the PC's DC power supply (relative to the AC power supply to the input device) can cause the designated bit of KEEP(11) to be reset. This situation is shown below.



Bits used in KEEP are not reset in interlocks. Refer to the *5-7 INTERLOCK and INTERLOCK CLEAR – IL(02) and ILC(03*) for details.

Example

If a HR bit is used, bit status will be retained even during a power interruption. KEEP(11) 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 system error. Bit 0005 would be turned ON to reset the warning display. HR 000, which is turned ON for any of the three bits which indicates emergency situation, is used to turn ON the warning indicator through 0500.

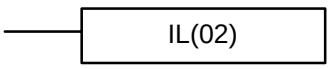


Address	Instruction	Operands
0000	LD	0002
0001	OR	0003
0002	OR	0004
0003	LD	0005
0004	KEEP(11)	HR 000
0005	LD	HR 000
0006	OUT	0500

KEEP(11) can also be combined with TIM to produce delays in turning bits ON and OFF. Refer to *5-11-1 TIMER – TIM* for details.

5-7 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 enable branching in the same way as can be achieved with TR bits, but treatment of instructions between IL(02) and ILC(03) differs from that with TR bits when the execution condition for IL(02) is OFF. If the execution condition of IL(02) is ON, the program will be executed as written, with an ON execution condition used to start each instruction line from the point where IL(02) is located through ILC(03).

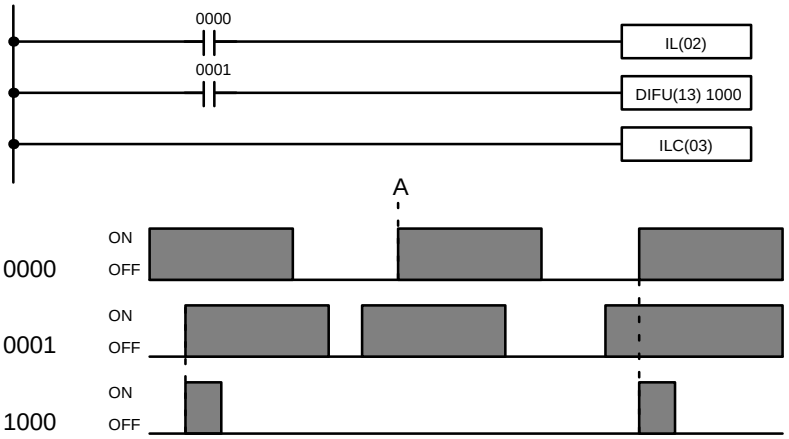
If the execution condition for IL(02) condition is OFF, the interlocked section between IL(02) and ILC(03) will be treated as shown in the following table:

Instruction	Treatment
OUT and OUT NOT	Designated bit turned OFF.
TIM and TIMH(15)	Reset.
CNT, CNTR(12)	PV maintained.
KEEP(11)	Bit status maintained.
DIFU(13) and DIFD(14)	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(13) and DIFD(14) in Interlocks

Changes in the execution condition for a DIFU(13) or DIFD(14) are not recorded if the DIFU(13) or DIFD(14) is in an interlocked section and the execution condition for the IL(02) is OFF. When DIFU(13) or DIFD(14) is executed in an interlocked section immediately after the execution condition for the IL(02) has gone ON, the execution condition for the DIFU(13) or DIFD(14) 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 0000 is OFF. Notice that 1000 is not turned ON at the point labeled A even though 0001 has turned OFF and then back ON.



Address	Instruction	Operands
0000	LD	0000
0001	IL(02)	
0002	LD	0001
0003	DIFU(13)	1000
0004	ILC(03)	

Precautions

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

Although as many IL(02) as necessary can be used with one ILC(03), ILC(03) cannot be used consecutively without at least one IL(02) in between, i.e., nesting is not possible. Whenever a ILC(03) is executed, all interlocks are cleared.

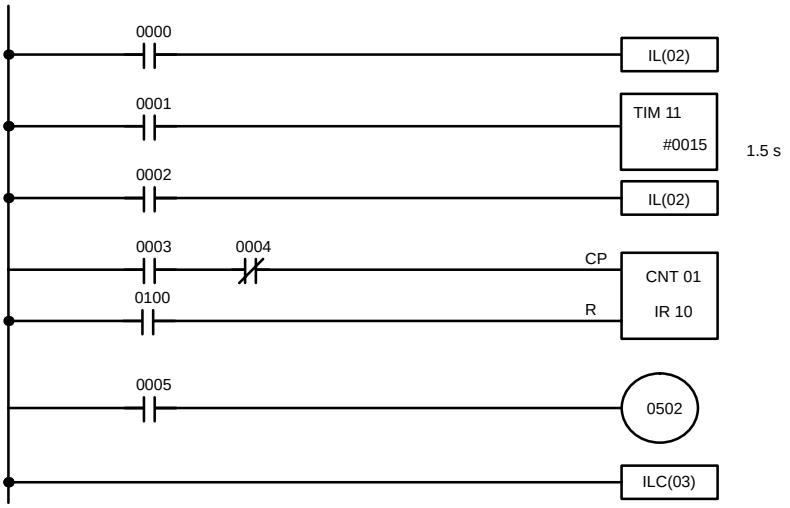
When more than one IL(02) is used with a single ILC(03), an error message will appear when the program check is performed, but execution will proceed normally.

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
0000	LD	0000
0001	IL(02)	
0002	LD	0001
0003	TIM	11
		# 0015
0004	LD	0002
0005	IL(02)	
0006	LD	0003
0007	AND NOT	0004
0008	LD	0100
0009	CNT	01
		10
0010	LD	0005
0011	OUT	0502
0012	ILC(03)	

When the execution condition for the first IL(02) is OFF, TIM 11 will be reset to 1.5 s, CNT 01 will not be changed, and 0502 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 11 will be executed according to the status of 0001, CNT 01 will not be changed, and 0502 will be turned OFF. When the execution conditions for both the IL(02) are ON, the program will execute as written.

5-8 JUMP and JUMP END – JMP(04) and JME(05)

Ladder Symbols **Definer Values**

JMP(04) N

N: Jump number
(00 to 08)

Ladder Symbols **Definer Values**

JME(05) N

N: Jump number
(00 to 08)

Limitations

Jump numbers 01 through 08 may be used only once in JMP(04) and once in JME(05), i.e., each can be used to define one jump only. Jump number 00 can be used as many times as desired.

Description

JMP(04) is always used in conjunction with JME(05) to create jumps, i.e., to skip from one point in a ladder diagram to another point. JMP(04) defines the point from which the jump will be made; JME(05) defines the destination of the jump. When the execution condition for JMP(04) in ON, no jump is made and the program is executed as written. When the execution condition for JMP(04) is OFF, a jump is made to the JME(05) with the same jump number and the instruction following JME(05) is executed next.

If the jump number for JMP(04) is between 01 and 08, jumps, when made, will go immediately to JME(05) without executing any instructions in between. The status of timers, counters, bits used in OUT, bits used in OUT NOT, and all other status controlled by the instructions between JMP(04) and JMP(05)

will not be changed. Each of these jump numbers can be used to define one jump. Because all of instructions between JMP(04) and JME(05) are skipped, jump numbers 01 through 08 can be used to reduce cycle time.

If the jump number for JMP(04) is 00, the CPU will look for the next JME(05) with a Jump number of 00. To do so, it must search through the program, causing a longer cycle time than for other jumps (i.e., longer when the execution condition is OFF). The status of timers, counters, bits used in OUT, bits used in OUT NOT, and all other status controlled by the instructions between JMP(04) 00 and JMP(05) 00 will not be changed. Jump number 00 can be used as many times as desired. A jump from JMP(04) 00 will always go to the next JME(05) 00 in the program. It is thus possible to use JMP(04) 00 consecutively and match them all with the same JME(05) 00. It makes no sense, however, to used JME(05) 00 consecutively, because all jumps made to them will end at the first JME(05) 00.

DIFU(13) and DIFD(14) in Jumps

Although DIFU(13) and DIFD(14) are designed to turn ON the designated bit for one cycle, they will not necessarily do so when written between JMP(04) and JMP (05). Once either DIFU(13) or DIFD(14) has turned ON a bit, it will remain ON until the next time DIFU(13) or DIFD(14) is executed again. In normal programming, this means the next cycle. In a jump, it means the next time the jump from JMP(04) to JME(05) is not made, i.e., if a bit is turned ON by DIFU(13) or DIFD(14) and then a jump is made that skips the DIFU(13) or DIFD(14), the designated bit will remain ON until the next time the execution condition for the JMP(04) controlling the jump is ON.

Precautions

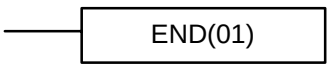
When JMP(04) and JME(05) are not used in pairs, an error message will appear when the program check is performed. Although this message also appears if JMP(04) 00 and JME(05) 00 are not used in pairs, the program will execute properly as written.

Flags

There are no flags affected by these instructions.

5-9 END – END(01)

Ladder Symbol



Description

END(01) is required as the last instruction in any program. No instruction 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 ER, CY, GR, EQ, and LE.

5-10 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 the next instruction is moved to. 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).

5-11 Timer and Counter Instructions

TIM and TIMH are decrementing ON-delay timer instructions which require a TC number and a set value (SV).

CNT is a decrementing counter instruction and CNTR is a reversible 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) and a reset.

HDM(61) is used to create a 2-kHz high-speed drum counter; RDM(60) is used to create a reversible drum counter. RDM(60) cannot be used to create a high-speed counter. If you require a high-speed counter, use HDM(61).

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 other than timer and counter instructions.

TC numbers run from 00 through 47. No prefix is required when using a TC number as a definer in a timer or counter instruction. Once defined as a timer, 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.

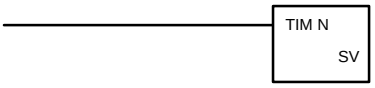
TC numbers can be designated for operands that require bit data or for operands that require word data. When designated as an operand that requires bit data, the TC number accesses a bit that functions as a “completion flag” that indicates when the time/count has expired, i.e., the bit, which is normally 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(20) or any other instruction for which the TC area is allowed by designating the TC number used to define that timer or counter to access the memory location that holds the PV.

Note that “TIM 00” is used to designate the Timer instruction defined with TC number 00, to designate the completion flag for this timer, and to designate the PV of this timer. The meaning of the term in context should be clear, i.e., the first is always an instruction, the second is always a bit operand, and the third is always a word operand. The same is true of all other TC numbers prefixed with TIM or CNT. In explanations of ladder diagrams, the completion flag and PV accessed through a TC number are generally called the completion flag or the PV of the instruction (e.g., the completion flag of TIM 00 is the completion flag of TC number 00, which has been defined using TIM).

An SV can be input as a constant or as a word address in a data area. If an IR area word assigned to an Input Unit is designated as the word address, the Input Unit can be wired so that the SV can be set externally through thumbwheel switches or similar devices. Timers and counter wired in this way can be set externally only during RUN or MONITOR mode. All SVs, including those set externally, must be in BCD.

5-11-1 TIMER – TIM

Ladder Symbol



Definer Values

N: TC number
(00 through 47)

Operand Data Areas

SV: Set value (word, BCD)
IR, HR, #

Limitations

SV may be between 000.0 and 999.9 seconds. The decimal point of SV is not input.

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

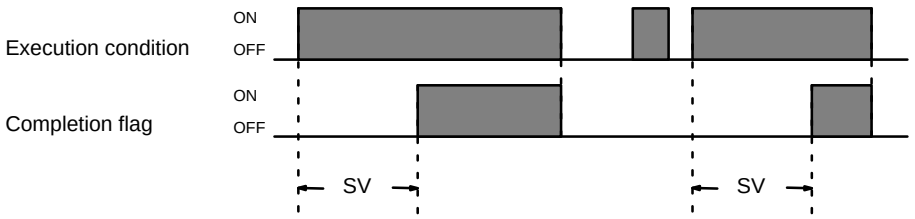
TC 00 through TC 47 should not be used in TIM if they are required for TIMH(15). Refer to *5-11-2 HIGH-SPEED TIMER – TIMH(15)* for details.

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 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, SR area clock pulse bits can be counted to produce timers using CNT. Refer to *5-11-4 COUNTER – CNT* for details.

Program execution will continue even if a non-BCD SV is used, but timing will not be accurate.

Flags

ER: SV is not in BCD.

Examples

All of the following examples use OUT in diagrams that would generally be used to control output bits in the IR area. 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, one set with a constant and one set via input word 01. Here, 0200 will be turned ON 15 seconds after 0000 goes ON and stays ON for at least 15 seconds. When 0000 goes OFF, the timer will be reset and 0200 will be turned OFF. When 0001 goes ON, TIM 01 is started from the SV provided through IR word 01. Bit 0201 is also turned ON when 0001 goes ON. When the SV in 01 has expired, 0201 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
0000	LD	0000
0001	TIM	00
		# 0150
0002	LD	TIM 00
0003	OUT	0200
0004	LD	0001
0005	TIM	01
		01
0006	AND NOT	TIM 01
0007	OUT	0201

Example 2: Extended Timers

Timers operating longer than 999.9 seconds can be formed in two ways. One is by programming 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
0000	LD	0000
0001	TIM	01
		# 9000
0002	LD	TIM 01
0003	TIM	02
		# 9000
0004	LD	TIM 02
0005	OUT	0200

In this example, 0200 will be turned ON 30 minutes after 0000 goes ON.

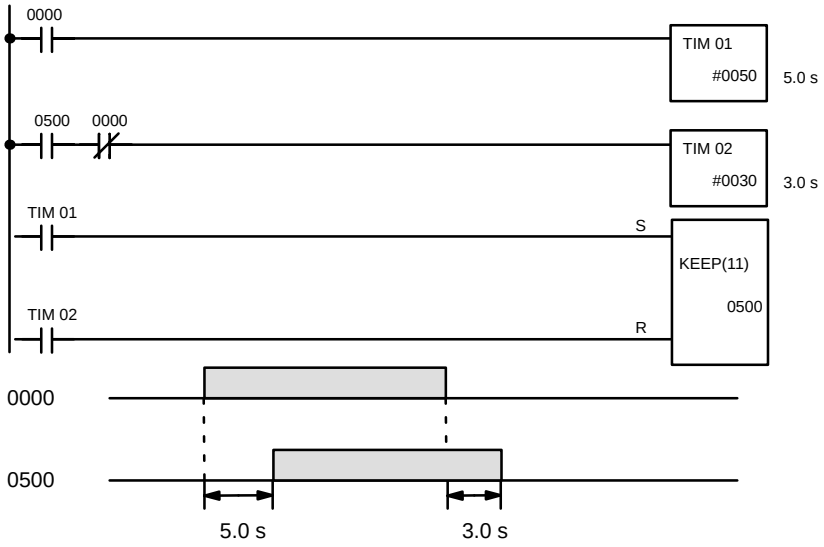
TIM can also be combined with CNT or CNT can be used to count SR area clock pulse bits to produce longer timers. An example is provided in *5-11-4 COUNTER – CNT*.

Example 3: ON/OFF Delays

TIM can be combined with KEEP(11) to delay turning a bit ON and OFF in reference to a desired execution condition. KEEP(11) is described in *5-6-3 KEEP – KEEP(11)*.

To create delays, the completion flags for two timers are used to determine the execution conditions for setting and resetting the bit designated for KEEP(11). The bit whose manipulation is to be delayed is used in KEEP(11). Turning ON and OFF the bit designated for KEEP(11) is thus delayed by the SV for the two timers. The two SV could naturally be the same if desired.

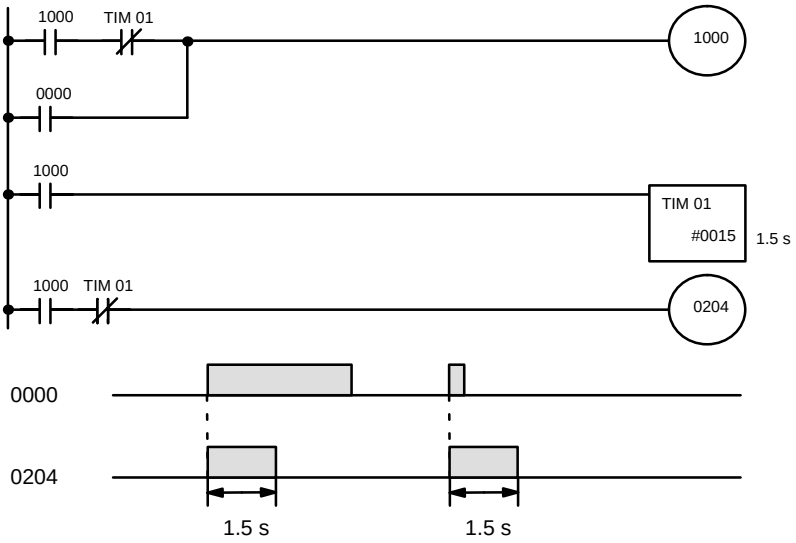
In the following example, 0500 would be turned ON 5.0 seconds after 0000 goes ON and then turned OFF 3.0 seconds after 0000 goes OFF. It is necessary to use both 0500 and 0000 to determine the execution condition for TIM 02; 0000 in an normally closed condition is necessary to reset TIM 02 when 0000 goes ON and 0500 is necessary to activate TIM 02 when 0000 goes OFF, setting 0500 by resetting TIM 01.



Address	Instruction	Operands
0000	LD	0000
0001	TIM	01
		# 0050
0002	LD	0500
0003	AND NOT	0000
0004	TIM	02
		# 0030
0005	LD	TIM 01
0006	LD	TIM 02
0007	KEEP(11)	0500

Example 4: One-shot Bits

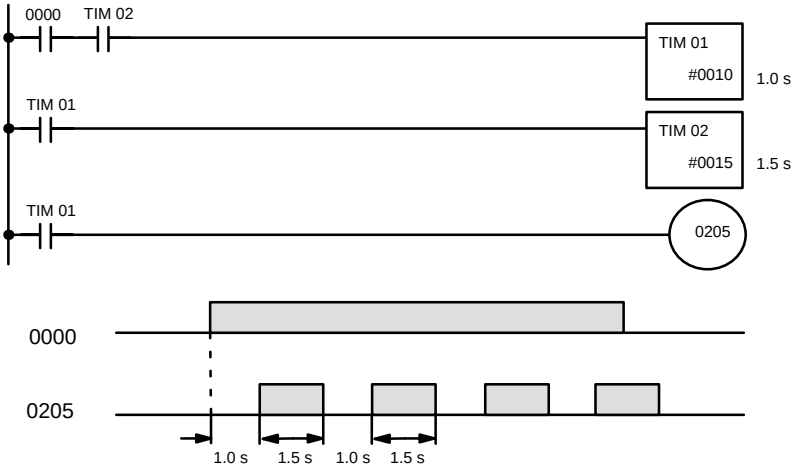
The length of time that a bit is kept ON or OFF can be controlled by combining TIM with OUT or OUT NOT. The following diagram demonstrates how this is possible. In this example, 0204 would remain ON for 1.5 seconds after 0000 goes ON regardless of the time 0000 stays ON. This is achieved by using 1000, activated by 0000, to turn ON 0204. When TIM 01 comes ON (i.e., when the SV of TIM 01 has expired), 0204 will be turned OFF through TIM 01 (i.e., TIM 01 will turn ON for an normally closed condition, creating an OFF execution condition for OUT 0204). TIM 01 will also turn OFF 1000 the next cycle, resetting the one-shot.



Address	Instruction	Operands
0000	LD	1000
0001	AND NOT	TIM 01
0002	OR	0000
0003	OUT	1000
0004	LD	1000
0005	TIM	01
		# 0015
0006	LD	1000
0007	AND NOT	TIM 01
0008	OUT	0204

Example 5: Flicker Bits

Bits can be programmed to turn ON and OFF at a regular interval 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.

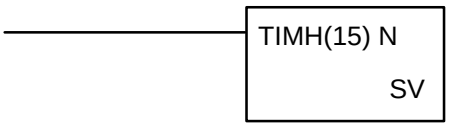


Address	Instruction	Operands
0000	LD	0000
0001	AND	TIM 02
0002	TIM	01
		# 0010
0003	LD	TIM 01
0004	TIM	02
		# 0015
0005	LD	TIM 01
0006	OUT	0205

An easier but more limited method of creating a flicker bit is to AND one of the SR area 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 SR area.

5-11-2 HIGH-SPEED TIMER – TIMH(15)

Ladder Symbol



Definer Values

N: TC number
(00 though 47)

Operand Data Areas

SV: Set value (word, BCD)
IR, HR, #

Limitations

SV may be between 00.02 and 99.99 seconds. (Actually settings of 00.00 and 00.01 are allowed, but 00.00 is meaningless and 00.01 is not reliable.) The decimal point of SV is not input.

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

A cycle time of greater than 10 ms will affect the accuracy of the timer.

Description

TIMH(15) operates the same as TIM except that TIMH measures in units of 0.01 second.

Refer to *5-11-1 TIMER – TIM* for operational details and examples. All aspects except for the above considerations 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, SR area clock pulse bits can be counted to produce timers using CNT. Refer to *5-11-4 COUNTER – CNT* for details.

Program execution will continue even if a non-BCD SV is used, but timing will not be accurate.

Flags

ER: SV is not in BCD.

5-11-3 Analog Timer Unit

The Analog Timer Unit uses two I/O words to provide four timers (T_0 to T_3). Each of the four timers may be set to a specific timer value (SV) within one of four ranges. The SV for each timer may be set using either a variable resistor on the Analog Timer Unit or from an external variable resistor.

Each timer is allocated five bits within the IR words allocated to the Analog Timer Units. The function of these is shown below. The words shown in the table are as seen from the CPU, i.e., the input word goes from the Analog Timer Unit to the CPU, the output word, from the CPU to the Analog Timer Unit. The CPU receives the Time Expired flag from the Unit and sends the Start control bit Pause control bit and Range bits to the Unit.

Bit	Input word	Output word
00	T_0 Time Expired flag	T_0 Start control bit
01	T_1 Time Expired flag	T_1 Start control bit
02	T_2 Time Expired flag	T_2 Start control bit
03	T_3 Time Expired flag	T_3 Start control bit
04	Cannot be used.	T_0 Pause control bit
05		T_1 Pause control bit
06		T_2 Pause control bit
07		T_3 Pause control bit
08		T_0 Range bits
09		
10		T_1 Range bits
11		
12		T_2 Range bits
13		
14		T_3 Range bits
15		

There is a SET indicator and a time expired indicator on the Analog Timer Unit for each timer. These indicators are lit when the corresponding timer's Start control bit or Time Expired flag is ON.

When the Start control bit is turned ON, the timer begins operation and the SET indicator is lit.

When the time set with the internal or external adjustment has expired, the corresponding Time Expired flag is set. The time up indicator also lights.

If the Pause control bit for a timer is turned ON from the PC, the timer will cease timing and the present value (PV) will be retained. Timing will resume when the Pause control bit is turned OFF. If the Start control bit is turned OFF before the set value (SV) of the timer has expired, the Time Expired flag will not be turned ON.

Timer ranges are set in the output words as shown in the following table.

Timer	Output word bit	0.1 to 1s	1 to 10s	10 to 60s	1 to 10m
T_0	08	OFF	ON	OFF	ON
	09	OFF	OFF	ON	ON
T_1	10	OFF	ON	OFF	ON
	11	OFF	OFF	ON	ON
T_2	12	OFF	ON	OFF	ON
	13	OFF	OFF	ON	ON
T_3	14	OFF	ON	OFF	ON
	15	OFF	ON	OFF	ON

Example

Setup

This example uses an Analog Timer Unit connected to a C28K CPU. Word allocations are shown in the following table.

Unit	Input word	Output word
CPU	00	01
Analog Timer Unit	02	03

All four time's are used. Times for two of them are adjusted on the variable resistors provided on the Analog Timer Unit. The other two times are adjusted using external resistors. These adjustments are made as follows. Refer to the *Analog Timer Unit Installation Guide* for hardware details.

Timer	SV	Range	Resistor adjustment
T_0	Approx. 0.6 s	0.1 to 1 s	6/10th turn clockwise
T_1	Approx. 3 s	1 to 10 s	3/10th turn clockwise
T_2	Approx. 2.6 s	10 to 60 s	2/10th turn clockwise
T_3	Approx. 8 min	1 to 10 min	8/10th turn clockwise

Programming

The following program sections are used to set up the required data and produce outputs from the four timers. The first section moves E400 into IR 06 to set the desired ranges (see table above). The second program section achieves the following operation.

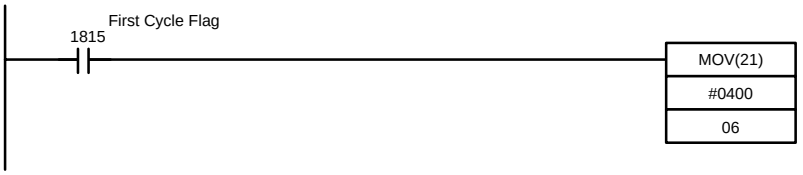
1, 2, 3... 1. IR 0500 is turned ON approximately 0.6 seconds after IR 0002 turns ON as the result of the action of T_0.

2. IR 0501 is turned ON approximately 3 seconds after IR 0003 turns ON as the result of the action of T_1.

3. IR 0502 is turned ON approximately 20 seconds after IR 0004 turns ON as the result of the action of T_2.

4. IR 0503 is turned ON approximately 8 minutes after IR 0004 turns ON as the result of the action of T_3.

5. T_2 and T_3 are made inoperative if IR 0015 is turned ON.

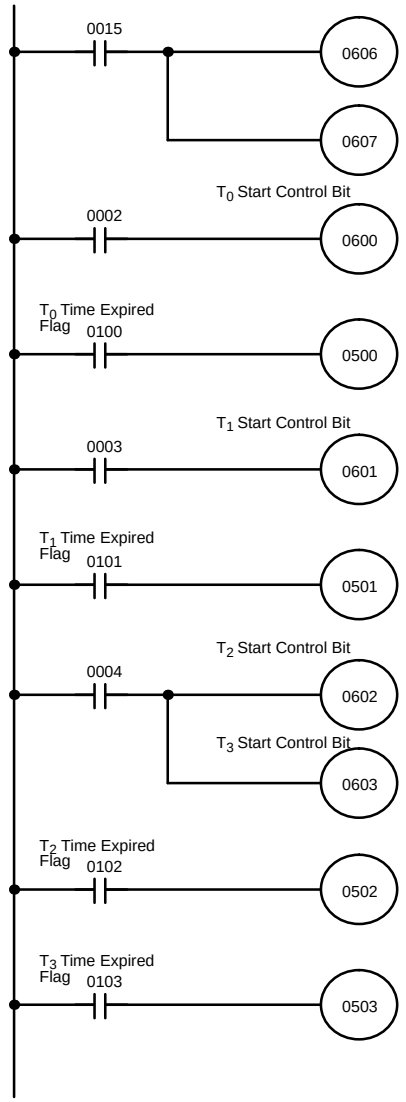


Address	Instruction	Operands
0000	LD	1815
0001	MOV(21)	
		# 0400
		06

Content of IR O6 after MOV(21)

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0

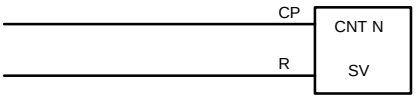
Range settings



Address	Instruction	Operands
0000	LD	0015
0001	OUT	0606
0002	OUT	0607
0003	LD	0002
0004	OUT	0600
0005	LD	0100
0006	OUT	0500
0007	LD	0003
0008	OUT	0601
0009	LD	0101
0010	OUT	0501
0011	LD	0004
0012	OUT	0602
0013	OUT	0603
0014	LD	0102
0015	OUT	0502
0016	LD	0103
0017	OUT	0503

5-11-4 COUNTER – CNT

Ladder Symbol



Definer Values

N: TC number
(00 through 47)

Operand Data Areas

SV: Set value (word, BCD)
IR, HR, #

Limitations

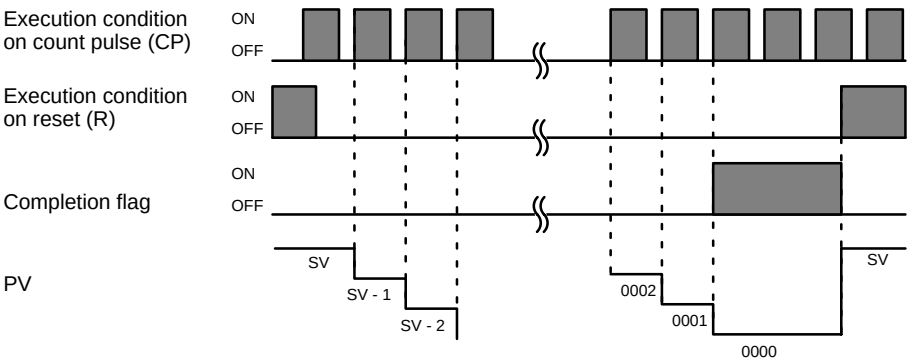
Each TC number can be used as the definer in only one timer or counter instruction.

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. 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 for power interruptions.

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



Precautions

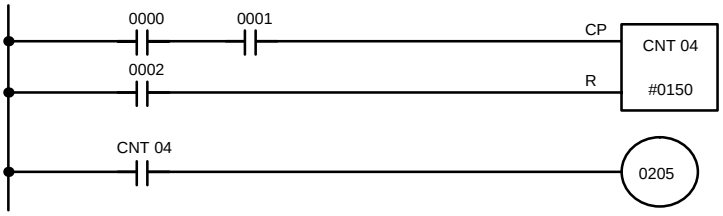
Program execution will continue even if a non-BCD SV is used, but the SV will not be correct.

Flags

ER: SV is not in BCD.

Example 1: Basic Application

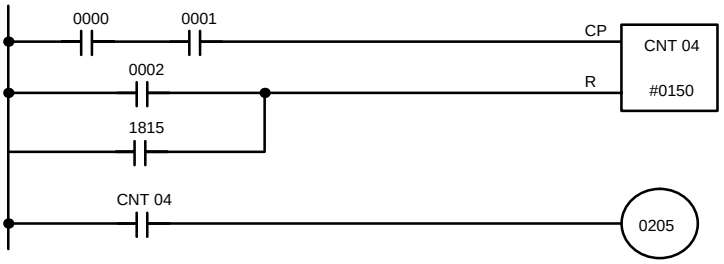
In the following example, the PV will be decremented whenever both 0000 and 0001 are ON provided that 0002 is OFF and either 0000 or 0001 was OFF the last time CNT 04 was executed. When 150 pulses have been counted down (i.e., when PV reaches zero), 0205 will be turned ON.



Address	Instruction	Operands
0000	LD	0000
0001	AND	0001
0002	LD	0002
0003	CNT	04
		# 0150
0004	LD	CNT 04
0005	OUT	0205

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.

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 Cycle flag in the SR area (1815) to reset CNT as shown below.



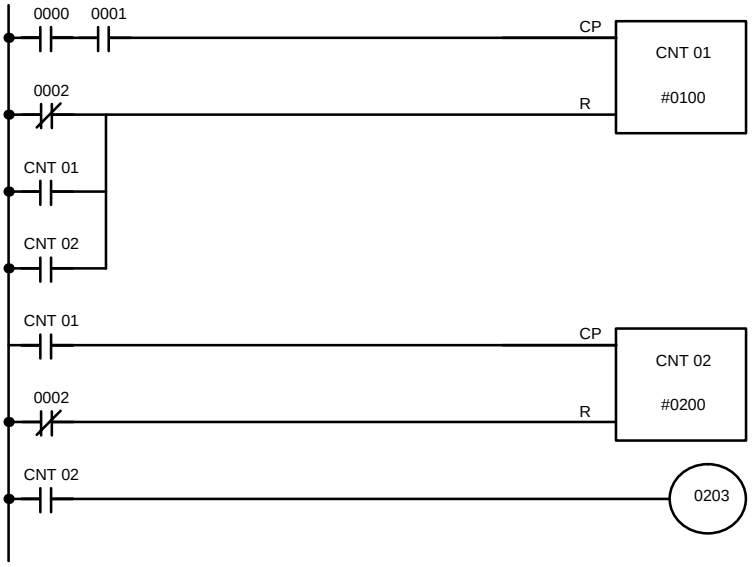
Address	Instruction	Operands
0000	LD	0000
0001	AND	0001
0002	LD	0002
0003	OR	1815
0004	CNT	04
		# 0150
0005	LD	CNT 04
0006	OUT	0205

Example 2: 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 reached zero from SV.

In the following example, 0000 is used to control when CNT 01 operates and 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 PV 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 0203 being turned ON.



Address	Instruction	Operands	
0000	LD		0000
0001	AND		0001
0002	LD NOT		0002
0003	OR	CNT	01
0004	OR	CNT	02
0005	CNT		01
		#	0100
0006	LD	CNT	01
0007	LD NOT		0002
0008	CNT		02
		#	0200
0009	LD	CNT	02
0010	OUT		0203

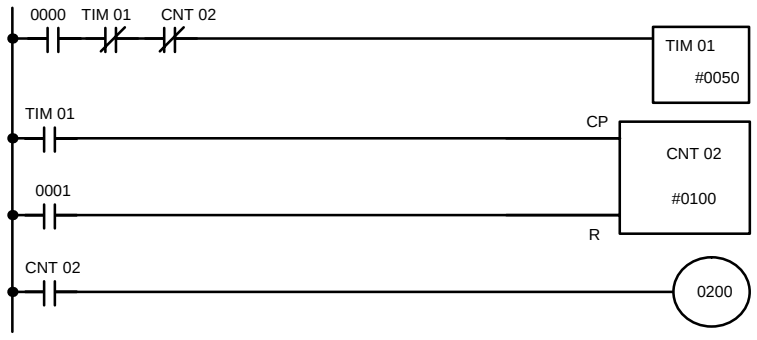
CNT can be used in sequence as many times as required to produce counters capable of counting down even higher values.

Example 3: Extended Timers

CNT can be used to create extended timers in two ways: by combining TIM with CNT and by counting SR area 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.

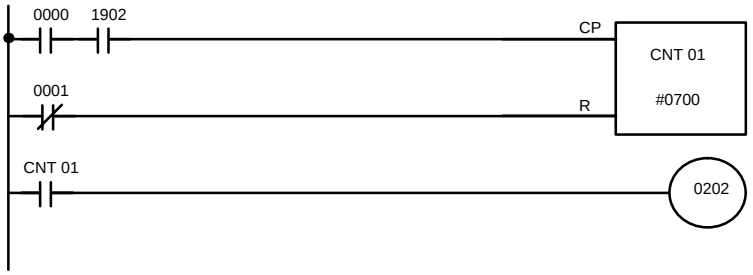
As the SV for TIM 01 is 5.0 seconds and the SV for CNT 02 is 100, the completion flag for CNT 02 turns ON when 5 seconds x 100 times, or 8 minutes and 20 seconds have expired. This would result in 0201 being turned ON.



Address	Instruction	Operands	
0000	LD		0000
0001	AND NOT	TIM	01
0002	AND NOT	CNT	02
0003	TIM		01
		#	0050
0004	LD	TIM	01
0005	LD		0001
0006	CNT		02
		#	0100
0007	LD	CNT	02
0008	OUT		0200

In the following example, CNT 01 counts the number of times the 1-second clock pulse bit (1902) goes from OFF to ON. Here again, 0000 is used to control when CNT is operating.

As the SV for CNT 01 is 700, the completion flag for CNT 02 turns ON when 1 second x 700 times, or 10 minutes and 40 seconds have expired. This would result in 0202 being turned ON.

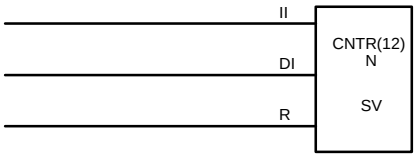


Address	Instruction	Operands
0000	LD	0000
0001	AND	1902
0002	LD NOT	0001
0003	CNT	01
		# 0700
0004	LD	CNT 01
0005	OUT	0202

⚠ Caution The shorter clock pulses may not produce accurate timers because their short ON times may not be read accurately for longer cycle times. In particular the 0.02-second and 0.1-second clock pulses should not be used to create timers with CNT.

5-11-5 REVERSIBLE COUNTER – CNTR(12)

Ladder Symbol



Definer Values

N: TC number
(00 through 47)

Operand Data Areas

SV: Set value (word, BCD)
IR, HR, #

Limitations

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

Description

The CNTR(12) is a reversible, up-down circular counter, i.e., it is used to count between zero and SV according to changes in two execution conditions, those in the increment input (II) and those in the decrement input (DI).

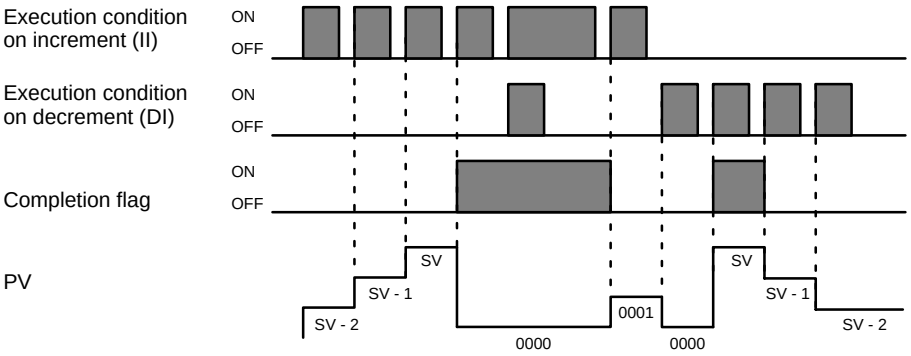
The present value (PV) will be incremented by one whenever CNTR(12) is executed with an ON execution condition for II and the execution condition was OFF for II for the last execution. The present value (PV) will be decremented by one whenever CNTR(12) is executed with an ON execution condition for DI and the execution condition was OFF for DI for the last execution. If OFF to ON changes have occurred in both II and DI since the last execution, the PV will not be changed.

If the execution conditions have not changed or has changed from ON to OFF for both II and DI, the PV of CNT will not be changed.

When decremented from 0000, the present value is set to SV and the completion flag is turned ON until the PV is decremented again. When incremented past the SV, the PV is set to 0000 and the completion flag is turned ON until the PV is incremented again.

CNTR(12) is reset with a reset input, R. When R goes from OFF to ON, the PV is reset to zero. The PV will not be incremented or decremented while R is ON. Counting will begin again when R goes OFF. The PV for CNTR(12) will not be reset in interlocked program sections or for power interruptions.

Changes in II and DI execution conditions, the completion flag, and the PV are illustrated below starting from part way through CNTR(12) operation (i.e., when reset, counting begins from zero). PV line height is meant to indicate changes in the PV only.



Precautions

Program execution will continue even if a non-BCD SV is used, but the SV will not be correct.

Flags

ER: SV is not in BCD.

5-11-6 HIGH-SPEED DRUM COUNTER – HDM(61)

Ladder Symbol

Definer Values

N: TC number
Must be 47

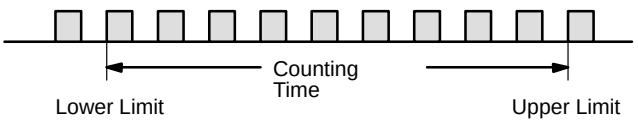
Operand Data Areas

R: Result word
IR, HR, DM

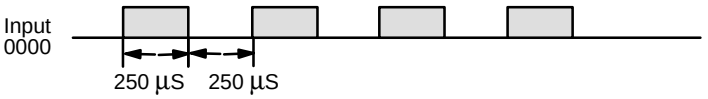
Limitations

If any of the lower limits for the DM ranges are set to “0000,” the corresponding output bits are turned ON when the high-speed counter is reset.

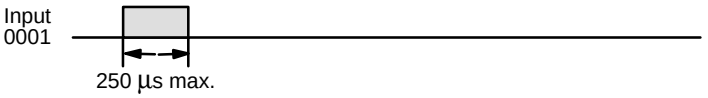
If the time it takes to count through some range is less than the cycle time of the CPU, the high-speed counter may count past between cycles and thus the output bit for this range may not be turned ON.



The count signal must be at least 250 μs (2 kHz) wide and have a duty factor of 1:1, as shown below.



In the hard reset mode, the reset signal must have an ON time of at least 250 μs.



Description

General

The high-speed counter counts the signals input from an external device connected to input 0000 and, when the high-speed counter instruction is executed, compares the current value with a set of ranges which have been preset in DM words 32 through 63. If the current value is within any of the preset ranges, the corresponding bit of a specified result word is turned ON. The bit in the result word remains ON until the current value is no longer within the specified range.

An internal buffer is incremented whenever bit 0000 goes from OFF to ON. When the high-speed counter instruction is executed, the value in the counter buffer is transferred to counter 47 which serves as the count value storage area.

When using the high-speed counter, the following bits are reserved and cannot be used for any other purpose:

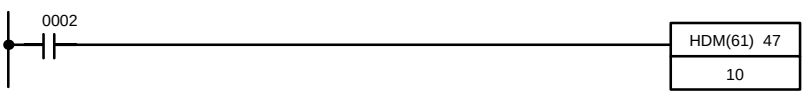
- Input 0000 (count input)
- Input 0001 (hard reset)
- SR bit 1807 (soft reset)
- TC 47 (present count value)
- DM 32 to 63 (upper and lower limits)

Note If a power failure occurs, the count value of the high speed counter immediately before the power failure is retained.

The high-speed counter is programmed differently depending on how it is to be reset. Two resetting modes are possible: hard-reset and soft-reset. The hard reset is made effective or ineffective with the DIP switch in the CPU.

Hard Reset

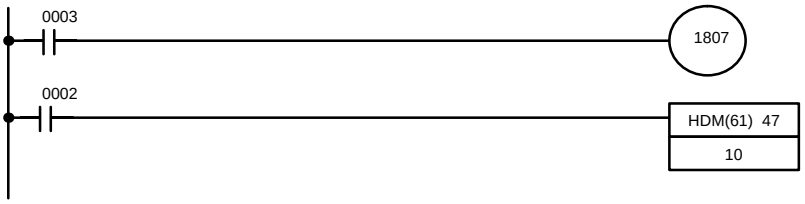
To use the hard reset, turn pins 7 and 8 ON. In this mode, input 0001 is the reset input. When it is turned ON, the present value in the high-speed counter buffer is reset to “0000.” When the reset is ON, the count signal from input 0000 is not accepted. When programmed with the hard reset, the high-speed counter would appear as below.



Address	Instruction	Operands
0000	LD	0002
0001	HDM(61)	47
		10

Soft Reset

SR bit 1807 is the soft reset. When it is turned ON, the present value in the high-speed counter buffer is reset to “0000.” As for the hard reset, when the soft reset is ON, the count signal from input 0000 is not accepted. When programmed with the soft reset, the high-speed counter would appear as below. Note that when the soft reset is used, the timing at which the counter buffer is reset may be delayed due to the cycle time of the CPU.



Address	Instruction	Operands
00000	LD	0003
00001	OUT	1807
00002	LD	0002
00003	HDM(61)	47
		10

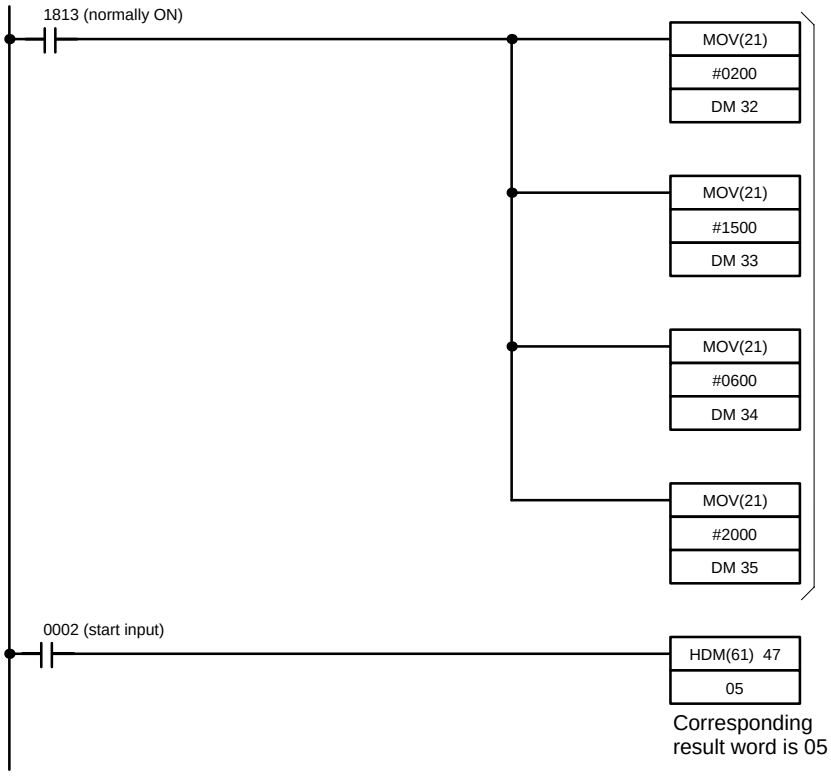
If required, both the hard reset and the soft reset can be used together.

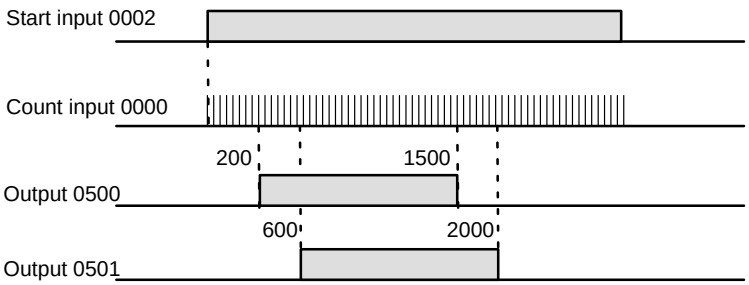
Upper and Lower Limit Setting

The following table shows the upper and lower limits that need to be set in DM 32 through DM 63. In this table, “S” denotes the present value of counter 47 and R is the results word.

Lower limit	Upper limit	Present value of the counter	Bit of R that turns ON
DM 32	DM 33	Value of DM 32 ≤ S ≤ value of DM 33	00
DM 34	DM 35	Value of DM 34 ≤ S ≤ value of DM 35	01
DM 36	DM 37	Value of DM 36 ≤ S ≤ value of DM 37	02
DM 38	DM 39	Value of DM 38 ≤ S ≤ value of DM 39	03
DM 40	DM 41	Value of DM 40 ≤ S ≤ value of DM 41	04
DM 42	DM 43	Value of DM 42 ≤ S ≤ value of DM 43	05
DM 44	DM 45	Value of DM 44 ≤ S ≤ value of DM 45	06
DM 46	DM 47	Value of DM 46 ≤ S ≤ value of DM 47	07
DM 48	DM 49	Value of DM 48 ≤ S ≤ value of DM 49	08
DM 50	DM 51	Value of DM 50 ≤ S ≤ value of DM 51	09
DM 52	DM 53	Value of DM 52 ≤ S ≤ value of DM 53	10
DM 54	DM 55	Value of DM 54 ≤ S ≤ value of DM 55	11
DM 56	DM 57	Value of DM 56 ≤ S ≤ value of DM 57	12
DM 58	DM 59	Value of DM 58 ≤ S ≤ value of DM 59	13
DM 60	DM 61	Value of DM 60 ≤ S ≤ value of DM 61	14
DM 62	DM 63	Value of DM 62 ≤ S ≤ value of DM 63	15

The values must be four-digit BCD in the range 0000 to 9999. Note that failure to enter BCD values will not activate the ERR flag. Always set a lower limit which is less than the corresponding upper limit. MOV is useful in setting limits. The following ladder diagram shows the use of MOV for setting limits and the associated timing diagram shows the state of the relevant bits of the result word (IR 05) as the counter is incremented.





Address	Instruction	Operands
0000	LD	1813
0001	MOV(21)	
		# 0200
		DM 32
0002	MOV(21)	
		# 1500
		DM 33
0003	MOV(21)	
		# 0600
		DM 34
0004	MOV(21)	
		# 2000
		DM 35
0005	LD	0002
0006	HDM(61)	47
		05

Response Speed

The maximum response speed of the high-speed counter hardware is 2 kHz. Note however that the start signal, reset signal (in the case of soft reset), and corresponding outputs are all processed by software. Because of this, response may be delayed by the cycle time.

Precautions

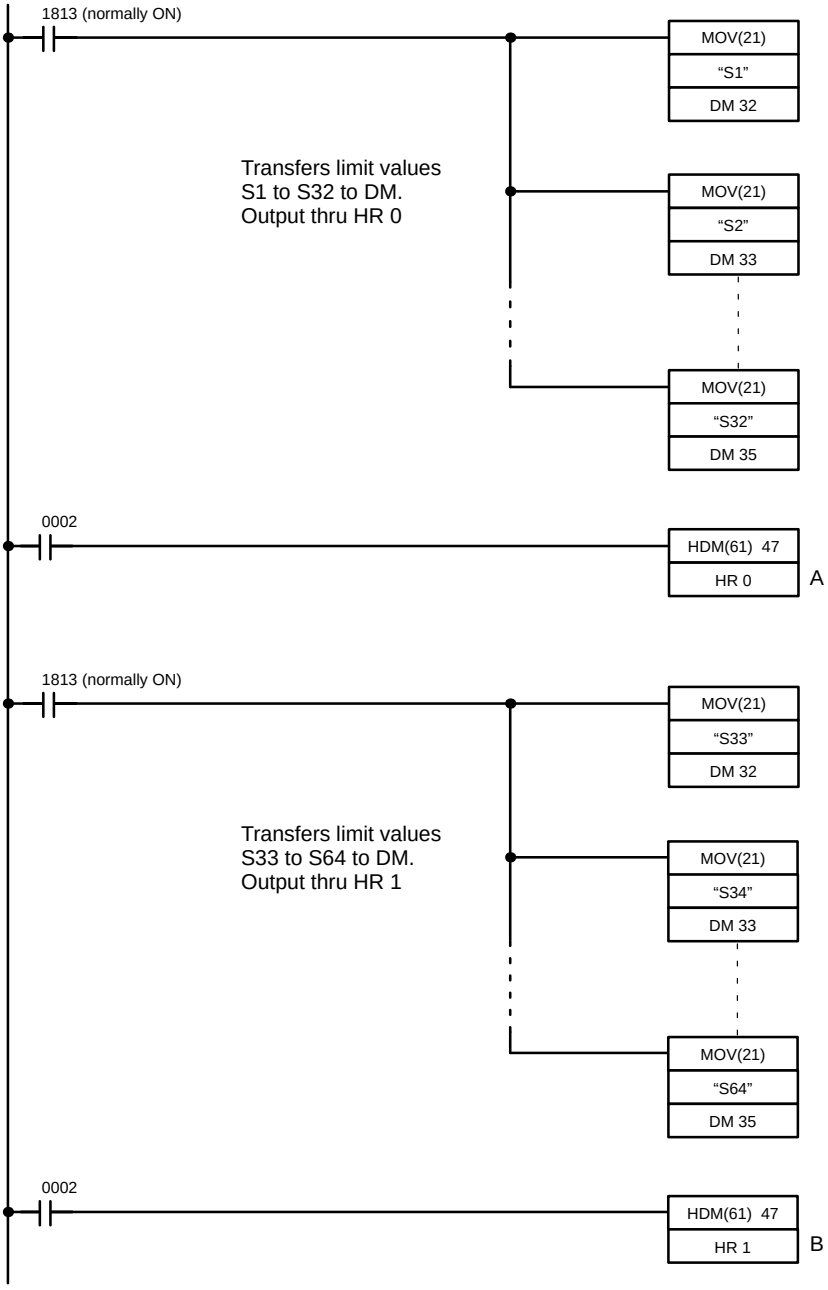
When programming the high-speed counter with the GPC, “00” is displayed on each of the three lines below the instruction code (HDM(61)). Do not alter the second and third lines; if they are not “00,” an error occurs when an attempt is made to transfer the program from the GPC to the PC.

Do not program the high-speed counter between JMP and JME. The high-speed counter can be programmed between IL and ILC. However, the hard reset signal remains active, causing the corresponding output(s) to turn ON or OFF, even when the IL condition is OFF.

Examples

Extending the Counter

The high-speed counter normally provides 16 output bits. If more than 16 are required, the high-speed counter may be programmed more than once. In the following program example, the high-speed counter is used twice to provide 32 output bits.



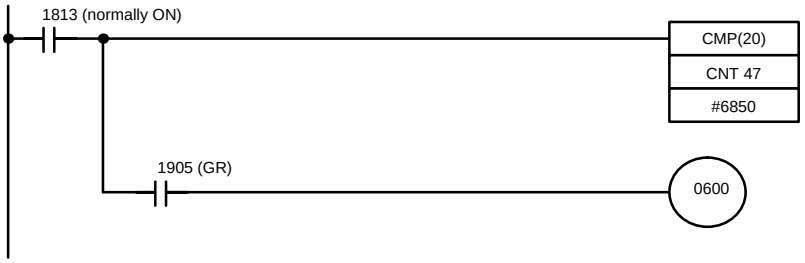
Address	Instruction	Operands
0000	LD	1813
0001	MOV(21)	
		"S1"
		DM 32
0002	MOV(21)	
		"S2"
		DM 33
0003	MOV(21)	
		"S32"
		DM 35
0004	LD	0002
0005	HDM(61)	47
		HR 0
0006	LD	1813
0007	MOV(21)	
		"S33"
		DM 32
0008	MOV(21)	
		"S34"
		DM 33
0009	MOV(21)	
		"S64"
		DM 35
0010	LD	0002
0011	HDM(61)	47
		HR 1

In this program, each bit in the specified words, HR 0 and HR 1 are turned ON under the following conditions (where S is the present count value of the high-speed counter stored as the data of CNT 47):

Where $S1 \leq S \leq S2$, HR 000 is ON.
Where $S3 \leq S \leq S4$, HR 001 is ON.
Where $S31 \leq S \leq S32$, HR 015 is ON.
Where $S33 \leq S \leq S34$, HR 100 is ON.
Where $S63 \leq S \leq S64$, HR 115 is ON.

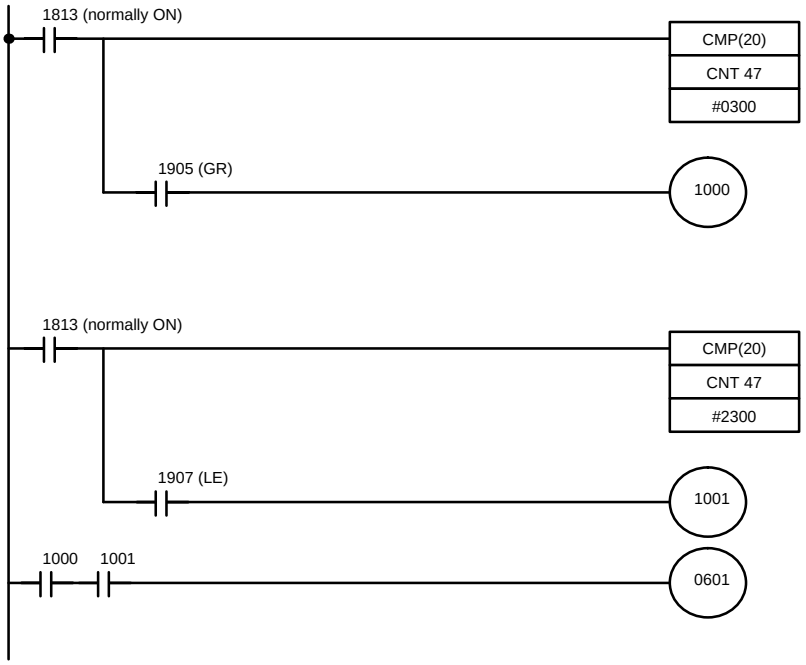
Note that in the program just mentioned, the present value in the counter buffer is transferred to counter number 47 at points A and B. In this case, if S31 (=1,000) < S < S32 (=2,000) and S33 (=2,000) < S < S34 (=3,000), and if the present count value of the first high-speed counter (at point A) is 1,999 and that of the second counter (at point B) is 2,003, HR 015 and HR100 may be simultaneously turned ON. If it is necessary to avoid this, set the values of S32 and S33 so that there is a value difference equivalent to the time lag from points A to B. For example, set the value of S32 to 2,000 and that of S33 to 2,010.

More than 16 output bits may be obtained using CMP.



Address	Instruction	Operands	
0000	LD		1813
0001	CMP(20)		
		CNT	47
		#	6850
0002	AND		1905
0003	OUT		0600

In the above program, output 0600 is turned ON when the following condition is satisfied, where S is the present count value of the high-speed counter: 6,850 < S ≤ 9,999.



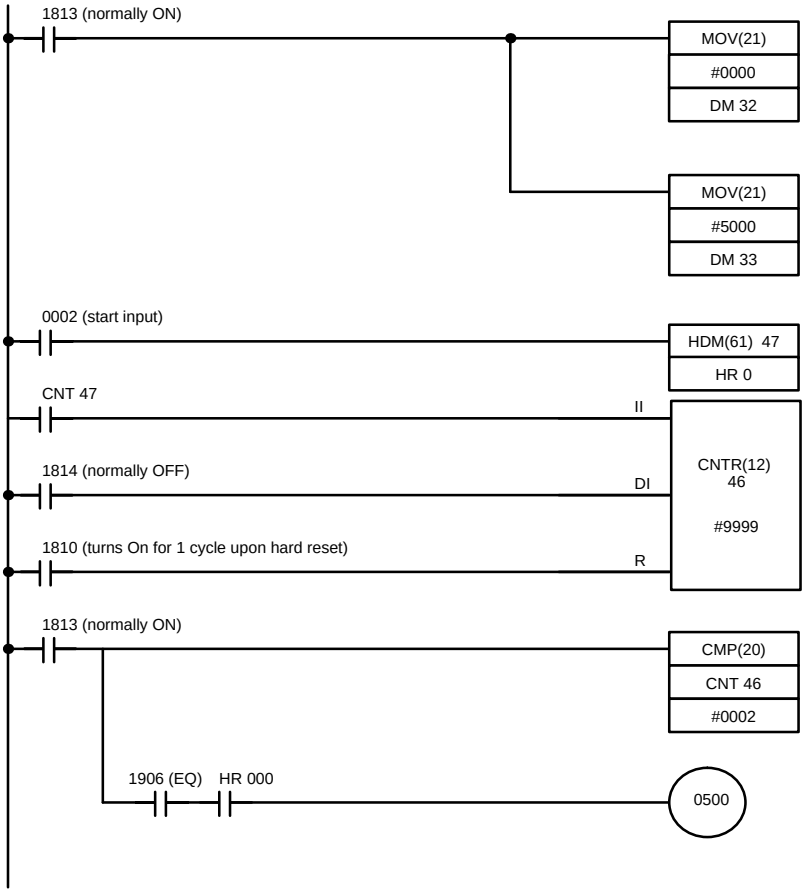
Address	Instruction	Operands	
0000	LD		1813
0001	CMP(20)		
		CNT	47
		#	0300
0002	AND		1905
0003	OUT		1000
0004	LD		1813
0005	CMP(20)		
		CNT	47
		#	2300
0006	AND		1907
0007	OUT		1001
0008	LD		1000
0009	AND		1001
0010	OUT		0601

In the above program, output 0601 is turned ON when the following condition is satisfied, where S is the present count value of the high-speed counter: 300 < S < 2,300.

Cascade Connection (Counting Past 9,999)

The number of digits of the upper and lower limits of the high-speed counter can be increased from four to eight by using the high-speed counter together with CNTR and CMP.

The high-speed counter is a ring counter and thus when its present count value is incremented from 9999 to 0000, the completion flag of CNT 47 is turned ON for one cycle. By using this flag as an input to the UP input of the reversible counter (i.e., cascade connection) you can increase the number of digits to eight. Although an ordinary counter can be cascade-connected to the high-speed counter, programming is easier with CNTR since an ordinary counter is decrementing.



Address	Instruction	Operands	
0000	LD		1813
0001	MOV(21)		
		#	0000
		DM	32
0002	MOV(21)		
		#	5000
		DM	33
0003	LD		0002
0004	HDM(61)		47
		HR	0
0005	LD	CNT	47
0006	LD		1814
0007	LD		1810
0008	CNTR(12)		
			46
		#	9999
0009	LD		1813
0010	CMP(20)		
		CNT	46
		#	0002
0011	AND		1906
0012	AND	HR	000
0013	OUT		0500

In the above program example, output 0500 is turned ON when the following condition is satisfied (where S is the present count value of the high-speed counter):
$20{,}000 \leq S \leq 25{,}000$.

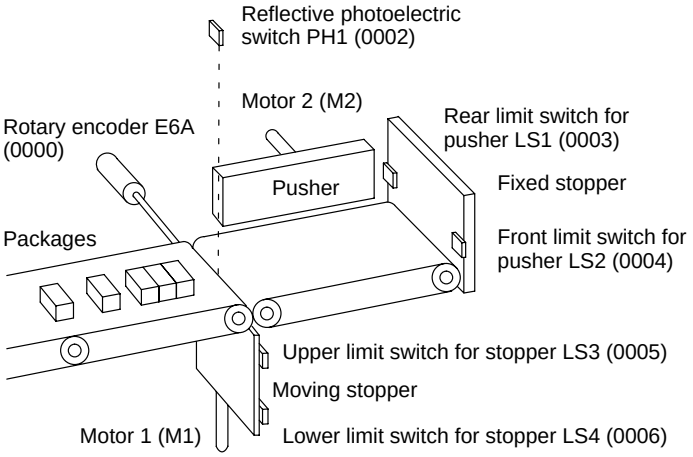
Note In hard reset mode, program SR 1810, which turns ON for one cycle time upon input of the hard reset signal, to CNTR as the reset input. Unless CNTR and CMP are programmed immediately after the high-speed counter, the correct corresponding outputs may not be produced.

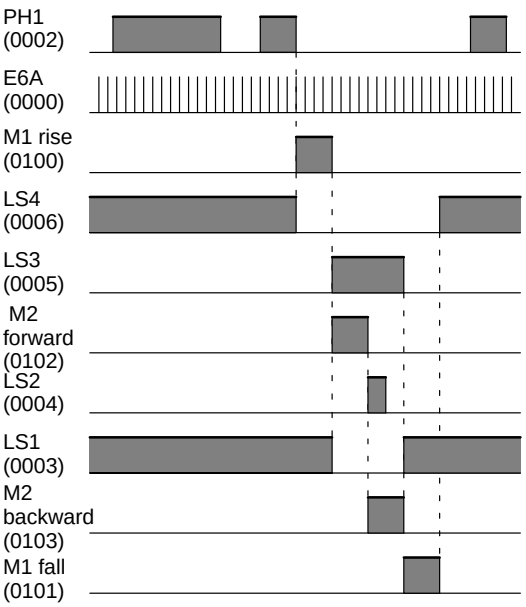
Packaging Machine

The high-speed counter is very useful in the following application. Here, packages are being carried on a conveyor belt at random intervals. Some of them are spaced far apart and others are clustered together, making it impossible to accurately detect and count them with photoelectric switches alone.

By presetting the number of pulses generated when a single package is detected and by counting those pulses, the number of packages can be accurately counted, regardless of whether the packages are spaced or clustered.

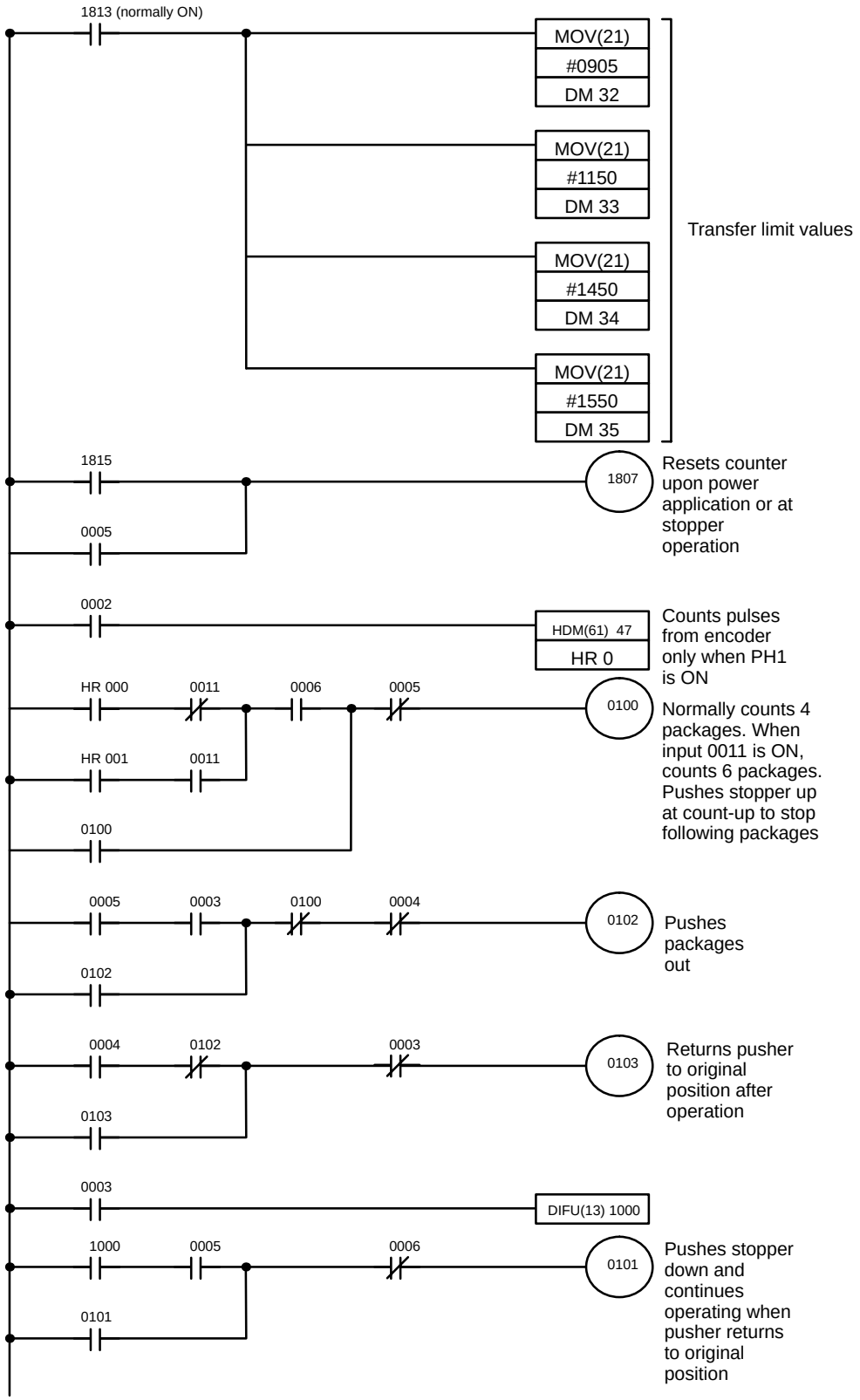
The following diagram shows the packaging system and the corresponding timing chart.





In this example, “x” is the number of pulses per package. To detect four packages therefore, 4x must be set as the preset value of the high-speed counter.

Here is the program example for the application.

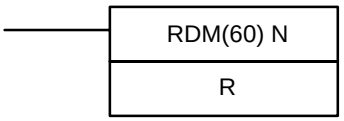


Address	Instruction	Operands	
0000	LD		1813
0001	MOV(21)		
		#	0905
		DM	32
0002	MOV(21)		
		#	1150
		DM	33
0003	MOV(21)		
		#	1450
		DM	34
0004	MOV(21)		
		#	1550
		DM	35
0005	LD		1815
0006	OR		0005
0007	OUT		1807
0008	LD		0002
0009	HDM(61)		47
		HR	0
0010	LD	HR	000
0011	AND NOT		0011
0012	LD	HR	001
0013	AND		0011

Address	Instruction	Operands
0014	OR LD	
0015	AND	0006
0016	OR	0100
0017	AND NOT	0005
0018	OUT	0100
0019	LD	0005
0020	AND	0003
0021	OR	0102
0022	AND NOT	0100
0023	AND NOT	0004
0024	OUT	0102
0025	LD	0004
0026	AND NOT	0102
0027	OR	0103
0028	AND NOT	0003
0029	OUT	0103
0030	LD	0003
0031	DIFU(13)	1000
0032	LD	1000
0033	AND	0005
0034	OR	0101
0035	AND NOT	0006
0036	OUT	0101

5-11-7 REVERSIBLE DRUM COUNTER – RDM(60)

Ladder Symbol



Definer Values

N: TC number
Must be 46

Operand Data Areas

R: Result word
IR, HR, DM

Limitations

If any of the lower limits for the DM ranges are set to “0000,” the corresponding output bits are turned ON when the counter is reset.

Description

The reversible drum counter is a ring counter with a counting range of 0000 to 9999. It requires three input signals to operate: a count input, reset input, and UP/DOWN selection input. For these inputs, SR bits 1804 to 1806 are reserved and cannot be used for any other purpose while the RDM(60) is being used.

Operation is enabled when RDM(60) is executed with an ON execution condition. RDM(60) increments when the UP/DOWN selection input (1806) is OFF. When this input is ON, RDM(60) decrements. Incrementing or decrementing occurs on the rising edge of the count input signal. When RDM(60) is executed by the CPU, the value in the counter buffer is transferred to CNT 46.

The transferred count value is then compared with the upper and lower limits of a set of ranges which have been preset in DM 00 through DM 31. If the current value is within any of the preset ranges, the corresponding bit of the results word, R, is turned ON. The bit in the result word will remain ON until the current value is no longer within the specified range.

When the reset input (1804) goes ON, the present value is cleared to 0000, even if the start input is OFF, and all bits in R are set.

Note RDM(60) cannot be used to create a high-speed counter. If you require a high-speed counter, use HDM(61).

Dedicated Bits

When using the counter, the following bits are reserved and cannot be used for any other purpose:

- SR bit 1804 (reset input)
- SR bit 1805 (count input)
- SR bit 1806 (UP/DOWN selection input)
- TC 46 (present count value)
- DM 00 to 31 (upper and lower limits)

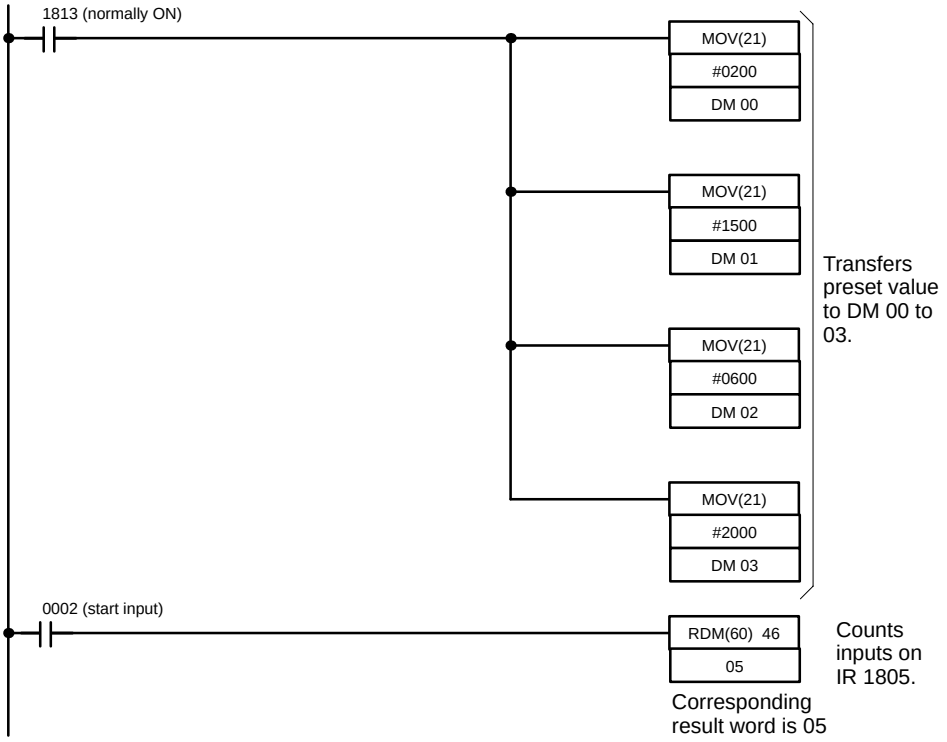
Note If a power failure occurs, the count value of the counter immediately before the power failure is retained.

Upper and Lower Limit Setting

The following table shows the upper and lower limits that need to be set in DM 00 through DM 31. In this table, S is the present value of counter 46 and R is the result word.

Lower limit	Upper limit	Present value of the counter	Bit of R that turns ON
DM 00	DM 01	Value of DM 00 ≤ S ≤ value of DM 01	00
DM 02	DM 03	Value of DM 02 ≤ S ≤ value of DM 03	01
DM 04	DM 05	Value of DM 04 ≤ S ≤ value of DM 05	02
DM 06	DM 07	Value of DM 06 ≤ S ≤ value of DM 07	03
DM 08	DM 09	Value of DM 08 ≤ S ≤ value of DM 09	04
DM 10	DM 11	Value of DM 10 ≤ S ≤ value of DM 11	05
DM 12	DM 13	Value of DM 12 ≤ S ≤ value of DM 13	06
DM 14	DM 15	Value of DM 14 ≤ S ≤ value of DM 15	07
DM 16	DM 17	Value of DM 16 ≤ S ≤ value of DM 17	08
DM 18	DM 19	Value of DM 18 ≤ S ≤ value of DM 19	09
DM 20	DM 21	Value of DM 20 ≤ S ≤ value of DM 21	10
DM 22	DM 23	Value of DM 22 ≤ S ≤ value of DM 23	11
DM 24	DM 25	Value of DM 24 ≤ S ≤ value of DM 25	12
DM 26	DM 27	Value of DM 26 ≤ S ≤ value of DM 27	13
DM 28	DM 29	Value of DM 28 ≤ S ≤ value of DM 29	14
DM 30	DM 31	Value of DM 30 ≤ S ≤ value of DM 31	15

The values must be four-digit BCD in the range 0000 through 9999. Failure to enter BCD values will not activate the ERR flag. Always set a lower limit which is less than the corresponding upper limit. MOV(21) is useful in setting limits. The following ladder diagram shows the use of MOV(21) for setting limits.

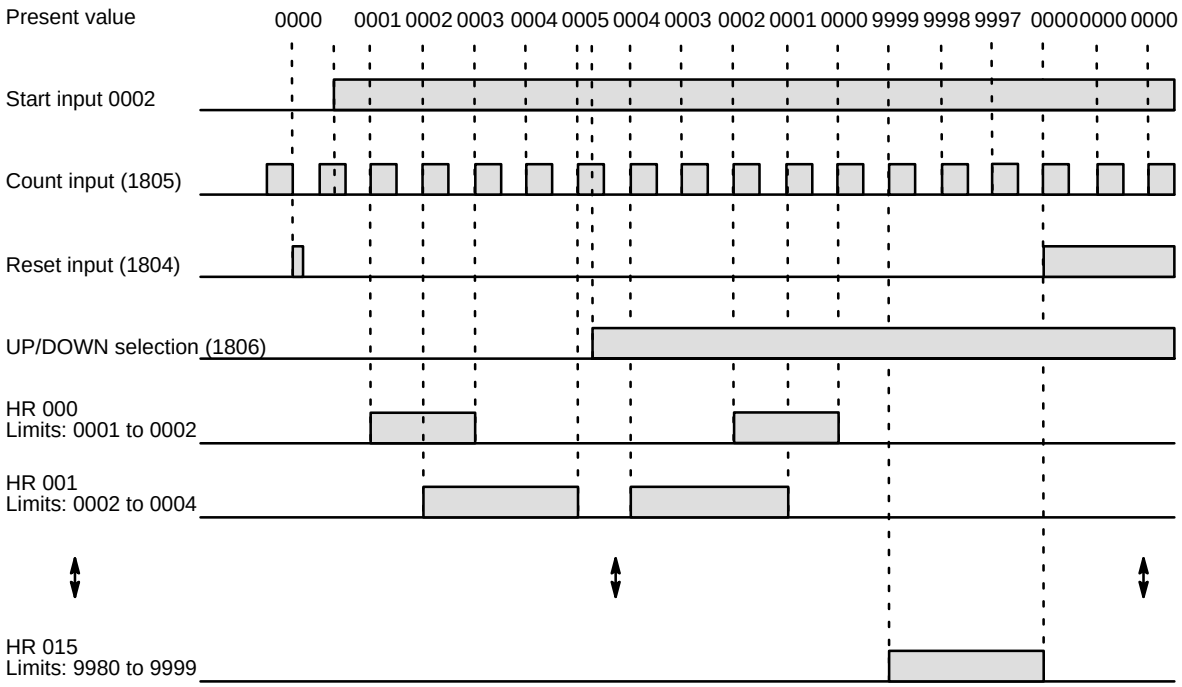


Address	Instruction	Operands
0000	LD	1813
0001	MOV(21)	
		# 0200
		DM 00
0002	MOV(21)	
		# 1500
		DM 01
0003	MOV(21)	
		# 0600
		DM 02

Address	Instruction	Operands
0004	MOV(21)	
		# 2000
		DM 03
0005	LD	0002
0006	RDM(60)	46
		05

Timing Example

The following timing example uses HR 0 as the results word.

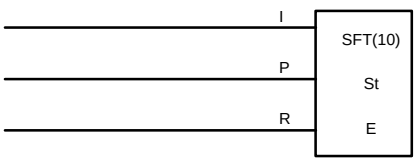


5-12 Data Shifting

This section describes the instructions that are used to create and manipulate shift registers. SFT(10) creates a single- or multiple-word register that shift in a second execution condition when executed with an ON execution condition. SFTR(84) creates a reversible shift register that is controlled through the bits in a control word. WSFT(16) creates a multiple-word register that shifts by word.

5-12-1 SHIFT REGISTER – SFT(10)

Ladder Symbol



Operand Data Areas

St : Starting word
IR, HR

E : End word
IR, HR

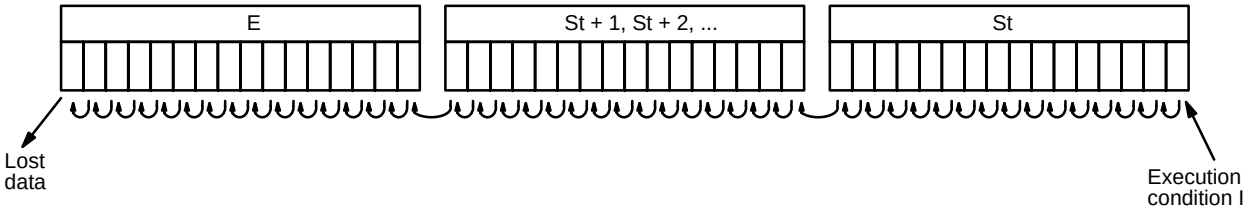
Limitations

E must be less than or equal to St, and St and E must be in the same data area.

If a bit address in one of the words used in a shift register is also used in an instruction that controls individual bit status (e.g., OUT, KEEP(11)), an error ("COIL DUPL") will be generated when program syntax is checked on the Programming Console or another Programming Device. The program, however, will be executed as written. See *Example 2: Controlling Bits in Shift Registers* for a programming example that does this.

Description

SFT(10) shifts an execution condition into a shift register. SFT(10) is controlled by three execution conditions, I, P, and R. If SFT(10) 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(10) was executed. If execution condition P has not changed or has gone from ON to OFF, the shift register will remain unaffected.

St designates the rightmost word of the shift register; E designates the leftmost. The shift register includes both of these words and all words between them. The same word may be designated for St and E to create a 16-bit (i.e., 1-word) shift register.

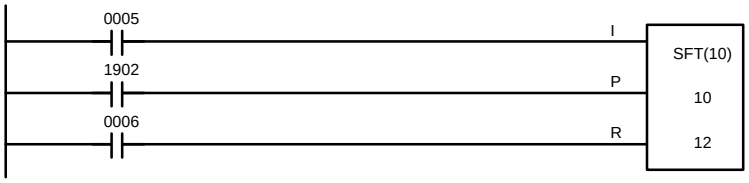
When execution condition R goes ON, all bits in the shift register will be turned OFF (i.e., set to 0) and the shift register will not operate until R goes OFF again.

Flags

There are no flags affected by SFT(10).

Example 1: Basic Application

The following example uses the 1-second clock pulse bit (1902) to so that the execution condition produced by 0005 is shifted into a 3-word register between 10 and 12 every second.

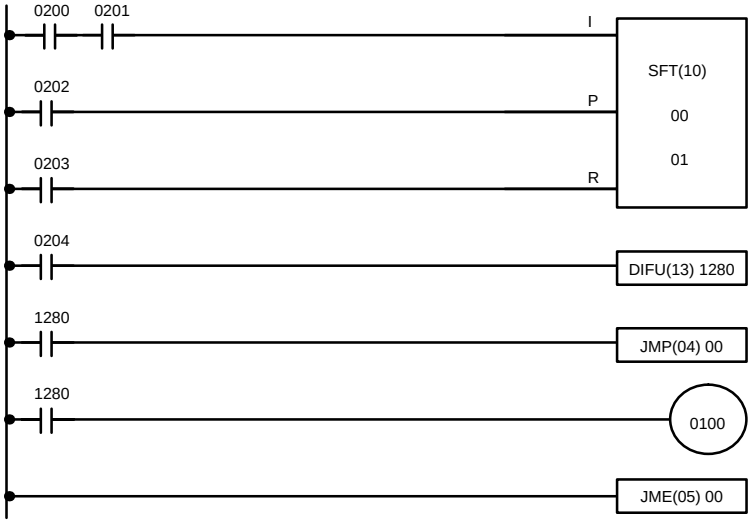


Address	Instruction	Operands
0000	LD	0005
0001	LD	1902
0002	LD	0006
0003	SFT(10)	
		10
		12

Example 2: Controlling Bits in Shift Registers

The following program is used to control the status of the 17th bit of a shift register running from IR 00 through IR 01 (i.e. bit 00 of IR 01). When the 17th bit is to be set, 0204 is turned ON. This causes the jump for JMP(04) 00 not to be made for that one cycle and IR 0100 (the 17th bit) will be turned ON.

When 1280 is OFF (all times but the first cycle after 0204 has changed from OFF to ON), the jump is taken and the status of 0100 will not be changed.



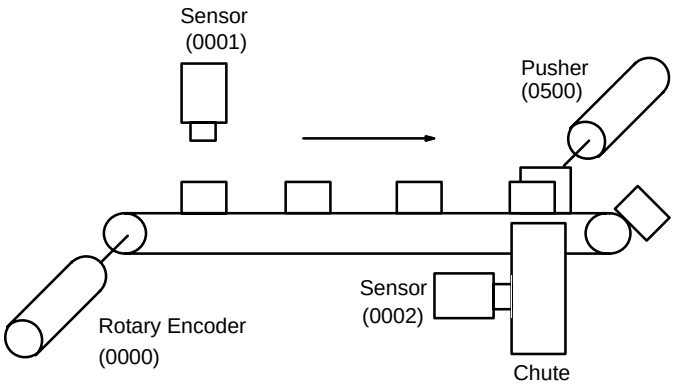
Address	Instruction	Operands
0000	LD	0200
0001	AND	0201
0002	LD	0202
0003	LD	0203
0004	SFT(10)	
		00
		01
0005	LD	0204
0006	DIFU(13)	1280
0007	LD	1280
0008	JMP(04)	00
0009	LD	1280
0010	OUT	0100
0011	JME(05)	00

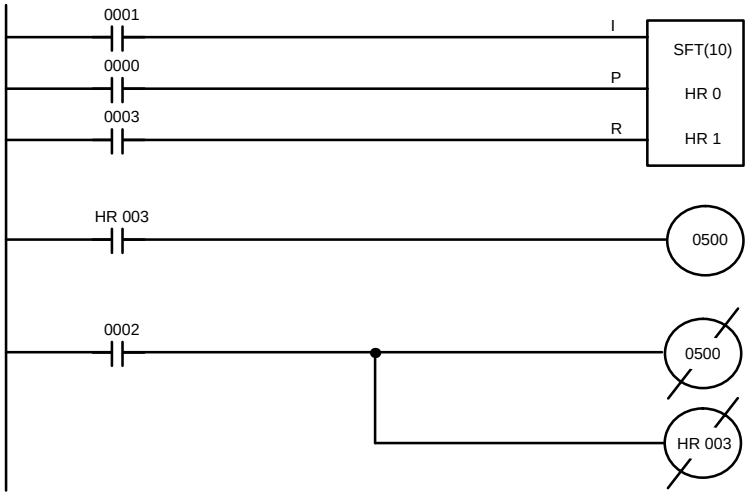
When a bit that is part of a shift register is used in OUT (or any other instruction that controls bit status), a syntax error will be generated during the program check, but the program will execute properly (i.e., as written).

Example 3: 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 HR 003 of the shift register can be used to activate a pusher (0500) when a faulty product reaches it, i.e., when HR 003 turns ON, 0500 is turned ON to activate the pusher.

The program is set up so that a rotary encoder (0000) controls execution of SFT(10) through a DIFU(13), 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 HR 003 of the shift register can be reset as required.

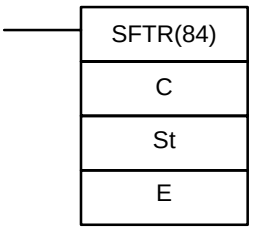




Address	Instruction	Operands	
0000	LD		0001
0001	LD		0000
0002	LD		0003
0003	SFT(10)		
		HR	0
		HR	1
0004	LD	HR	003
0005	OUT		0500
0006	LD		0002
0007	OUT NOT		0500
0008	OUT NOT	HR	003

5-12-2 REVERSIBLE SHIFT REGISTER – SFTR(84)

Ladder Symbols



Operand Data Areas

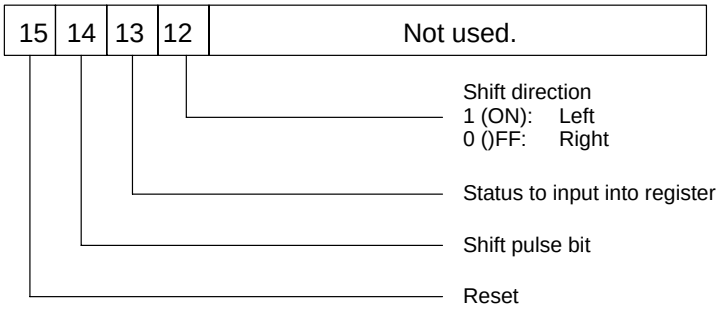
C : Control word
IR, DM, HR
St : Starting word
IR, DM, HR
E : End word
IR, DM, HR

Limitations

St and E must be in the same data area and St must be less than or equal to E.

Description

SFTR(84) is used to create a single- or multiple-word shift register that can be shifted to either the right or the left. To create a single-word shift register, designate the same word for St and E. The control word provides the shift direction, the status to be input into the register, the shift pulse, and the reset input. The control word is allocated as follows:



The data in the shift register will be shifted one bit in the direction indicated by bit 12, shifting one bit out to CY and the status of bit 13 into the other end whenever SFTR(84) is executed with an ON execution condition as long as the reset bit is OFF and as long as bit 14 is ON. If SFTR(84) is executed with an OFF execution condition or if SFTR(84) is executed with bit 14 OFF, the shift register will remain unchanged. If SFTR(84) is executed with an ON execution condition and the reset bit (bit 15) is OFF, the entire shift register and CY will be set to zero.

Flags

ER: St and E are not in the same data area or St is greater than E.

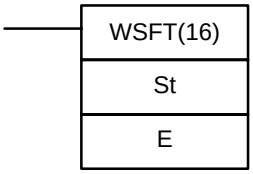
Indirectly addressed DM word is non-existent. (Content of ∗DM word is not BCD, or the DM area boundary has been exceeded.)

CY Receives the status of bit 00 of St or bit 15 of E depending on the shift direction.

⚠ Caution If the execution condition for SFTR(84) is ON and bit 14 is ON, the status of bit 13 will be shifted into the register every cycle. Use DIFU(13) or DIFD(14) when it is necessary to ensure that the shift is made only once each time the execution condition comes ON.

5-12-3 WORD SHIFT – WSFT(16)

Ladder Symbols



Operand Data Areas

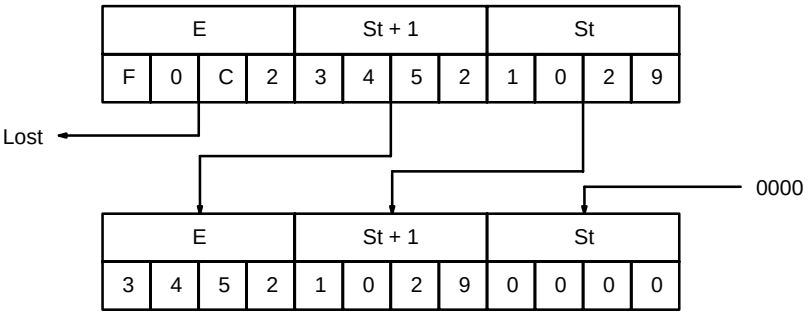
St : Start word	
IR, DM, HR	
E : End word	
IR, DM, HR	

Limitations

St and E must be in the same data area and St must be less than E.

Description

When the execution condition is OFF, WSFT(16) is not executed and the next instruction is moved to. When the execution condition is ON, 0000 is moved into St, the content of St is moved to St + 1, the content of St + 1 is moved to St + 2, etc., and the content of E is lost.



Flags

ER: St and E are not in the same data area.

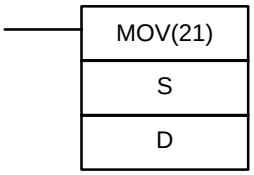
Indirectly addressed DM word is non-existent. (Content of ∗DM word is not BCD, or the DM area boundary has been exceeded.)

5-13 Data Movement

This section describes the instructions used for moving data between different addresses in data areas. These movements can be programmed within the same data area or between different data areas. Data movement is essential for utilizing all of the data areas of the PC. All of these instructions change only the content of the words to which data is being moved, i.e., the content of source words is the same before and after execution of any of the move instructions.

5-13-1 MOVE – MOV(21)

Ladder Symbol

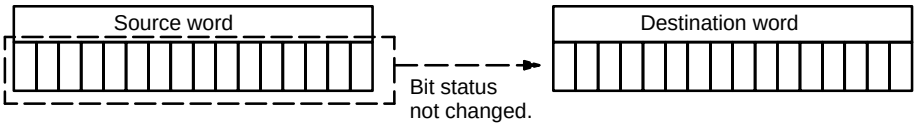


Operand Data Areas

S : Source word	
IR, SR, DM, HR, TC, #	
D : Destination word	
IR, DM, HR	

Description

When the execution condition is OFF, MOV(21) is not executed and the next instruction is moved to. When the execution condition is ON, MOV(21) transfers the content of S (specified word or four-digit hexadecimal constant) to D.



Precautions

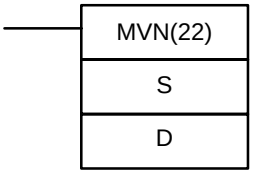
TC numbers cannot be designated as D to change the PV of the timer or counter.

Flags

EQ: ON when all zeros are transferred to D.

5-13-2 MOVE NOT – MVN(22)

Ladder Symbol

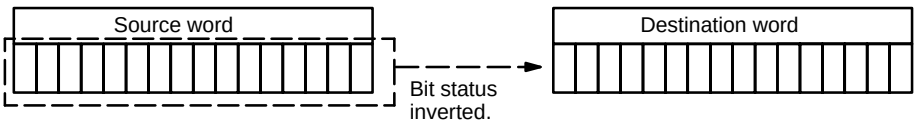


Operand Data Areas

S : Source word	
IR, SR, DM, HR, TC, #	
D : Destination word	
IR, DM, HR	

Description

When the execution condition is OFF, MVN(22) is not executed and the next instruction is moved to. When the execution condition is ON, MOV(21) transfers the inverted content of S (specified word or four-digit hexadecimal constant) 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

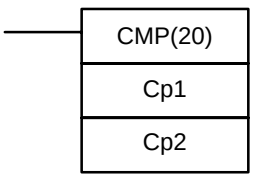
TC numbers cannot be designated as D to change the PV of the timer or counter.

EQ: ON when all zeros are transferred to D.

5-14 DATA COMPARE – CMP(20)

This section describes the instruction used for comparing data. CMP(20) is used to compare the contents of two words.

Ladder Symbols



Operand Data Areas

Cp1 : First compare word	
IR, SR, DM, HR, TC, #	
Cp2 : Second compare word	
IR, SR, DM, HR, TC, #	

Limitations

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

Description

When the execution condition is OFF, CMP(20) is not executed and the next instruction is moved to. When the execution condition is ON, CMP(20) compares Cp1 and Cp2 and outputs the result to the GR, EQ, and LE flags in the SR area.

Precautions

Placing other instructions between CMP(20) and accessing the EQ, LE, and GR flags may change the status of these flags. Be sure to access them before the desired status is changed.

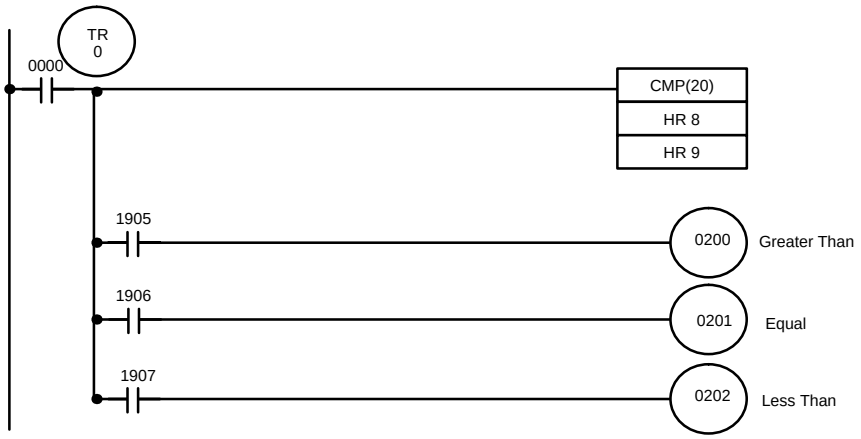
EQ: ON if Cp1 equals Cp2.

LE: ON if Cp1 is less than Cp2.

GR: ON if Cp1 is greater than Cp2.

Example 1:
Saving CMP(20) Results

The following example shows how to save the comparison result immediately. If the content of HR 8 is greater than that of 9, 0200 is turned ON; if the two contents are equal, 0201 is turned ON; if content of HR 8 is less than that of HR 9, 0202 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, 0200, 0201, and 0202 are changed only then CMP(20) is executed.



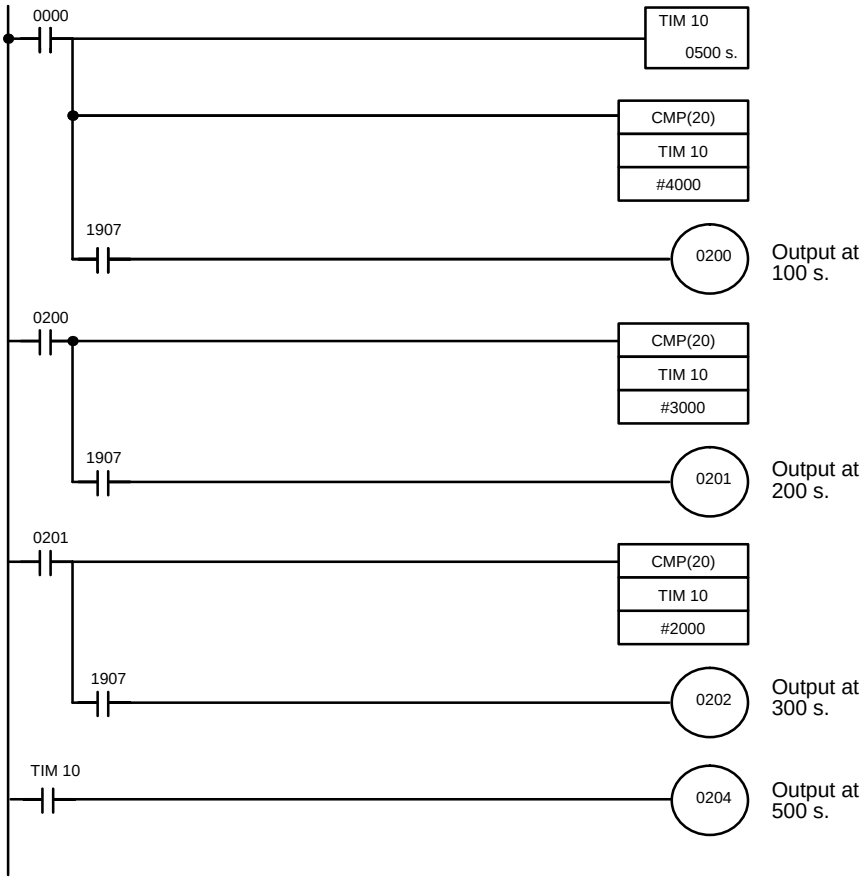
Address	Instruction	Operands	
0000	LD		0000
0001	OUT	TR	0
0002	CMP(20)		
		HR	8
		HR	9
0003	LD	TR	0
0004	AND		1905

Address	Instruction	Operands	
0005	OUT		0200
0006	LD	TR	0
0007	AND		1906
0008	OUT		0201
0009	LD	TR	0
0010	AND		1907
0011	OUT		0202

Example 2:
Obtaining Indications during Timer Operation

The following example uses TIM, CMP(20), and the LE flag (1907) to produce outputs at particular times in the timer's countdown. The timer is started by turning ON 0000. When 0000 is OFF, the TIM (10) is reset and the second two CMP(20)s are not executed (i.e., executed with OFF execution conditions). Output 0200 is output after 100 seconds; output 0201, after 200 seconds; output 0202, after 300 seconds; and output 0204, after 500 seconds.

The branching structure of this diagram is important so that 0200, 0201, and 0202 are controlled properly as the timer counts down. Because all of the comparisons here are to the timer's PV, the other operand for each CMP(20) must be in 4-digit BCD.



Address	Instruction	Operands	
0000	LD		0000
0001	TIM		10
		#	0500
0002	CMP(20)		
		TIM	10
		#	4000
0003	AND		1907
0004	OUT		0200
0005	LD		0200
0006	CMP(20)		
		TIM	10
		#	3000

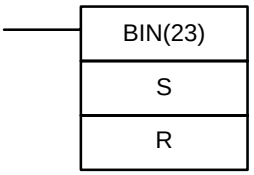
Address	Instruction	Operands	
0007	AND		1907
0008	OUT		0201
0009	LD		0201
0010	CMP(20)		
		TIM	10
		#	2000
0011	AND		1907
0012	OUT		0202
0013	LD	TIM	10
0014	OUT		0204

5-15 Data Conversion

The conversion instructions convert word data that is in one format into another format and output the converted data to specified result word(s). Conversions are available to convert between binary (hexadecimal) and BCD and between multiplexed and non-multiplexed data. All of these instructions change only the content of the words to which converted data is being moved, i.e., the content of source words is the same before and after execution of any of the conversion instructions.

5-15-1 BCD-TO- BINARY – BIN(23)

Ladder Symbol



Operand Data Areas

S : Source word (BCD)
IR, SR, DM, HR, TC

R : Result word
IR, DM, HR

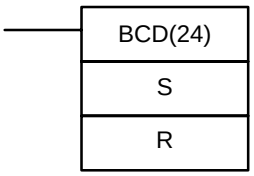
Description BIN(23) can be used to convert BCD to binary so that displays on the Programming Console or any other programming device will appear in hexadecimal rather than decimal. It can also be used to convert to binary to perform binary arithmetic operations rather than BCD arithmetic operations, e.g., when BCD and binary values must be added.

Flags **ER**: The content S is not BCD

EQ: ON when 0000 is placed in R.

5-15-2 BINARY-TO-BCD – BCD(24)

Ladder Symbol



Operand Data Areas

S : Source word (binary)
IR, SR, DM, HR, TC

R : Result word
IR, DM, HR

Limitations If the content of S exceeds 270F, the converted result would exceed 9999 and BCD(24) will not be executed. When the instruction is not executed, the content of R remains unchanged.

Description BCD(24) converts the binary (hexadecimal) content of S into the numerically equivalent BCD bits, and outputs the BCD bits to R. Only the content of R is changed; the content of S is left unchanged.

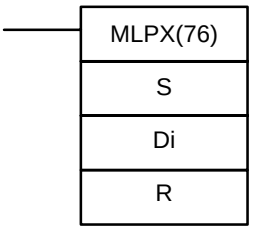
BCD(24) can be used to convert binary to BCD so that displays on the Programming Console or any other programming device will appear in decimal rather than hexadecimal. It can also be used to convert to BCD to perform BCD arithmetic operations rather than binary arithmetic operations, e.g., when BCD and binary values must be added.

Flags **ER**: S is greater than 270F.

EQ: ON when 0000 is placed in R.

5-15-3 4-TO-16 DECODER – MLPX(76)

Ladder Symbol



Operand Data Areas

S : Source word	IR, SR, DM, HR, TC
Di : Digit designator	IR, DM, HR, TC, #
R : First result word	IR, DM, HR

Limitations

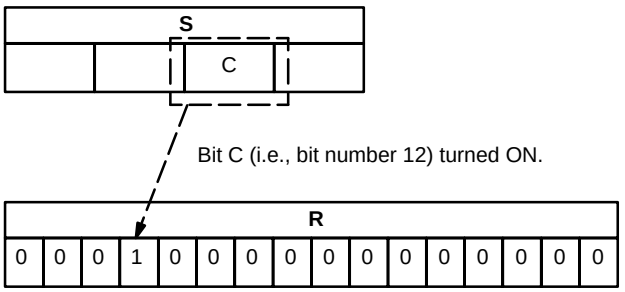
The rightmost two digits of Di must each be between D and 3.

All result words must be in the same data area.

Description

When the execution condition is OFF, MLPX(76) is not executed and the next instruction is moved to. When the execution condition is ON, MLPX(76) converts up to four, four-bit hexadecimal digits from S into decimal values from 0 to 15, each of which is used to indicate a bit position. The bit whose number corresponds to each converted value is then turned ON in a result word. If more than one digit is specified, then one bit will be turned ON in each of consecutive words beginning with R. (See examples, below.)

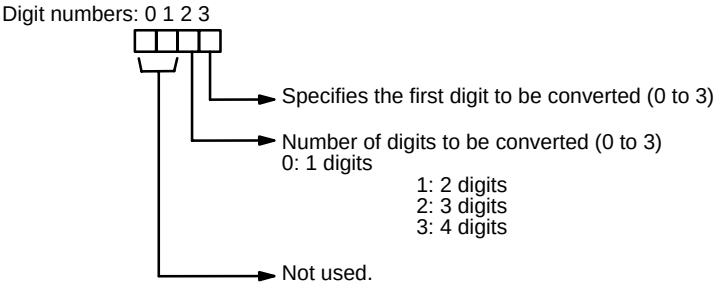
The following is an example of a one-digit decode operation from digit number 1 of S, i.e., here Di would be 0001.



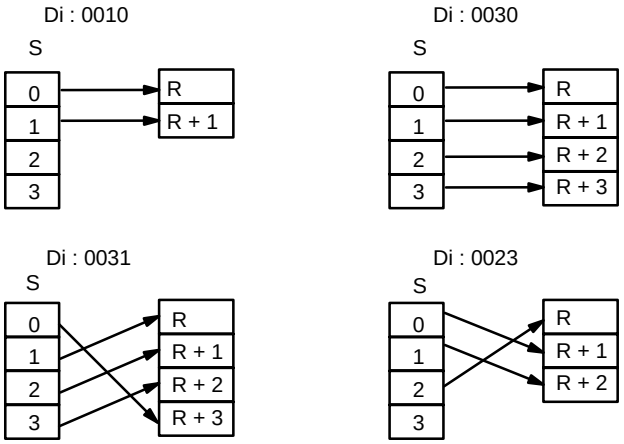
The first digit and the number of digits to be converted are designated in Di. If more digits are designated than remain in S (counting from the designated first digit), the remaining digits will be taken starting back at the beginning of S. The final word required to store the converted result (R plus the number of digits to be converted) must be in the same data area as R, e.g., if two digits are converted, the last word address in a data area cannot be designated; if three digits are converted, the last two words in a data area cannot be designated.

Digit Designator

The digits of Di are set as shown below.



Some example Di values and the digit-to-word conversions that they produce are shown below.



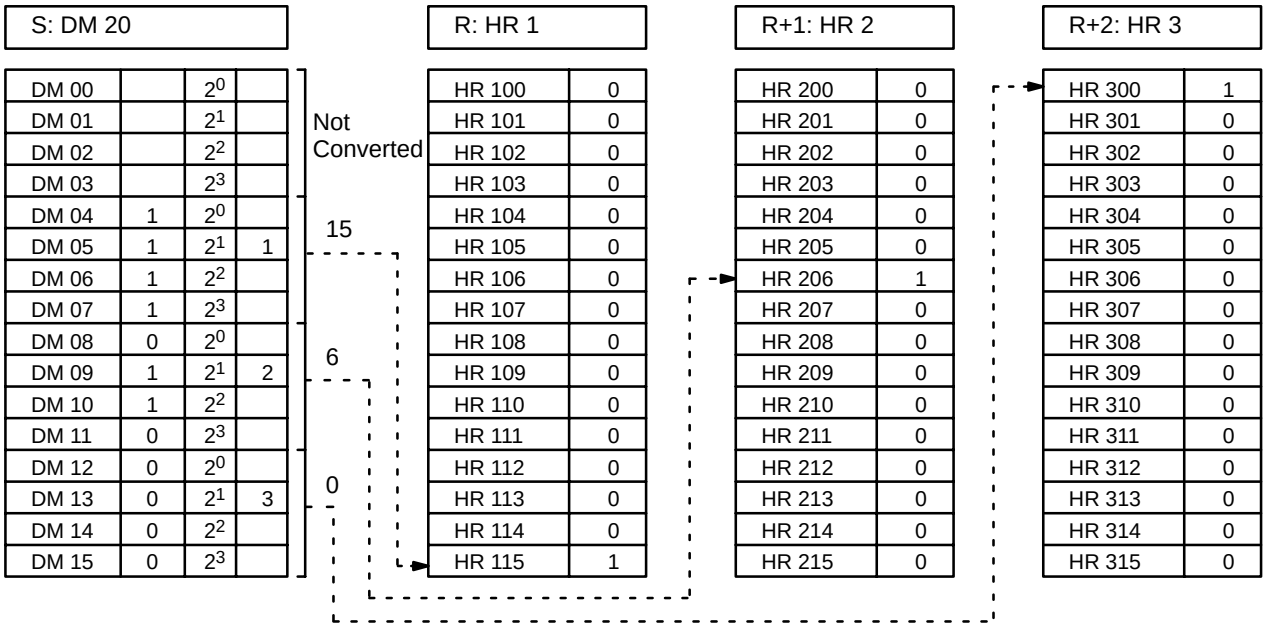
Flags

ER: Undefined digit designator, or R plus number of digits exceeds a data area.

Example

The following program converts three digits of data from DM 20 to bit positions and turns ON the corresponding bits in three consecutive words starting with HR 1.

Address	Instruction	Operands	
0000	LD		0000
0001	MLPX(76)		
		DM	20
		#	0021
		HR	1



S: DM 20			
DM 00		2^0	
DM 01		2^1	
DM 02		2^2	
DM 03		2^3	
DM 04	1	2^0	
DM 05	1	2^1	1
DM 06	1	2^2	
DM 07	1	2^3	
DM 08	0	2^0	
DM 09	1	2^1	2
DM 10	1	2^2	
DM 11	0	2^3	
DM 12	0	2^0	
DM 13	0	2^1	3
DM 14	0	2^2	
DM 15	0	2^3	

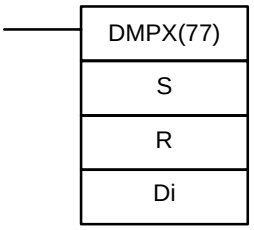
R: HR 1	
HR 100	0
HR 101	0
HR 102	0
HR 103	0
HR 104	0
HR 105	0
HR 106	0
HR 107	0
HR 108	0
HR 109	0
HR 110	0
HR 111	0
HR 112	0
HR 113	0
HR 114	0
HR 115	1

R+1: HR 2	
HR 200	0
HR 201	0
HR 202	0
HR 203	0
HR 204	0
HR 205	0
HR 206	1
HR 207	0
HR 208	0
HR 209	0
HR 210	0
HR 211	0
HR 212	0
HR 213	0
HR 214	0
HR 215	0

R+2: HR 3	
HR 300	1
HR 301	0
HR 302	0
HR 303	0
HR 304	0
HR 305	0
HR 306	0
HR 307	0
HR 308	0
HR 309	0
HR 310	0
HR 311	0
HR 312	0
HR 313	0
HR 314	0
HR 315	0

5-15-4 16-TO-4 ENCODER – DMPX(77)

Ladder Symbol



Operand Data Areas

S : First source word	IR, SR, DM, HR, TC
R : Result word	IR, DM, HR
Di : Digit designator	IR, DM, HR, TC, #

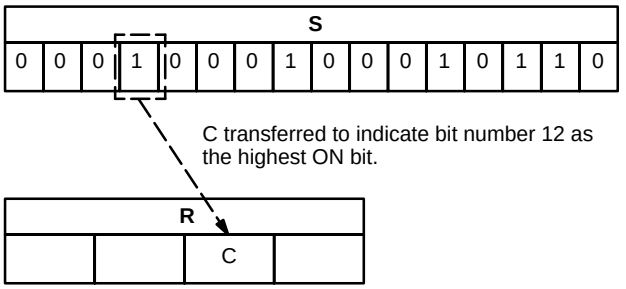
Limitations

The rightmost two digits of Di must each be between 0 and 3. All source words must be in the same data area.

Description

When the execution condition is OFF, DMPX(77) is not executed and the next instruction is moved to. When the execution condition is ON, DMPX(77) determines the position of the highest ON bit in S, encodes it into single-digit hexadecimal value corresponding to the bit number of the highest ON bit number, then transfers the hexadecimal value to the specified digit in R. The digits to receive the results are specified in Di, which also specifies the number of digits to be encoded.

The following is an example of a one-digit encode operation to digit number 1 of R, i.e., here Di would be 0001.

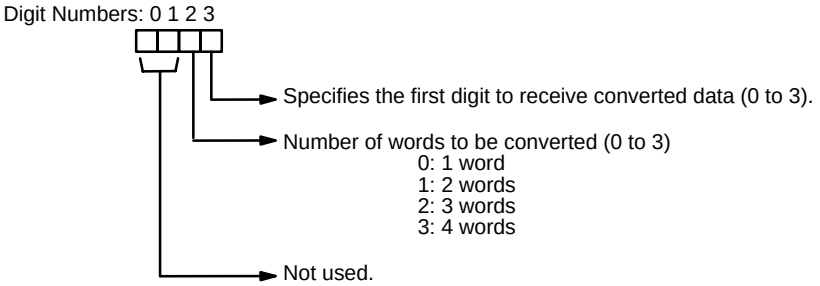


Up to four digits from four consecutive source words starting with S may be encoded and the digits written to R in order from the designated first digit. If more digits are designated than remain in R (counting from the designated first digit), the remaining digits will be placed at digits starting back at the beginning of R.

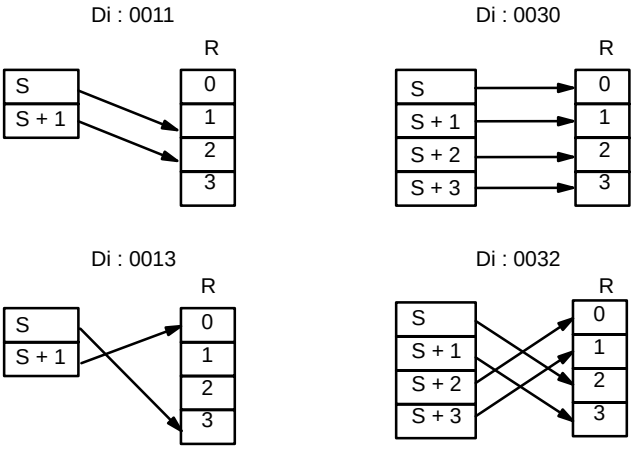
The final word to be converted (S plus the number of digits to be converted) must be in the same data area as SB.

Digit Designator

The digits of Di are set as shown below.



Some example Di values and the word-to-digit conversions that they produce are shown below.



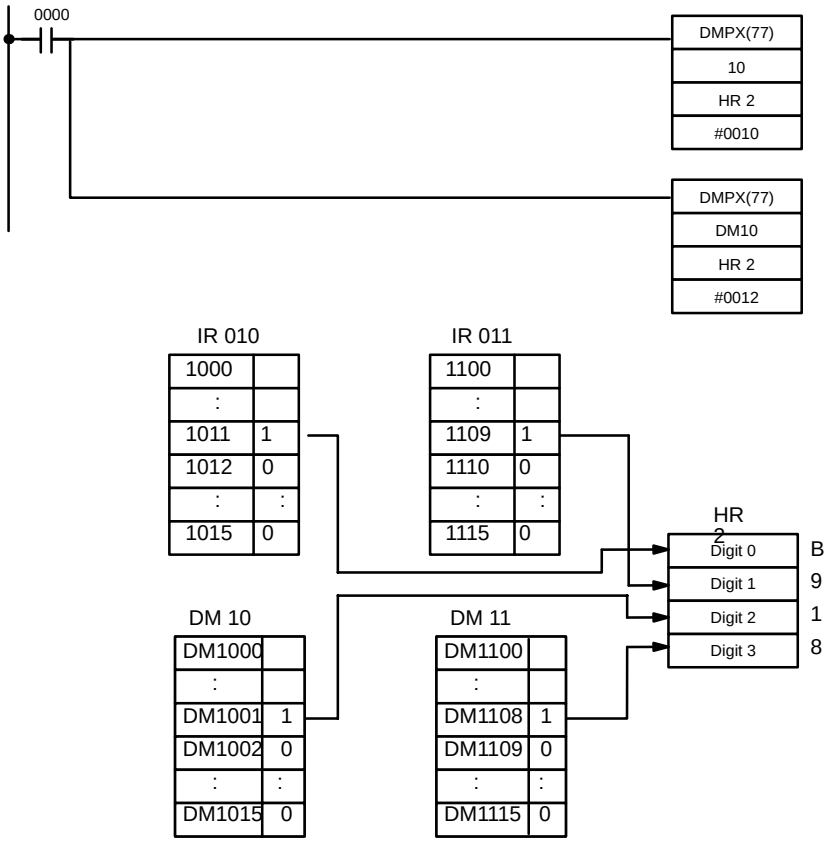
Flags

ER: Undefined digit designator, or S plus number of digits exceeds a data area.

Content of a source word is 0000.

Example

When 0000 is ON, the following diagram encodes IR words 10 and 11 to the first two digits of HR 2 and then encodes DM 10 and 11 to the last two digits of HR 2. Although the status of each source word bit is not shown, it is assumed that the bit with status 1 (ON) shown is the highest bit that is ON in the word.



Address	Instruction	Operands	
0000	LD		0000
0001	DMPX(77)		
			10
		HR	2
		#	0010
0002	DMPX(77)		
		DM	10
		HR	2
		#	0012

5-16 BCD Calculations

The BCD calculation instructions perform mathematic operations on BCD data.

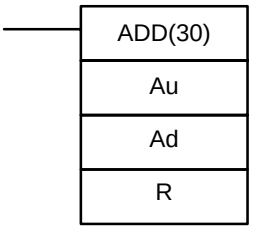
These instructions change only the content of the words in which results are placed, i.e., the contents of source words are the same before and after execution of any of the BCD calculation instructions.

STC(40) and CLC(41), which set and clear the carry flag, are included in this group because most of the BCD operations make use of the carry flag (CY) in their results. Binary arithmetic and shift operations also use CY.

The addition and subtraction instructions use CY in the calculation as well as in the result. Be sure to clear CY if its previous status is not required in the calculation, and to use the result placed in CY, if required, before it is changed by execution of any other instruction.

5-16-1 BCD ADD – ADD(30)

Ladder Symbol



Operand Data Areas

Au : Augend word (BCD)	
IR, SR, DM, HR, TC, #	
Ad : Addend word (BCD)	
IR, SR, DM, HR, TC, #	
R : Result word	
IR, DM, HR	

Description

When the execution condition is OFF, ADD(30) is not executed and the next instruction is moved to. When the execution condition is ON, ADD(30) 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 and Ad should not be designated as constants. This instruction will be executed every cycle as long as the execution condition remains ON. If the instruction is to be executed only once for a given ON execution condition then it must be used in conjunction with DIFU(13) or DIFD(14).

Flags

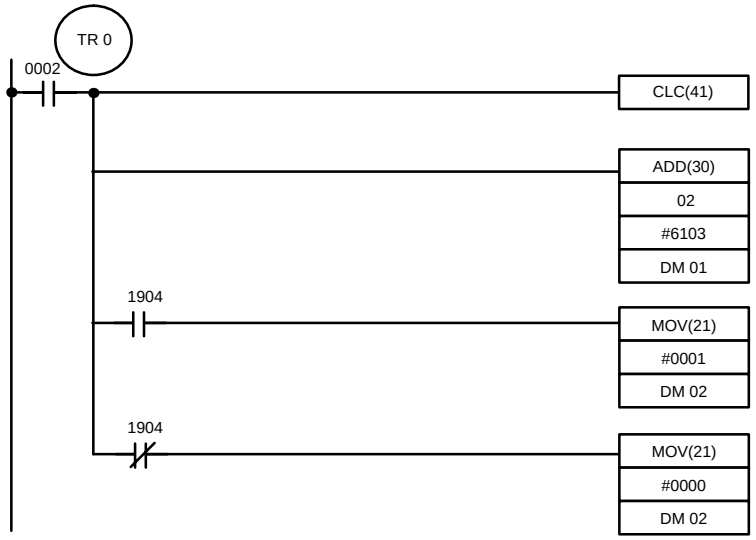
ER: Au and/or Ad is not BCD.

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

EQ: ON when the result is 0.

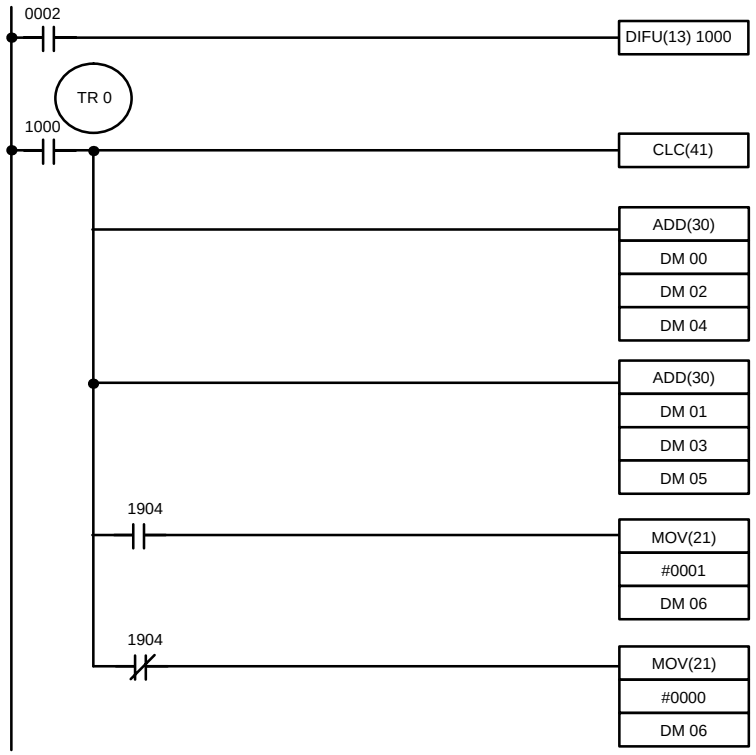
Example

If 0002 is ON, the following diagram clears CY with CLC(41), adds the content of IR 02 to a constant (6103), places the result in DM 01, and then moves either all zeros or 0001 into DM 02 depending on the status of CY (1904). 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	
0000	LD		0002
0001	OUT	TR	0
0002	CLC(41)		
0003	AND(30)		
			02
		#	6103
		DM	01
0004	AND		1904
0005	MOV(21)		
		#	0001
		DM	02
0006	LD	TR	0
0007	AND NOT		1904
0008	MOV(21)		
		#	0000
		DM	02

Consecutive ADD(30)s can be used to perform eight-digit BCD addition. By using two ADD(30)s and combining the augend and the addend words of one ADD(30) with those of the other, two 8-digit values can be added. The result may or may not be 9 digits depending on whether a carry is generated.



Address	Instruction	Operands	
0000	LD		0002
0001	DIFU(13)		1000
0003	LD		1000
0004	OUT	TR	0
0005	CLC(41)		
0006	AND(30)		
		DM	00
		DM	02
		DM	04
0007	AND(30)		
		DM	01
		DM	03
		DM	05
0008	AND		1904
0009	MOV(21)		
		#	0001
		DM	06
0010	LD	TR	0
0011	AND NOT		1904
0012	MOV(21)		
		#	0000
		DM	06

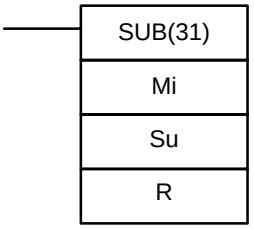
In the above program the 8 digit augend consists of two words: DM 00 and DM 01, with DM 01 being used for the 4 left digits and 00 for the 4 right digits. Similarly the 8-digit addend consist of DM 02 and 03. Three words are used to hold the results of the addition: DM 04, DM 05, and DM 06. In this

case DM 05 and DM 04 are used to represent the intermediate 4 digits and the 4 right digits respectively. DM 06 represents the leftmost digit, the 9th digit.

If a carry is generated, SR 1904 (CY) is turned ON and the constant 0001 is transferred to DM 06. If a carry is not generated SR 1904 remains OFF and the constant 0000 is transferred to DM 06.

5-16-2 BCD SUBTRACT – SUB(31)

Ladder Symbol



Operand Data Areas

Mi : Minuend word (BCD)
IR, SR, DM, HR, TC, #
Su : Subtrahend word (BCD)
IR, SR, DM, HR, TC, #
R : Result word
IR, DM, HR

Description

When the execution condition is OFF, SUB(31) is not executed and the next instruction is moved to. When the execution condition is ON, SUB(31) 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). This instruction will be executed every cycle as long as the execution condition remains ON. If the instruction is to be executed only once then it must be used in conjunction with DIFU(13) or DIFD(14).

Flags

ER: Mi and/or Su is not BCD.

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

EQ: ON when the result is 0.

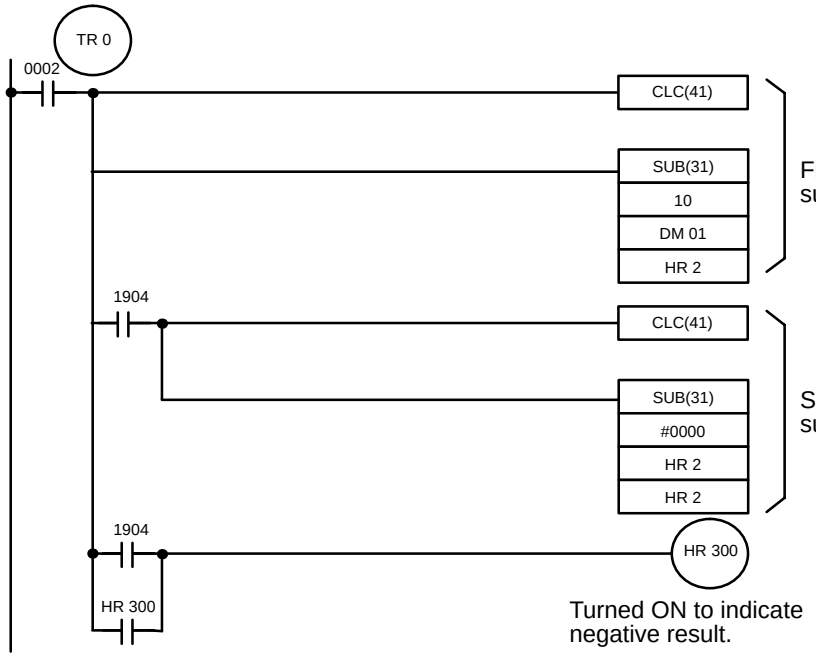
⚠ Caution Be sure to clear the carry flag with CLC(41) before executing SUB(31) if its previous status is not required, and check the status of CY after doing a subtraction with SUB(31). If CY is ON as a result of executing SUB(31) (i.e., if the result is negative), the result is output as the 10's complement 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 diagram clears CY, subtracts the contents of DM 01 and CY from the content of IR 10 and places the result in HR 2.

If CY is set by executing SUB(31), the result in HR 2 is subtracted from zero (note that CLC(41) is again required to obtain an accurate result), the result is placed back in HR 2, and HR 300 is turned ON to indicate a negative result.

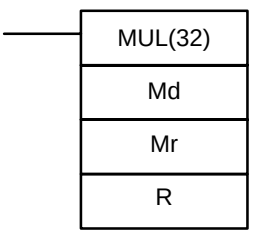
If CY is not set by executing SUB(31), the result is positive, the second subtraction is not performed and HR 300 is not turned ON. HR 300 is programmed as a self-maintaining bit so that a change in the status of CY will not turn it OFF when the program is recycled.



Address	Instruction	Operands	
0000	LD		0002
0001	OUT	TR	0
0002	CLC(41)		
0003	@SUB(31)		
			10
		DM	01
		HR	2
0004	AND		1904
0005	CLC(41)		
0006	@SUB(31)		
		#	0000
		HR	2
		HR	2
0007	LD	TR	0
0008	AND		1904
0009	OR	HR	300
0010	OUT	HR	300

5-16-3 BCD MULTIPLY – MUL(32)

Ladder Symbol



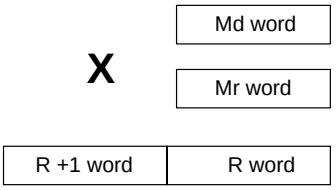
Operand Data Areas

Md : Multiplicand word (BCD)	IR, DM, HR, TC, #
Mr : Multiplier word (BCD)	IR, DM, HR, TC, #
R : First result word (BCD)	IR, DM, HR

Limitations R and R + 1 must be in the same data area.

Description

When the execution condition is OFF, MUL(32) is not executed and the next instruction is moved to. When the execution condition is ON, the contents of Md and Mr are multiplied and the rightmost four digits of the result are placed in R; the leftmost four digits, in R + 1.



Flags

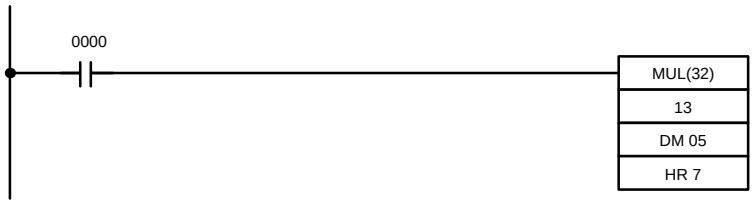
ER: Md or Mr is not in BCD.

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

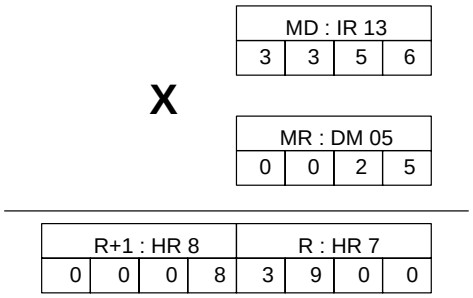
EQ: ON when the result is 0.

Example

When IR 0000 is ON with the following program, the contents of IR 13 and DM 05 are multiplied and the result is placed in HR 7 and HR 8. Example data and calculations are shown below the program.

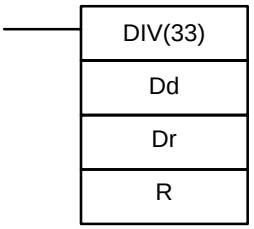


Address	Instruction	Operands
0000	LD	0000
0001	MUL(32)	
		13
		DM 05
		HR 7



5-16-4 BCD DIVIDE – DIV(33)

Ladder Symbol



Operand Data Areas

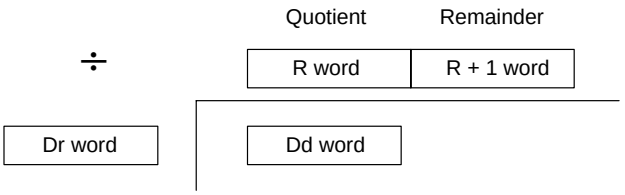
Dd : Dividend word (BCD)
IR, SR, DM, HR, TC, #
Dr : Divisor word (BCD)
IR, SR, DM, HR, TC, #
R : First result word (BCD)
IR, DM, HR

Limitations

R and R + 1 must be in the same data area.

Description

When the execution condition is OFF, DIV(33) is not executed and the next instruction is moved to. When the execution condition is ON, the content of Dd is divided by the content of Dr and the result is placed in R and R + 1: the quotient in R and the remainder in R + 1.



Flags

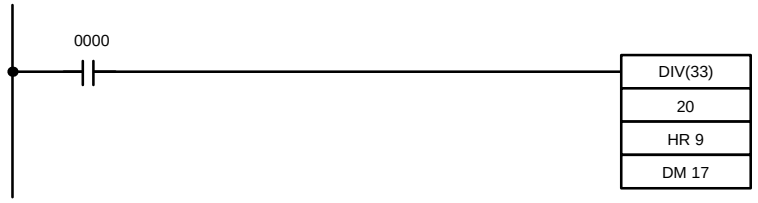
ER: Dd or Dr is not in BCD.

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

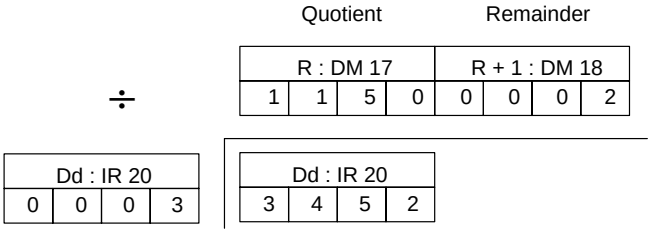
EQ: ON when the result is 0.

Example

When IR 0000 is ON with the following program, the content of IR 20 is divided by the content of HR 9 and the result is placed in DM 17 and DM 18. Example data and calculations are shown below the program.

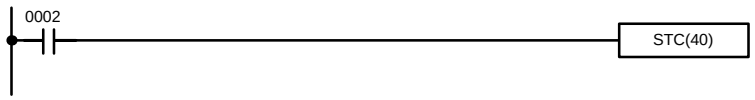


Address	Instruction	Operands
0000	LD	0000
0001	DIV(33)	
		20
		HR 9
		DM 17



5-16-5 SET CARRY – STC(40)

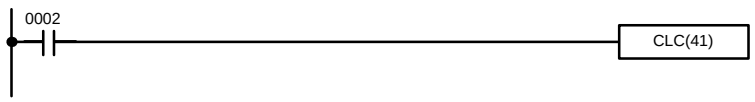
Set carry is used to set (turn ON) the CY (SR bit 1904) to “1.”



Address	Instruction	Operands
0000	LD	0002
0001	STC(40)	

5-16-6 CLEAR CARRY – CLC(41)

Clear carry is used to reset (turn OFF) the CY (SR bit 1904) to “0.”



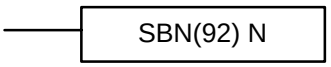
Address	Instruction	Operands
0000	LD	0002
0001	CLC(41)	

5-17 Subroutines

Subroutines can be used for one of two different purposes: either to separate off sections of large control tasks so that they can be handled as smaller ones and to enable you to reuse a given set of instructions at different places within one program or as a part of different programs. When the main program calls a subroutine, control is transferred to the subroutine and the instructions in the subroutine are executed. The instructions within a subroutine are written in the same way as main program code. When all the instructions in the subroutine have been executed, control returns to the main program.

5-17-1 SUBROUTINE DEFINE and SUBROUTINE RETURN SBN(92)/RET(93)

Ladder Symbol



Definer Data Areas

N: Subroutine number
(00 to 15)

Ladder Symbol

Limitations

Each subroutine number can be used in SBN(92) once only, i.e., up to 16 subroutines may be programmed.

Description

SBN(92) is used to mark the beginning of a subroutine program; RET(93) is used to mark the end. Each subroutine is identified with a subroutine number, N, that is programmed as a definer for SBN(92). This same subroutine number is used in any SBS(91) that calls the subroutine (see next section). No subroutine number is required with RET(93).

All subroutines must be programmed at the end of the main program. When one or more subroutines have been programmed, the main program will be executed up to the first SBN(92) before returning to address 0000 for the next cycle. Subroutines will not be executed unless called by SBS(91).

END(01) must be placed at the end of the last subroutine program, i.e., after the last RET(93). It is not required at any other point in the program.

Flags

There are no flags directly affected by these instructions.

5-17-2 SUBROUTINE ENTRY – SBS(91)

Ladder Symbol

Definer Data Areas

N: Subroutine number
(00 to 15)

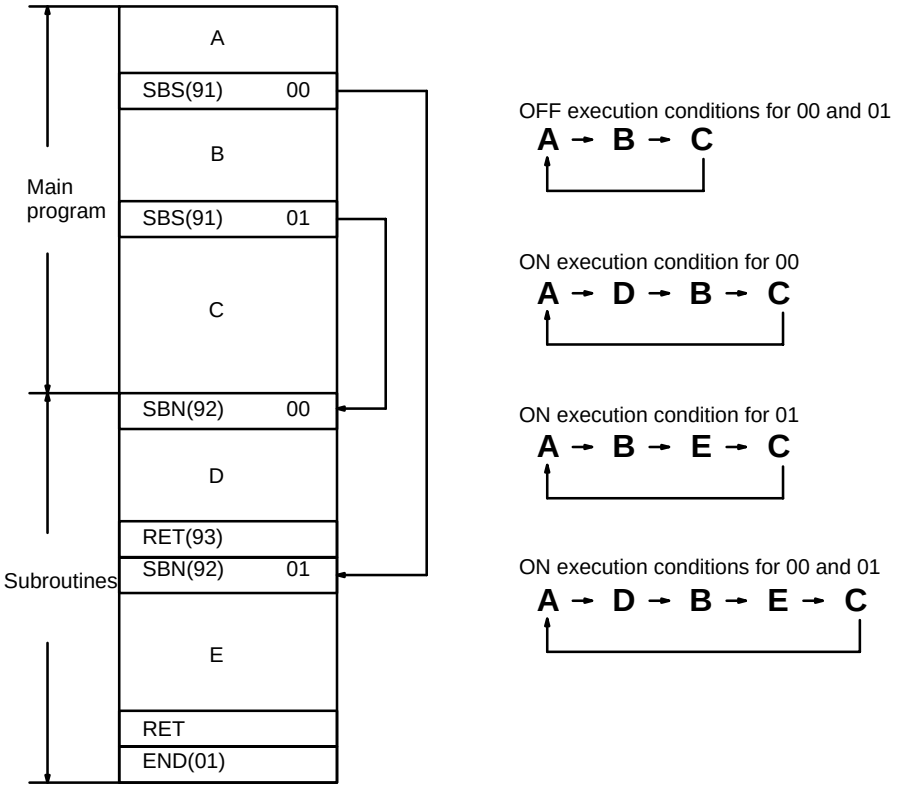
Description

To execute a subroutine, it must be called from the main by programming SBS(91) with the number of the desired subroutine. When SBS(91) is executed (i.e., when the execution condition for it is ON), the instructions between the SBN(92) with the same subroutine number and the first RET(93) after it are executed before execution returns to the instruction following the SBS(91) that made the call.

SBS(91) may be used as many times as desired in the program (i.e., the same subroutine may be called from different places in the program).

SBS(91) may also be placed into a subroutine to shift program execution from one subroutine to another, i.e., subroutines may be nested. When the second subroutine has been completed (i.e., RET(93) has been reached), the first subroutine is returned to and completed before returning to the main program. Nesting is possible to up to eight levels. A subroutine cannot call itself, e.g., SBS(91) 00 cannot be programmed within the subroutine defined with SBN(92).

The following diagram illustrates program execution flow for various execution conditions for two SBS(91). In actual execution, the PC actually executes a sequential program, with the subroutines inserted in at the required locations.



Flags

ER: A subroutine does not exist for the specified subroutine number.

A subroutine has called itself.

Subroutines have been nested to more than eight levels.

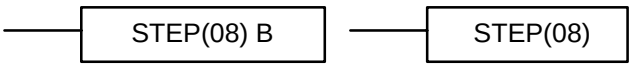
⚠ Caution SBS(91) will not be executed and the subroutine will not be called when ER is ON.

5-18 Step Instructions

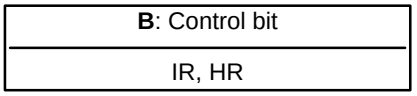
The step instructions STEP(08) and SNXT(09) are used in conjunction to set up breakpoints between sections in large programs so that the sections can be executed as units and reset upon completion. A step of program will usually be defined to correspond with an actual process in the application. (Refer to the application examples later in this section.) A section is like normal programming code except that certain instruction (e.g. IL(02)/ILC(03), JMP(04)/JME(05)) may not be included.

5-18-1 STEP DEFINE and STEP START – STEP(08)/SNXT(09)

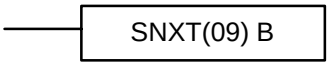
Ladder Symbols



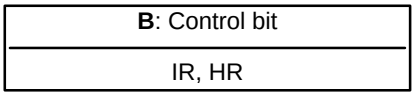
Definer Data Areas



Ladder Symbol



Definer Data Areas



Description

STEP(08) is used with a control bit in the IR or HR area to define the beginning of a section of the program called a step. STEP(08) does not require an execution condition, i.e., its execution is controlled through the control bit. To start execution of the step, SNXT(09) is used with the same control bit as the STEP(08) that defines the beginning of the step. If SNXT(09) 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(09) must be written into the program before the step it starts. It can be used at different locations before the step to control the step according to two different execution conditions (see example 2, below). Any step in the program that has not been started with SNTX(09) will not be executed.

Once SNXT(09) is used in the program, step execution will continue until STEP(08) is executed without a control bit. STEP(08) without a control bit must be preceded by SNXT(09) with a dummy control bit. The dummy control bit may be any unused IR or HR bit. It cannot be a control bit used in a STEP(08). All control bits used to produce steps, however, must be from the same word and must be used consecutively, and therefore a maximum of 16 steps can be programmed.

Execution of a step is completed either by execution of the next SNXT(09) or by turning OFF the control bit for the step (see example 3 below). When the step is completed, all of the IR and HR bits in the step are turned OFF and all timers in the step are reset to their SV. Counters, shift registers, and bits used in KEEP(11) maintain status.

More than one step can be programmed in series. Each step must start with STEP(08) and generally ends with SNXT(09) (see example 3, below, for an exception). When steps are programmed in series, three types of execution are possible: sequential, branching, or parallel. The execution conditions for and positioning of SNXT(09) determine how the steps are executed. The three examples given below demonstrate these three types of step execution.

Precautions

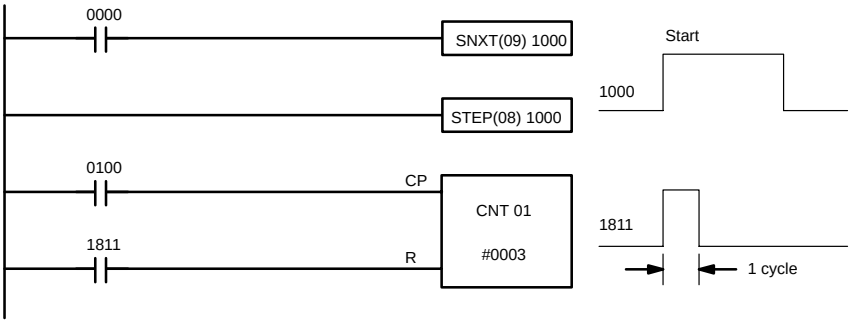
Interlocks, jumps, SBN(92), and END(01) must not be used within step programs.

Bits used as control bits must not be used anywhere else in the program unless they are used to control the step (see example 3, below).

If IR 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, HR bits must be used.

Flags

1811: Step Start flag; turns ON for one cycle when STEP(08) is executed and can be used to reset counters in steps as shown below if necessary.



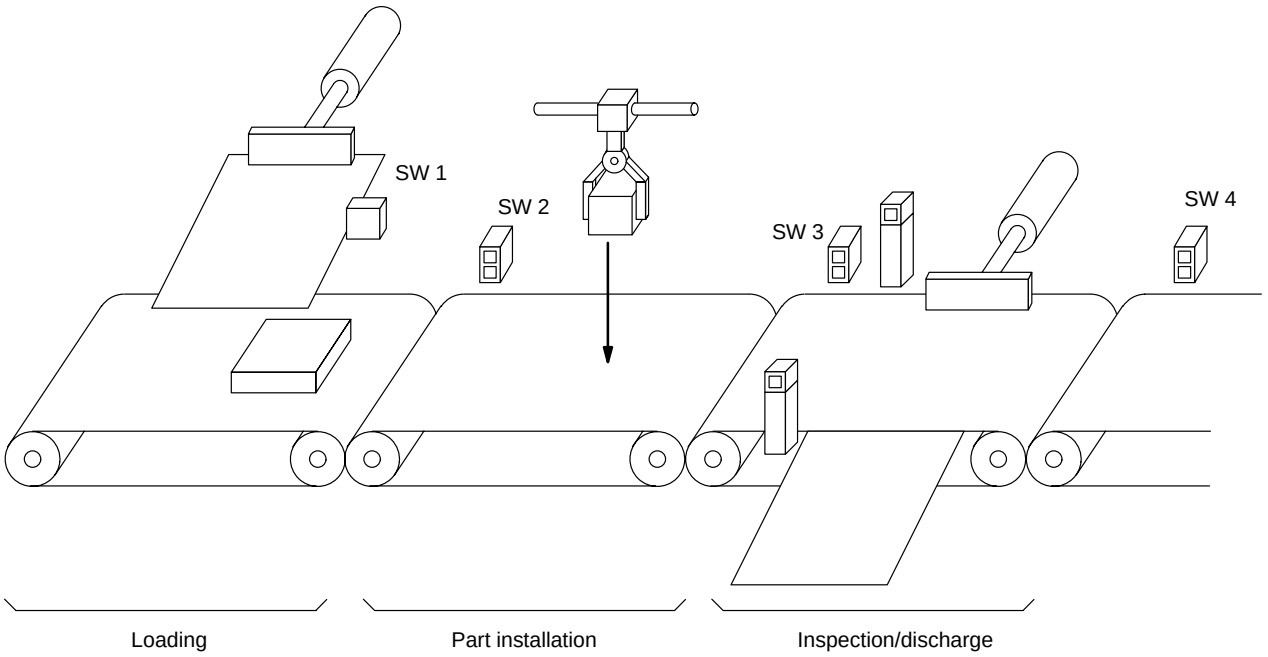
Address	Instruction	Operands
0000	LD	0000
0001	SNXT(09)	1000
0002	STEP(08)	1000
0003	LD	0100
0004	LD	1811
0005	CNT	01
		# 0003

Examples

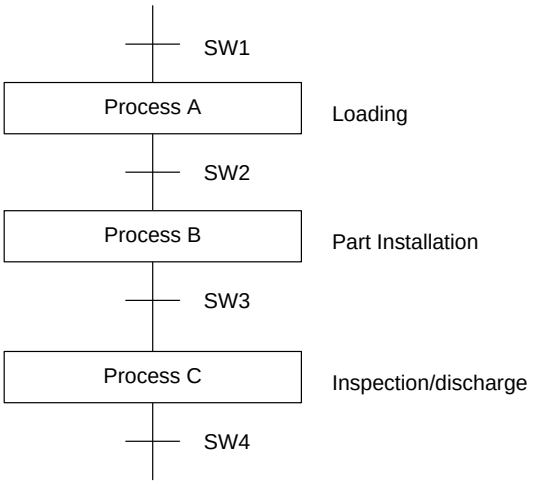
The following three examples demonstrate the three types of execution control possible with step programming. Example 1 demonstrates sequential execution; example 2, branching execution; and example 3, parallel execution.

Example 1: Sequential Execution

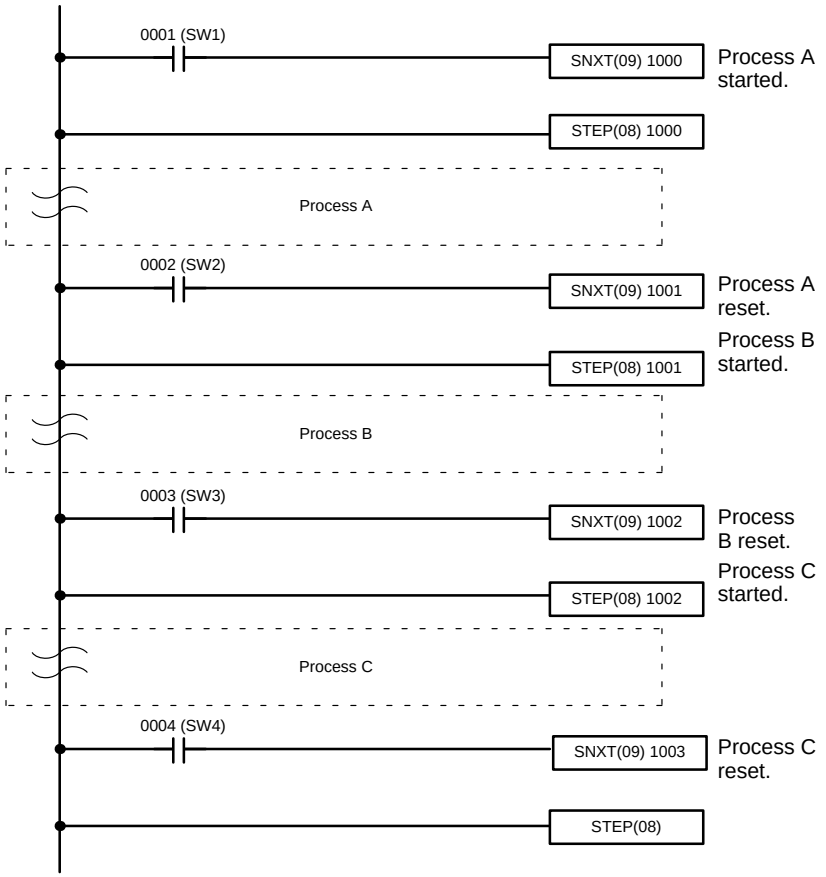
The following process requires that three processes, loading, part installation, and inspection/discharge, be executed in sequence with each process being reset before continuing on to the next process. Various sensors (SW1, SW2, SW3, and SW4) are positioned to signal when processes are to start and end.



The following diagram demonstrates the flow of processing and the switches that are used for execution control.



The program for this process, shown below, utilizes the most basic type of step programming: each step is completed by a unique SNXT(09) that starts the next step. Each step starts when the switch that indicates the previous step has been completed turns ON.



Address	Instruction	Operands
0000	LD	0001
0001	SNXT(09)	1000
0002	STEP(08)	1000

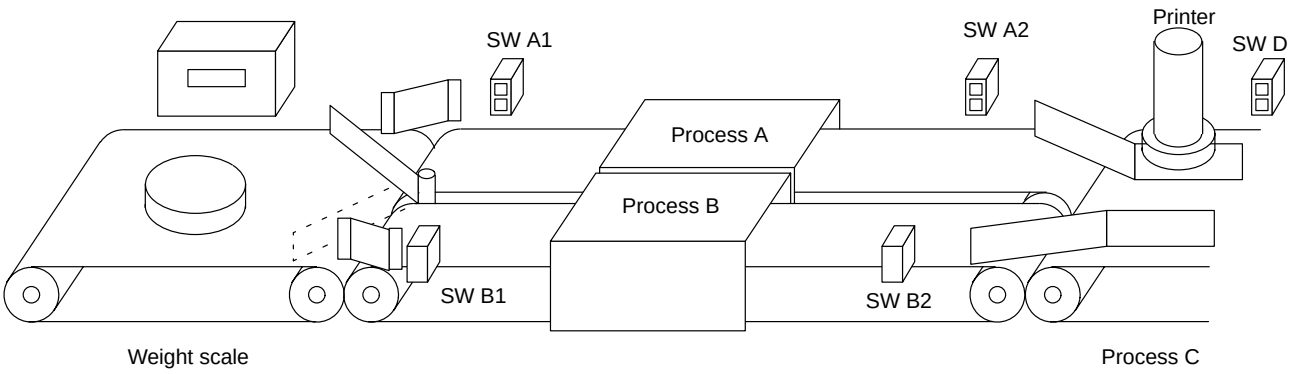
Address	Instruction	Operands
0100	LD	0002
0101	SNXT(09)	1001
0102	STEP(08)	1001

Address	Instruction	Operands
0200	LD	0003
0201	SNXT(09)	1002
0202	STEP(08)	1002

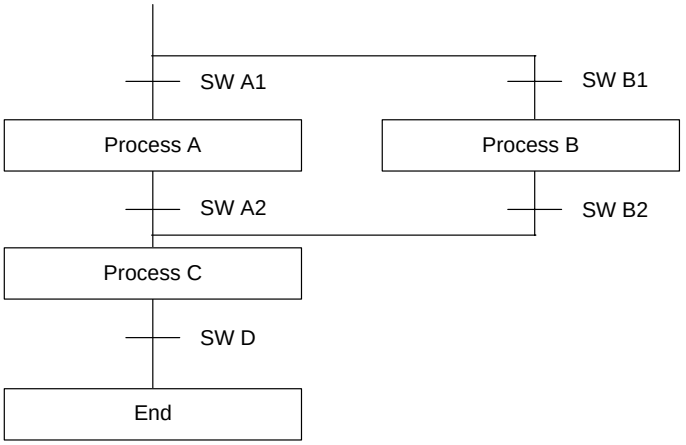
Address	Instruction	Operands
0300	LD	0004
0301	SNXT(09)	1003
0302	STEP(08)	1003

Example 2: Branching Execution

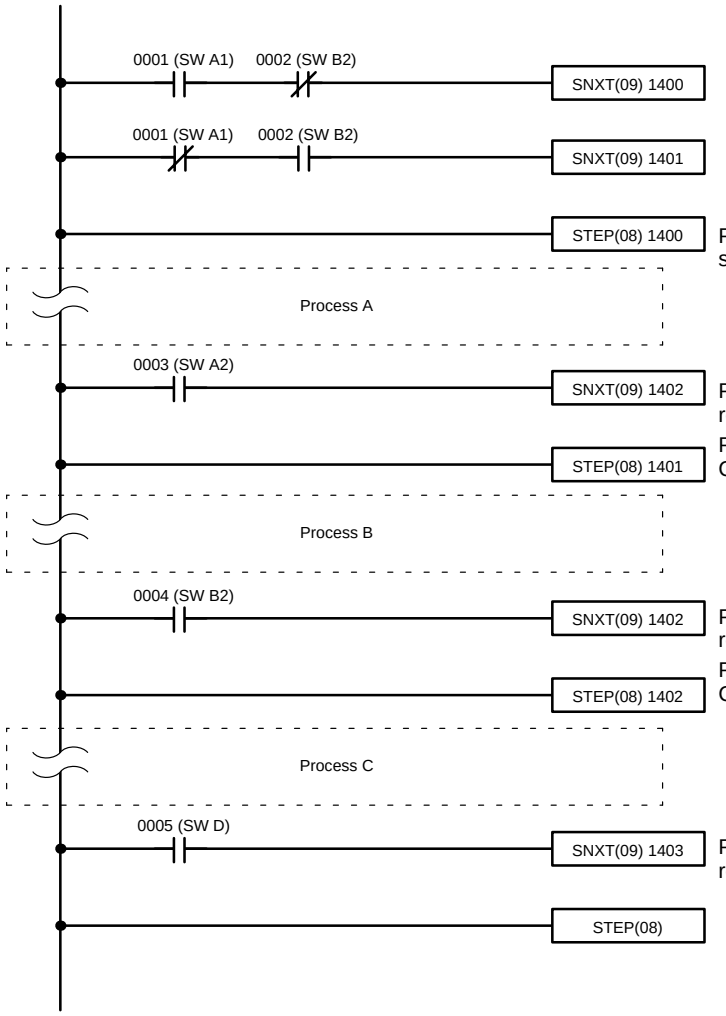
The following process requires that a product is processed in one of two ways, depending on its weight, before it is printed. The printing process is the same regardless of which of the first processes is used. Various sensors are positioned to signal when processes are to start and end.



The following diagram demonstrates the flow of processing and the switches that are used for execution control. Here, either process A or process B is used depending on the status of SW A1 and SW B1.



The program for this process, shown below, starts with two SNXT(09) that start processes A and B. Because of the way 0001 (SWA1) and 0002 (SB B1) are programmed, only one of these will be executed to start either process A or process B. Both of the steps for these processes end with a SNXT(09) that starts the step for process C.



Address	Instruction	Operands
0000	LD	0001
0001	AND NOT	0002
0002	SNXT(09)	1400
0003	LD NOT	0001
0004	AND	0002
0005	SNXT(09)	1401
0006	STEP(08)	1400

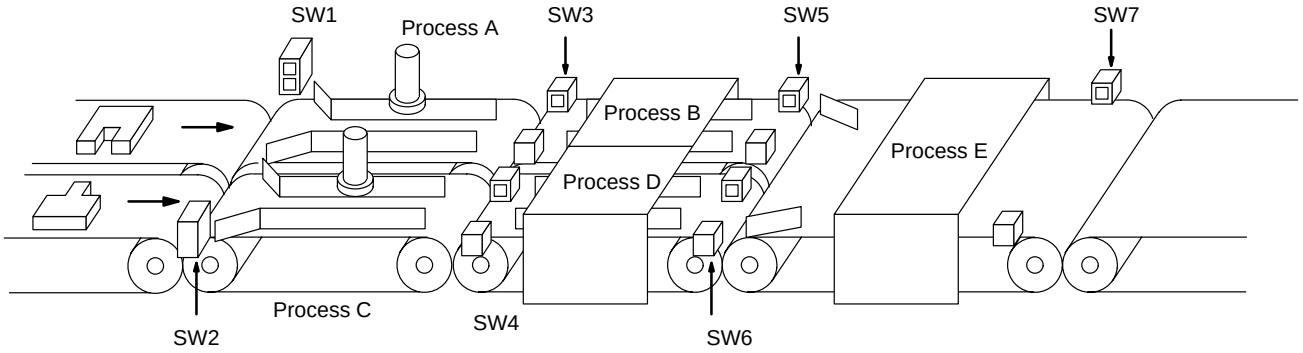
Address	Instruction	Operands
0100	LD	0003
0101	SNXT(09)	1402
0102	STEP(08)	1401

Address	Instruction	Operands
0200	LD	0004
0201	SNXT(09)	1402
0202	STEP(08)	1402

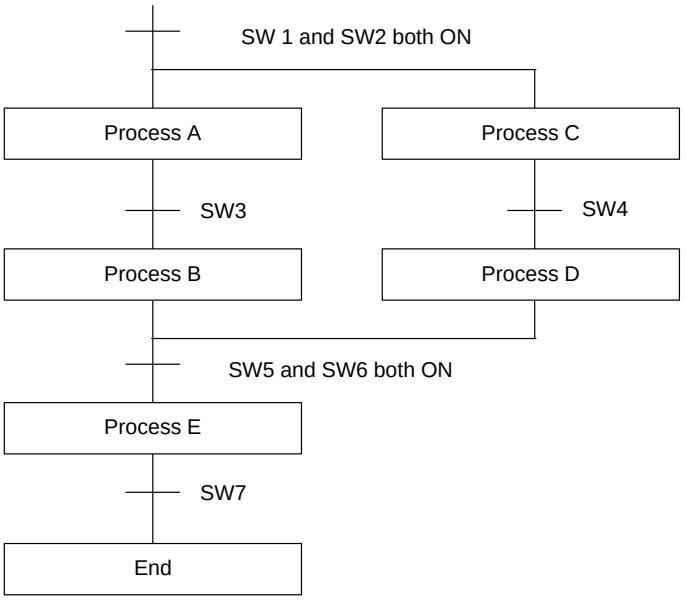
Address	Instruction	Operands
0300	LD	0005
0301	SNXT(09)	1403
0302	STEP(08)	

Example 3: Parallel Execution

The following process requires that two parts of a product pass simultaneously through two processes each before they are joined together in a fifth process. Various sensors are positioned to signal when processes are to start and end.



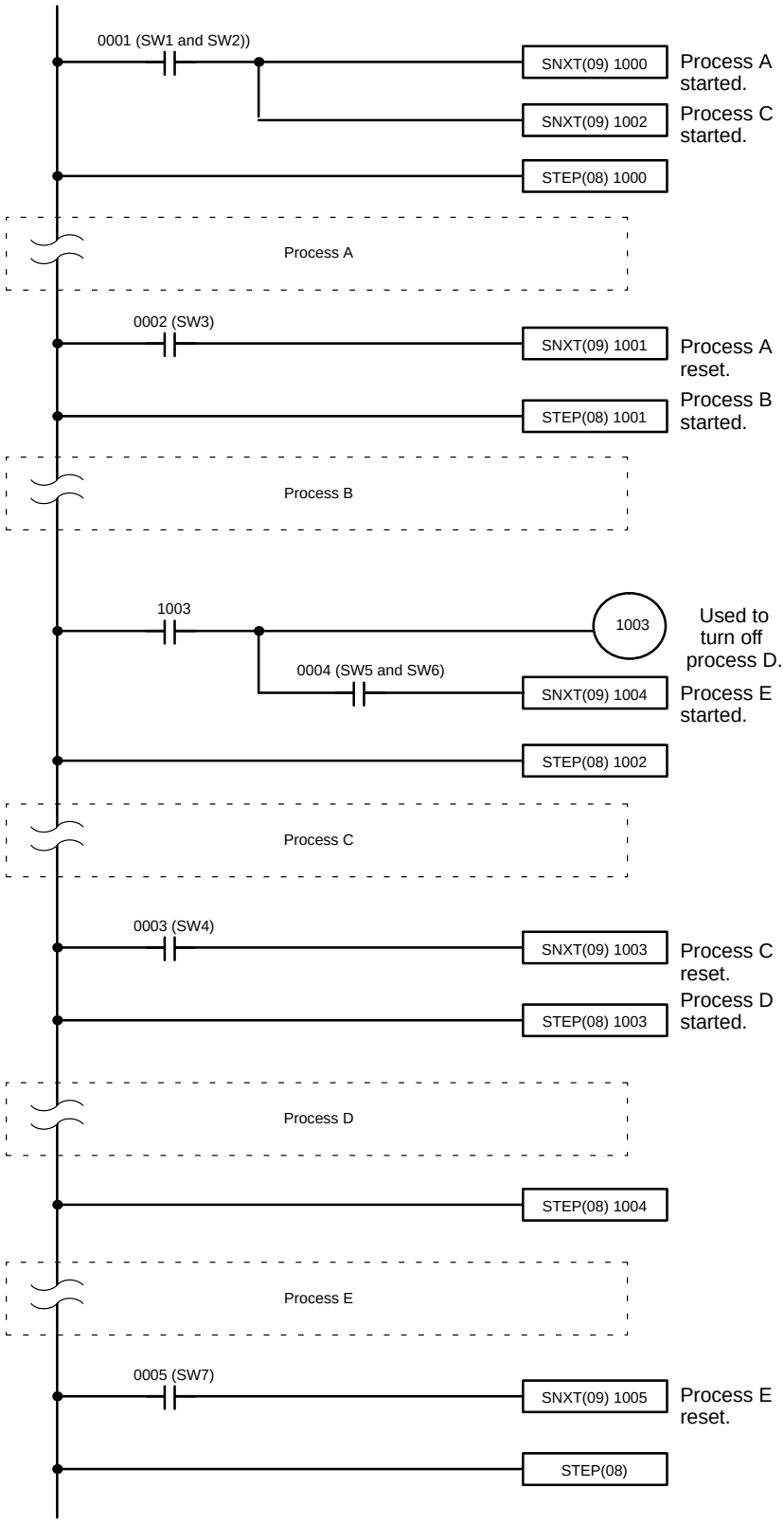
The following diagram demonstrates the flow of processing and the switches that are used for execution control. Here, process A and process C are started together. When process A finishes, process B starts; when process C finishes, process D starts. When both processes B and D have finished, process E starts.



The program for this operation, shown below, starts with two SNXT(09) that start processes A and C. These instructions branch from the same instruction line and are always executed together, starting steps for both A and C. When the steps for both A and C have finished, the steps for process B and D begin immediately.

When both process B and process D have finished (i.e., when SW5 and SW6 turn ON), processes B and D are reset together by the SNXT(09) at the end of the programming for process B. Although there is no SNXT(09) at the end of process D, the control bit for it is turned OFF by executing SNXT(09) 1002. This is because the OUT to IR 1101 is in the step reset by SNXT(09) 1002.

Thus, process B is reset directly and process B is set indirectly before executing the step for process E.



Address	Instruction	Operands
0000	LD	0001
0001	SNXT(09)	1000
0002	SNXT(09)	1002
0003	STEP(08)	1000

Address	Instruction	Operands
0100	LD	0002
0101	SNXT(09)	1001
0102	STEP(08)	1001

Address	Instruction	Operands
0200	LD	1003
0201	OUT	1003
0202	AND	0004
0203	SNXT(09)	1004
0204	STEP(08)	1002

Address	Instruction	Operands
0300	LD	0003
0301	SNXT(09)	1003
0302	STEP(08)	1003

Address	Instruction	Operands
0401	STEP(08)	1004

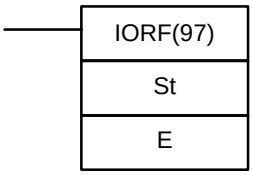
Address	Instruction	Operands
0500	LD	0005
0501	SNXT(09)	1005
0502	STEP(08)	

5-19 Special Instructions

The following instructions provide for special purposes: refreshing I/O bits during program execution, designating minimum cycle time, and inserting comments into a program.

5-19-1 I/O REFRESH – IORF(97)

Ladder Symbol



Operand Data Areas

St : Starting word
IR (00 through 09)

E : End word
IR (00 through 09)

Limitations

IORF can be used for refreshing I/O words allocated on the CPU or an Expansion I/O Rack only. It cannot be used for other I/O words. St must be less than or equal to E.

Description

When the execution condition is OFF, IORF(97) is not executed and the next instruction is moved to. When the execution condition is ON, all words between St and E will be refreshed. This will be in addition to the normal I/O refresh performed during the PC's cycle.

Execution Time

The execution time for IORF(97), T_{IORF}, is computed as follows:

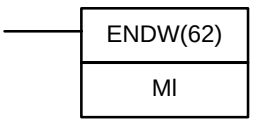
$$T_{IORF} = 40\ \mu s + (74\ \mu s \times \text{number of words refreshed})$$

Flags

There are no flags affected by this instruction.

5-19-2 END WAIT – ENDW(62)

Ladder Symbol



Operand Data Areas

Ml : Multiplier (BCD)
#

Description

ENDW(62) can be used to specify a minimum cycle time for the PC. When the execution condition is OFF, ENDW(62) is not executed and the next instruction is moved to. When the execution condition is ON, the CPU will wait at the end of program execution until the cycle time reaches 100 μs X Ml beginning the next cycle. If program execution requires less that the minimum time set, a wait will be inserted immediately after execution of END(01).

If the minimum cycle time set with ENDW(62) is exceeded by normal program execution, the cycle will continue as normal. If more than one ENDW(62) is written into the program, only the last one will be effective.

The accuracy of TIMH(15) may be affected if Ml is greater than 10. The accuracy of TIM and the response speed of the Programming Console may be affected if Ml is set to greater than 1,000.

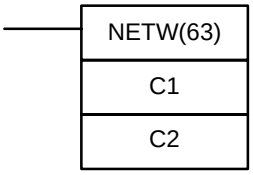
The cycle time set with ENDW(62) is accurate to plus or minus 0.05 ms.

Flags

There are no flags affected by this instruction.

5-19-3 NOTATION INSERT – NETW(63)

Ladder Symbol



Operand Data Areas

C1 : Comment 1 (Hex)
#

C2 : Comment 2 (Hex)
#

Description NETW(63) is not executed regardless of its execution condition. It is provided so that the programmer can leave comments in the program. The operands may be any hexadecimal value from 0000 through FFFF.

Flags There are no flags affected by this instruction.

SECTION 6
Program Execution Timing

6-1 Introduction . . . . . .
6-2 Cycle Time . . . . . .
6-3 Calculating Cycle Time . . . . . .
6-3-1 Single PC Unit . . . . . .
6-3-2 PC with Additional Units . . . . . .
6-4 Instruction Execution Times . . . . . .
6-5 I/O Response Time . . . . . .

6-1 Introduction

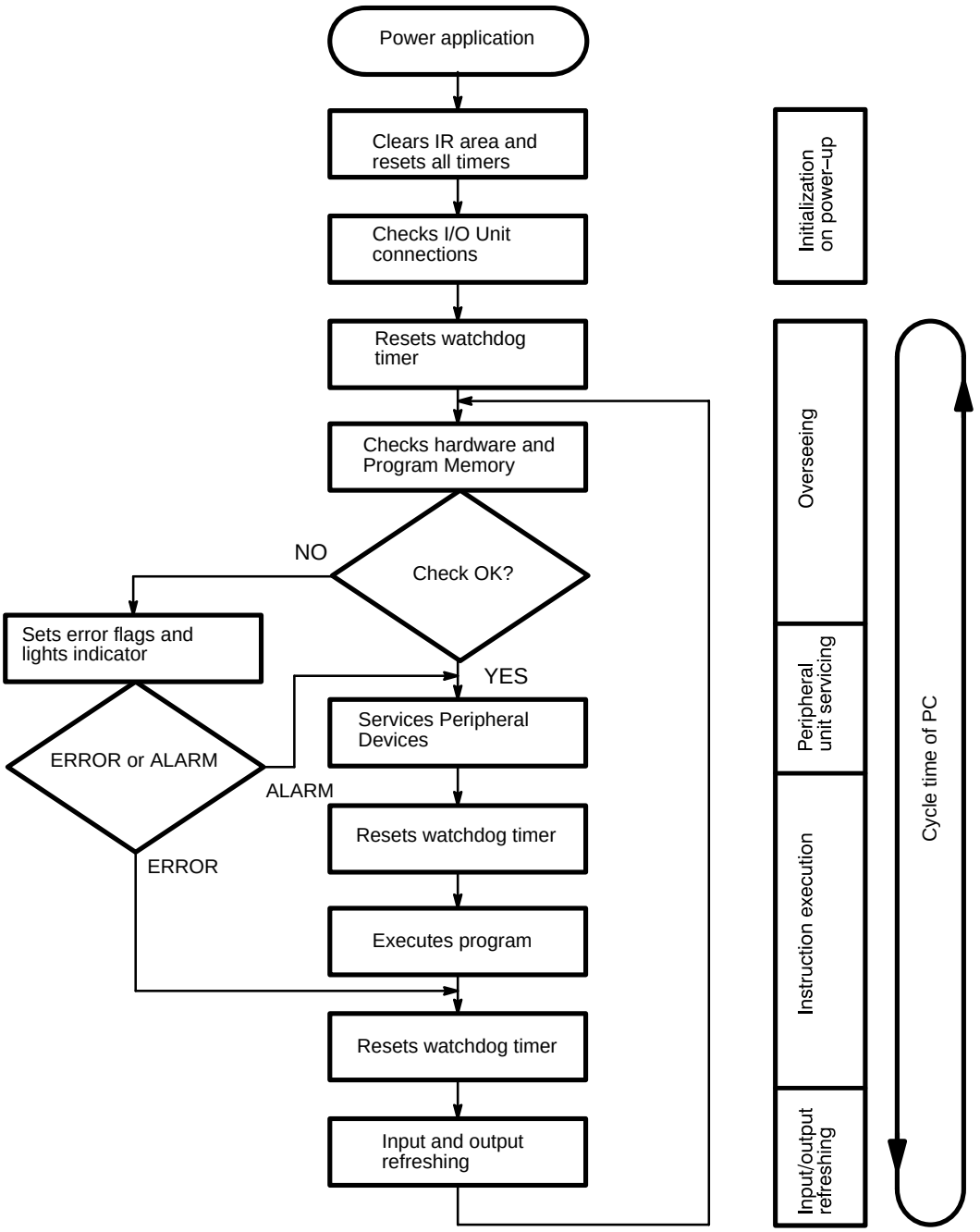
When writing and debugging a program, the timing of various operations must be considered. Not only is the time required to execute the program and perform other CPU operations important, but also the timing of each signal coming into and leaving the PC must be such that the desired control action is achieved at the right time.

The major factors in determining program timing are the cycle time and the I/O response time. One cycle of CPU operation is called a cycle; the time required for one cycle is called the cycle time. The time required to produce a control output signal following reception of an input signal is called the I/O response time. This section explains the cycle and shows how to calculate the cycle time and I/O response time.

6-2 Cycle Time

To aid in PC operation, the average cycle time can be displayed on the Programming Console or any other Programming Device. Understanding the operations that occur during the cycle and the elements that affect cycle time is essential to effective programming and PC operations.

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



The first three operations immediately after power application are performed once each time the PC is turned on. The then on the operations shown above are performed in cyclic fashion, with each cycle forming one cycle. The cycle time is the time that is required for the CPU to complete one of these cycles. This cycle includes four types of operation.

1, 2, 3...

1. Overseeing
2. Input/Output refreshing
3. Peripheral Device servicing
4. Instruction execution

Cycle time = Overseeing time + Input/output refreshing + Peripheral Device servicing time + Instruction execution time

1	**Overseeing**	Watchdog timer set and program memory and I/O bus checked.	1.6 ms (Fixed)
2	**Peripheral Device servicing**	Commands from Program Devices and Interface Units processed.	T = (1 + 3 + 4) x 0.05. T <= 1, execution time = 1 ms. T > 1, round off in units of 0.5 ms (e.g.,1.65 ms rounds to 1.5 ms) No peripherals connected = 0 ms.
3	**Instruction execution**	Instructions executed.	Total of execution time for each instruction. Varies with program size, the instructions used, and execution conditions. Refer to *6-4 Instruction Execution Times* for details.
4	**Input refreshing** **Output refreshing**	Reading data input from input terminals and writing the results of instruction execution to output terminals.	0.51 ms + 0.03 ms times N where N = number of input and output words – 2.

The cycle time can be obtained by adding the four cycle time components identified above. An adequately short cycle time is important to ensure efficient, error-free operation.

Watchdog Timer and Long Cycle Times

Within the PC, the watchdog timer measures the cycle time and compares it to a set value. If the cycle time exceeds the set value of the watchdog timer, an error is generated and the CPU stops.

Even if the cycle time does not exceed the set value of the watchdog timer, a long cycle time can adversely affect the accuracy of system operations as shown in the following table.

Cycle time (ms)	Possible adverse affects
10 or greater	TIMH(15) becomes inaccurate.
100 or greater	0.1-second clock pulse generator SR 1900 may malfunction.
Between 100 and 130	ALARM indicator on the CPU lights and SR 1809 turns ON.
130 or greater	ERROR indicator on the CPU lights and the system halts.

6-3 Calculating Cycle Time

The PC configuration, the program, and program execution conditions must be taken into consideration when calculating the cycle time. This means taking into account such things as the number of I/O points, the programming instructions used, and whether or not Peripheral Devices are employed. This subsection shows some basic cycle time calculation examples.

6-3-1 Single PC Unit

Configuration: A single C20K CPU.

Program: 300 addresses.

Instructions Used: LD and OUT.

Calculations

The equation for the cycle time from above is as follows:

Cycle time = Overseeing time
+ Input/output refreshing
+ Peripheral device servicing time
+ Instruction execution time

The overseeing time is fixed at 1.6 ms.

The input/output refresh time would be as follows: 0.51 ms + (0.03 ms x N). As the C20K is provided with only one input and one output word the value of the constant N is 0 (i.e. N = 2 – 2 = 0) and so the time required is 0.51 ms + (0.03 ms x 0) = 0.51 ms.

The execution time can be calculated by obtaining the average instruction execution time and multiplying this by the number of addresses used in the program. As only LD and OUT are used in this program and they have execution times of 12 μs and 17.5 μs respectively, the average instruction execution time is:

$$\frac{12\ \mu s + 17.5\ \mu s}{2} = 14.75\ \mu s$$

The total execution time is equal to this average instruction execution time multiplied by the number of program addresses.
Total execution time = 300 addresses x 14.75 μs = 4.43 ms

The peripheral device servicing time is calculated by adding the other three time values and multiplying the result by a factor of 0.05. This value is only required in configurations where a peripheral device is connected to the PC. The result is calculated as an example. As there are no peripheral devices used in this example the following results will be ignored in the final calculation.

Peripheral device servicing = (1.6 ms + 0.51 ms + 4.43 ms) x 0.05 = 0.3 ms
As this is less than 1 ms it must be rounded up to 1 ms. Had it been over 1 ms it would then need to be rounded down to the nearest 0.5 ms.

The cycle time is the total of all these calculations.

1.6 ms + 0.51 ms + 4.43 ms = 6.54 ms

If a peripheral device had been present it would have been:

1.6 ms + 0.51 ms + 4.43 ms + 1 ms = 7.54 ms

Process	Formula	Peripheral device servicing (ms)	
		With	Without
1. Overseeing	Fixed	1.6	1.6
2. Input/output refreshing	0.29 + 0.07 * (1–1)	0.51	0.51
3. Peripheral device servicing	((1) + (3) + (4)) * 0.05 = 0.3 < 1	1.00	0.00
4. Instruction execution	14.75 * 300	4.43	4.43
Total	(1) + (2) + (3) + (4)	7.54	6.54

6-3-2 PC with Additional Units

Configuration: A C40K CPU, a C40P Expansion I/O Unit, an I/O Link Unit.

Program: 1150 addresses.

Average instruction execution time: 30 μs.

Calculations

The equation for the cycle time from above is as follows:

Cycle time = Overseeing time
+ Input/output refreshing
+ Peripheral device servicing time
+ Instruction execution time

The overseeing time is fixed at 1.6 ms.

The input/output refresh time would be as follows: 0.51 ms + (0.03 ms x N). As the C40K is provided with only one input and one output word and the C40P Expansion unit contains input and output words the value of the constant N is 8. (i.e., N = 10 – 2 = 8) and so the time required is 0.51 ms + (0.03 ms x 8) = 0.75 ms.

The total execution time can be calculated by obtaining the average instruction execution time and multiplying this by the number of addresses used in the program. As given above the average instruction execution time is 30 μs.

Total execution time = 1150 addresses x 30 μs = 34.50 ms

The peripheral device servicing time is calculated by adding the other three time values and multiplying the result by a factor of 0.05. This value is only required in configurations where a peripheral device is connected to the PC. The result is calculated as an example. As there are no peripheral devices used in this example the following results will be ignored in the final calculation.

Peripheral device servicing = (1.6 ms + 0.75 ms + 34.50 ms) x 0.05 = 1.84 ms which is rounded down to 1.50 ms.

The cycle time is the total of all these calculations.

1.6 ms + 0.75 ms + 34.50 ms = 36.85 ms

If a peripheral device had been present it would have been:

1.6 ms + 0.75 ms + 34.50 ms + 1.50 ms = 38.35 ms

Process	Formula	Peripheral device servicing (ms)	
		With	Without
1. Overseeing	Fixed	1.6	1.6
2. Input/output refreshing	0.29 + 0.07 * (1–1)	0.75	0.75
3. Peripheral device servicing	((1) + (2) + (4)) * 0.05 = 1.8 >1	1.50	0.00
4. Instruction execution	30 * 1150	34.50	34.50
Total	(1) + (2) + (3) + (4)	38.35	36.85

6-4 Instruction Execution Times

This following table lists the execution times for all instructions that are available for the K-types. The maximum and minimum execution times and the conditions which cause them are given where relevant.

Execution times for most instructions depend on whether they are executed with an ON or an OFF execution condition. The OFF execution time for an instruction can also vary depending on the circumstances, i.e., whether it is in an interlocked program section and the execution condition for IL is OFF, whether it is between JMP(04) 00 and JME(05) 00 and the execution condition for JMP(04) 00 is OFF, or whether it is reset by an OFF execution condition. “R,” “IL,” and “JMP” are used to indicate these three times.

Execution times are expressed in microseconds except where noted.

Function code	Instruction	Execution time(μs)	Conditions
---	LD	12	Always
	LD NOT	12	Always
	AND	11.5	Always
	AND NOT	11.5	Always
	OR	11.5	Always
	OR NOT	11.5	Always
	AND LD	4	Always
	OR LD	4	Always
	OUT	17	When outputting logical “1” (ON)
		17.5	When outputting logical “0” (OFF)
	OUT NOT	19	When outputting logical “1” (ON)
		17.5	When outputting logical “0” (OFF)
	TIM	95	When timing
		95.5 to 186.5	When reset
	CNT	80.5	When counting
		91.5 TO 184	When reset
00	NOP	2	Always
01	END	—	Refer to Cycle Time Calculation Example.
02	IL	2.5	Always
03	ILC	3	Always
04	JMP	94	Always
05	JME	38	Always
08	STEP	60 to 127	Always

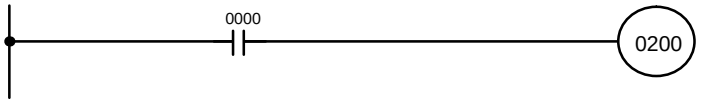
Function code	Instruction	Execution time(μs)	Conditions
09	SNXT	100	Always
10	SFT	102	When shifting 1 word
		248	When shifting 13 words
		90 to 254	When reset (1 to 13 words)
11	KEEP	19	When set
		20	When reset
12	CNTR	95	When counting DOWN
		190.5	When counting UP (word specified)
13	DIFU	60.5	When input = 1
		56.5	When input = 0
14	DIFD	59	When input = 1
		62.5	When input = 0
15	TIMH	94.5	When timing
		97 to 187.5	When reset
16	WSFT	97	When shifting DM by 1 word
		825.5	When shifting DM by 64 words
20	CMP	121.5	When comparing a constant with word data
		212	When comparing a TIM/CNT with word data
21	MOV	109	When transferring a constant to a word
		196	When transferring a TIM/CNT to a word
22	MVN	108.5	When inverting & transferring a constant to a word
		196	When inverting & transferring a TIM/CNT to a word
23	BIN	115	When converting & transferring a TIM/CNT to a word
		193.5	When converting & transferring a word to a word
24	BCD	194	When converting & transferring DM to DM
		202.5	When converting & transferring data in other areas
30	ADD	233	When adding two words
		352	When adding a TIM/CNT to a constant
31	SUB	237.5	When subtracting a word from a word
		356.5	When subtracting a constant from a TIM/CNT
32	MUL	655	When multiplying a DM word by a DM word
33	DIV	572	When dividing a DM word by a DM word
40	STC	16	Always
41	CLC	16	Always
60	RDM	695	At reset
61	HDM	734	Always
62	ENDW	197	With DM word
63	NETW	58	Always
76	MLPX	212.5	Word, 1 digit (constant) —> word
		288	Word, 4 digits (constant) —> word
		355	TIM/CNT, 1 digit (TIM/CNT) —> word
		431	TIM/CNT, 4 digits (TIM/CNT) —> word
77	DMPX	298.5	Word, 1 digit (constant) —> word
		658.5	Word, 4 digits (constant) —> word
		456	TIM/CNT, 1 digit (TIM/CNT) —> word
		1,080	TIM/CNT, 4 digits (TIM/CNT) —> word
		145	When shifting one word
		743	When shifting 64 DM words

Function code	Instruction	Execution time(μs)	Conditions
84	SFTR	136 to 668	When resetting 1 to 64 DM words
		44	NOP
		42	IL
91	SBS	75	Always
92	SBN	26	Always
93	RET	49	Always
97	IORF	108	When refreshing 1 word

6-5 I/O Response Time

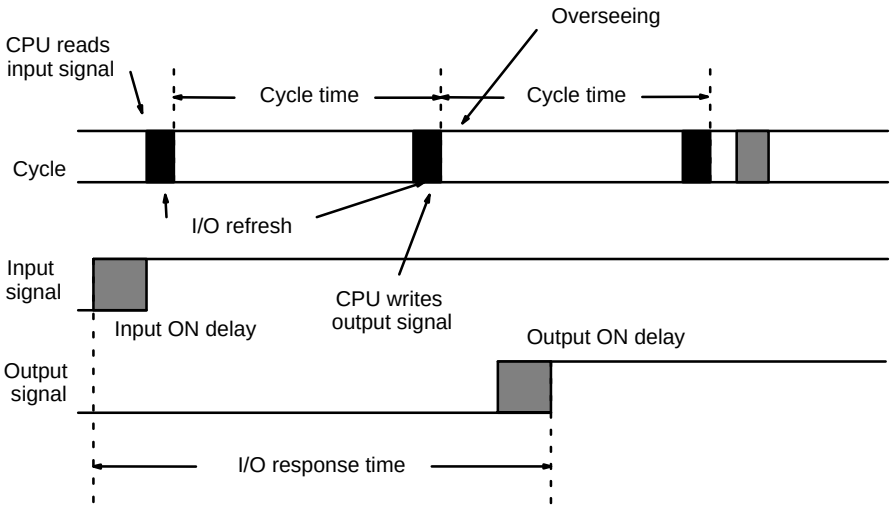
The I/O response time is the time it takes for the PC to output a control signal after it has received an input signal. How long it takes to respond depends on the cycle time and when the CPU receives the input signal relative to the input refresh period. The I/O response times for a PC not in a Link System are discussed below. For response times for PCs with Link Systems, refer to the relevant *System Manual*.

The minimum and maximum I/O response time calculations described below are for the following, where 0000 is the input bit that receives the signal and 0200 is the output bit corresponding to the desired output point.



Minimum I/O Response Time

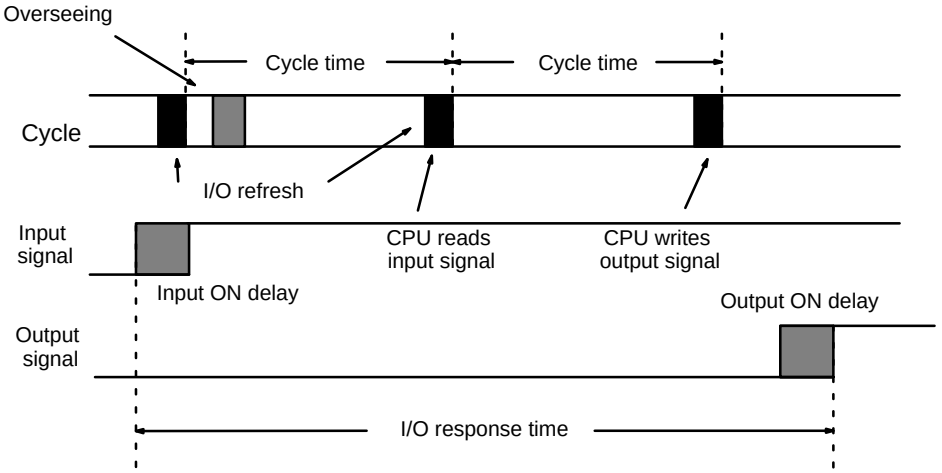
The PC responds most quickly when it receives an input signal just prior to the input refresh period in the cycle. Once the input bit corresponding to the signal has been turned ON, the program will have to be executed once to turn ON the output bit for the desired output signal and then the input refresh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



Minimum I/O response time = Input ON delay + Cycle time + I/O refresh time + Overseeing time + Output ON delay

Maximum I/O Response Time

The PC takes longest to respond when it receives the input signal just after the input refresh phase of the cycle. In this case the CPU does not recognize the input signal until the end of the next cycle. The maximum response time is thus one cycle longer than the minimum I/O response time, except that the input refresh time would not need to be added in because the input comes just after it rather than before it.



Maximum I/O response time = input ON delay + (cycle time x 2) + overseeing time + output ON delay

Calculation Example

The data in the following table would produce the minimum and maximum cycle times shown calculated below.

Input ON-delay	1.5 ms
Cycle time	20 ms
Input refresh time	0.23 ms
Overseeing time	3.0 ms
Output ON-delay	15 ms

Minimum I/O response time = 1.5 + 20 + 0.23 + 3.0 +15 = 39.73 ms

Maximum I/O response time = 1.5 + (20 x 2) + 3.0 +15 = 59.5 ms

SECTION 7
Program Debugging and Execution

7-1 Introduction . . . . .
7-2 Debugging . . . . .
7-3 Monitoring Operation and Modifying Data . . . . .
7-3-1 Bit/Digit Monitor . . . . .
7-3-2 Force Set/Reset . . . . .
7-3-3 Hexadecimal/BCD Data Modification . . . . .
7-3-4 Changing Timer/Counter SV . . . . .
7-4 Program Backup and Restore Operations . . . . .
7-4-1 Saving Program Memory Data . . . . .
7-4-2 Restoring or Comparing Program Memory Data . . . . .

7-1 Introduction

This section provides the procedures for inputting and debugging a program and monitoring and controlling the PC through a Programming Console. The Programming Console is the most commonly used Programming Device for the K-type PCs. It is compact and available both in hand-held models or CPU-mounted models. Refer to *Appendix A Standard Models* for model numbers and other details.

If you are using a GPC, FIT, or a computer running LSS, refer to the *Operation Manual* for corresponding procedures on these. If you are going to use a GPC, FIT, or a computer running LSS, but want to input in mnemonic code rather than in ladder diagram form, refer to *4-3-2 Mnemonic Code*.

7-2 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 cycle time, errors in settings for various Units in the PC, and inappropriate control actions, i.e., the program not doing what it is designed to do.

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

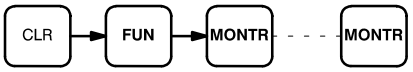
Displaying and Clearing Error Messages

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

If a beeper sounds and the error cannot be cleared by pressing MONTR, the cause of the error still exists and must be eliminated before the error message can be cleared. If this happens, take the appropriate corrective action to eliminate the error. Refer to *Section 8 Troubleshooting* for all details on all error messages. The sequence in which error messages are displayed depends on the priority levels of the errors. The messages for fatal errors (i.e., those that stop PC operation) are displayed before non-fatal ones.

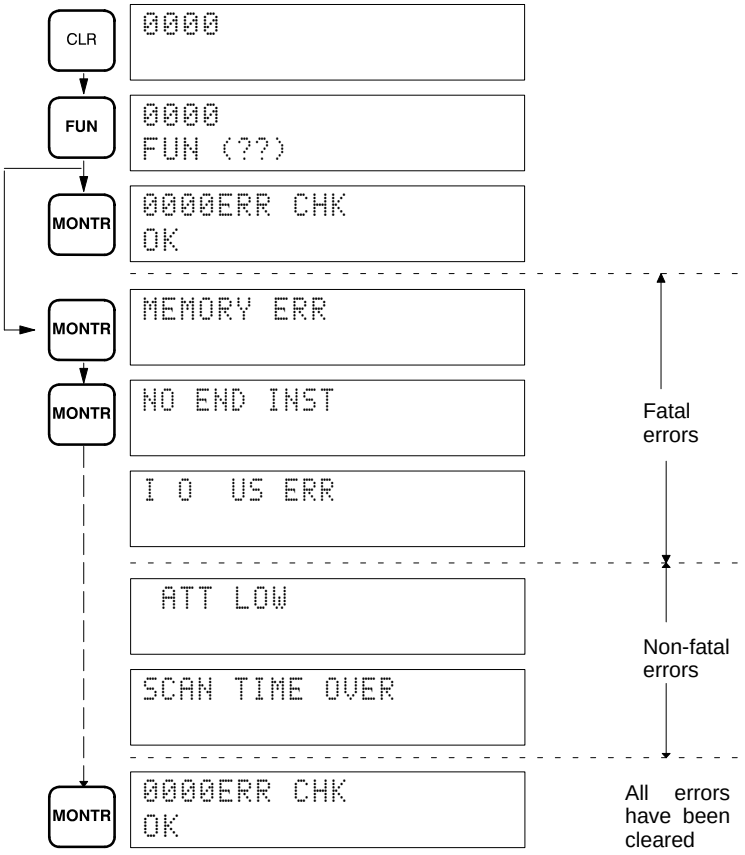
Although error messages can be displayed in any mode, they can be cleared only in PROGRAM mode. There is no way to restart the PC following a fatal error without first clearing the error message in PROGRAM mode.

Key Sequence



Example The following displays show some of the messages that may appear. Refer to *Section 8 Troubleshooting* for an inclusive list of error messages, meanings, and appropriate responses.

Note Cycle time is displayed as scan time.



7-3 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 one of the search operations. As long as the operation is performed in RUN or MONITOR 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) and SV (set value) for any timer or counter.

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

All data modification operations except for timer/counter SV changes are performed after first performing one of the monitor operations. Data modification is possible in either MONITOR or PROGRAM mode, but cannot be performed in RUN mode.

7-3-1 Bit/Digit 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 only in MONITOR or RUN mode.

The Bit/Digit 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 MONTR.

When a bit is monitored, it's ON/OFF status will be displayed (in MONITOR or 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 timer or counter's completion flag is ON. The status of TR bits and SR flags cleared when END(01) is executed (e.g., the arithmetic flags) cannot be monitored.

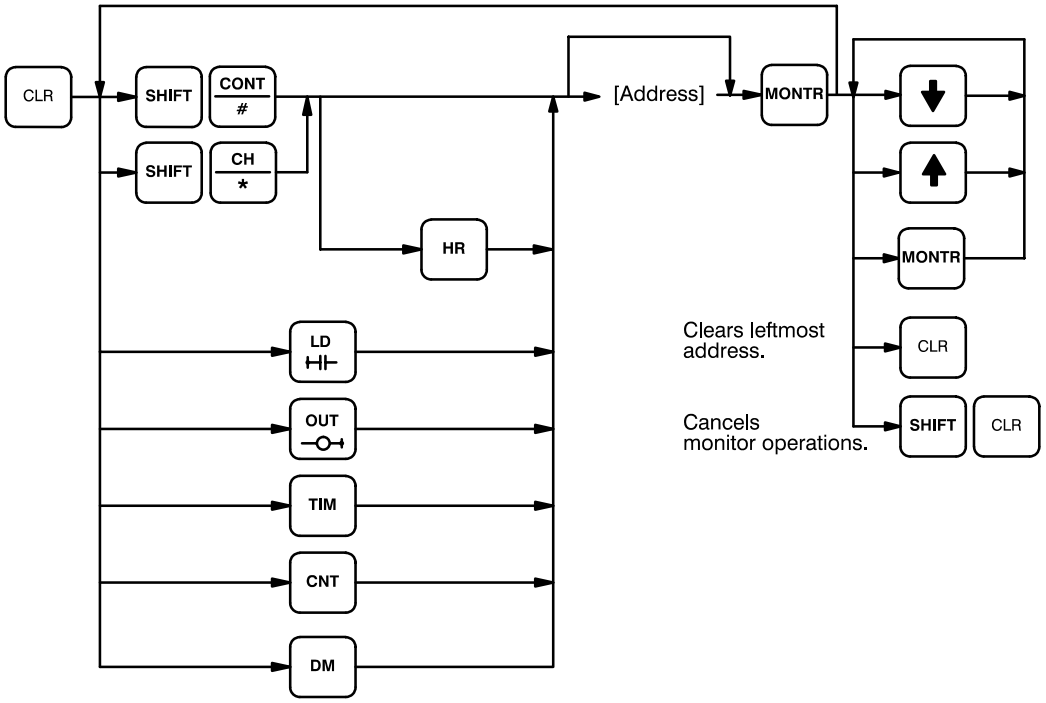
Up to six memory addresses, either bits, words, or a combination of both, can be monitored at once, although only three of these are displayed at any one time. To monitor more than one address, return to the start of the procedure and continue designating addresses. Monitoring of all designated addresses will be maintained unless more than six addresses are designated. If more than six addresses are designated, the leftmost address of those being monitored will be cancelled.

To display addresses that are being monitored but are not presently on the Programming Console display, press MONTR without designating another address. The addresses being monitored will be shifted to the right. As MONTR is pressed, the addresses being monitored will continue shifting to the right until the rightmost address is shifted back onto the display from the left.

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.

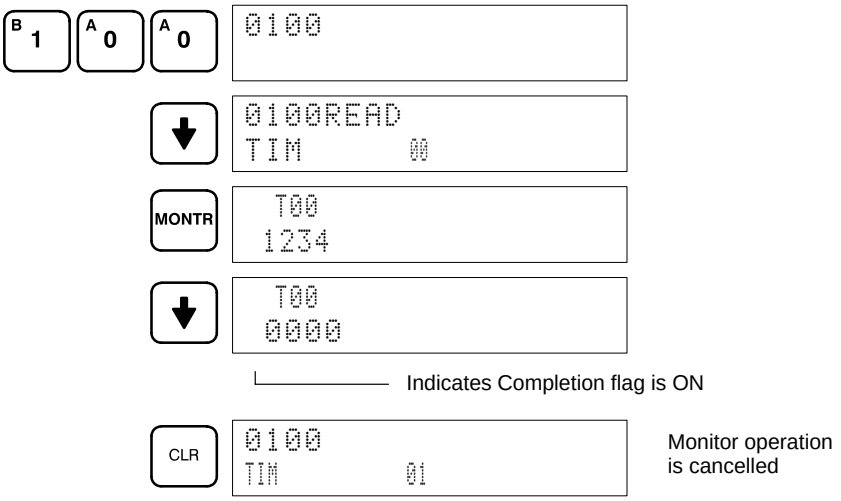
LD and OUT can be used only to designate the first address to be displayed; they cannot be used when an address is already being monitored.

Key Sequence

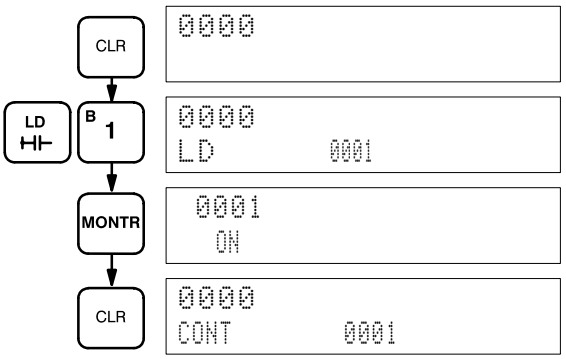


Examples The following examples show various applications of this monitor operation.

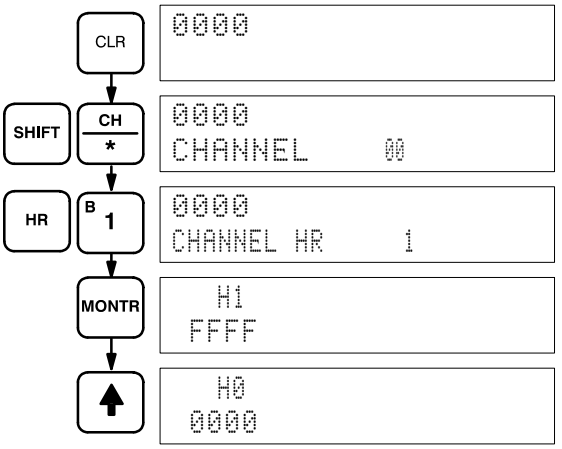
Program Read then Monitor



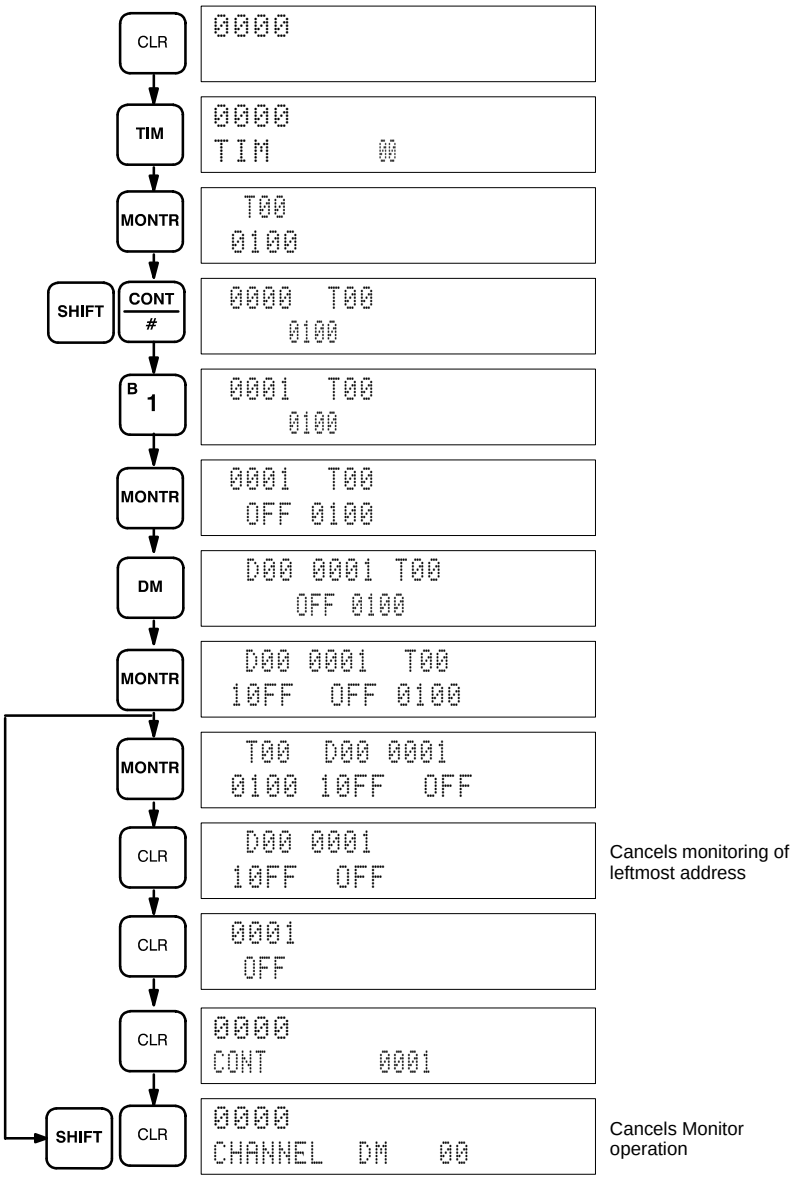
Bit Monitor



Word Monitor



Multiple Address Monitoring



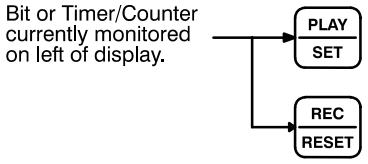
7-3-2 Force Set/Reset

When the Bit/Digit Monitor operation is being performed and a bit, timer, or counter address is leftmost on the display, PLAY/SET can be pressed to turn ON the bit, start the timer, or increment the counter and REC/RESET can be pressed to turn OFF the bit or reset the timer or counter. Timers will not operate in PROGRAM mode. SR bits cannot be turned ON and OFF with this operation.

Bit status will remain ON or OFF for only one scan after pressing the key; it will then return to its original status. When timers or counters are reset in MONITOR mode, they will start after one scan.

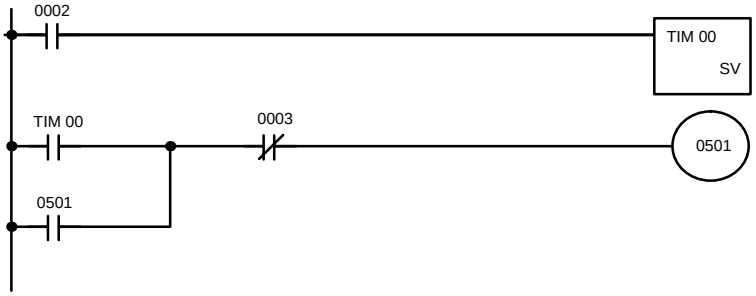
This operation can be used in MONITOR mode to check wiring of outputs from the PC prior to actual program execution. This operation cannot be used in RUN mode.

Key Sequence



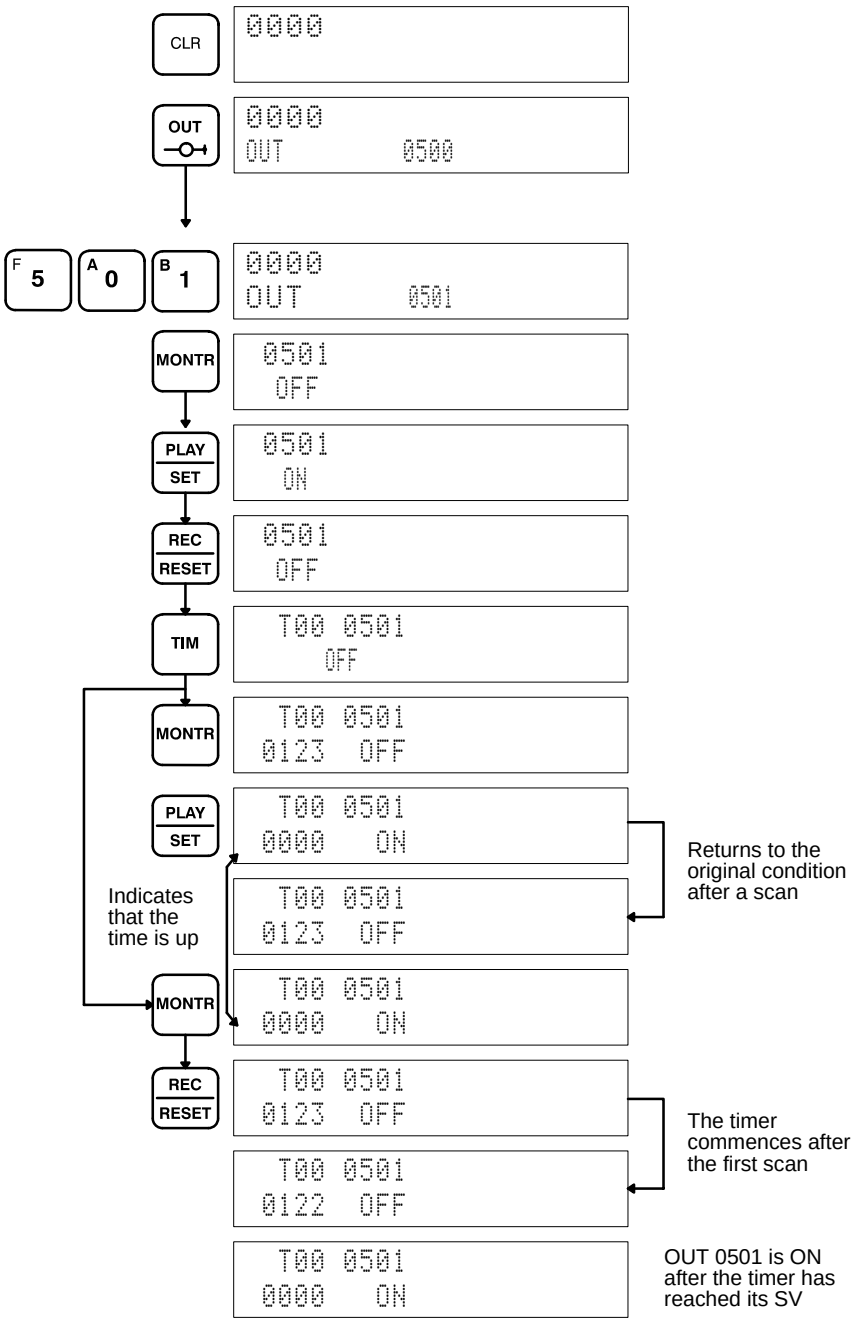
Example

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



Address	Instruction	Operands
0200	LD	0002
0201	TIM	00
		# 0123
0202	LD	TIM 00
0203	OR	0501
0204	AND NOT	0003
0205	OUT	0501

The following displays show what happens when TIM 00 is set with 0100 OFF (i.e., 0500 is turned ON) and what happens when TIM 00 is reset with 0100 ON (i.e., timer starts operation, turning OFF 0500, which is turned back ON when the timer has finished counting down the SV).



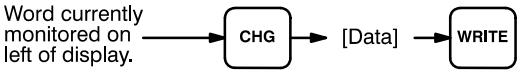
7-3-3 Hexadecimal/BCD Data Modification

When the Bit/Digit Monitor operation is being performed and a BCD or hexadecimal value is leftmost on the display, CHG can be input to change the value. SR 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. See *7-3-4 Changing Timer/Counter SV* for the procedure to change SV. PV can be changed in MONITOR mode and only when the timer or counter is operating.

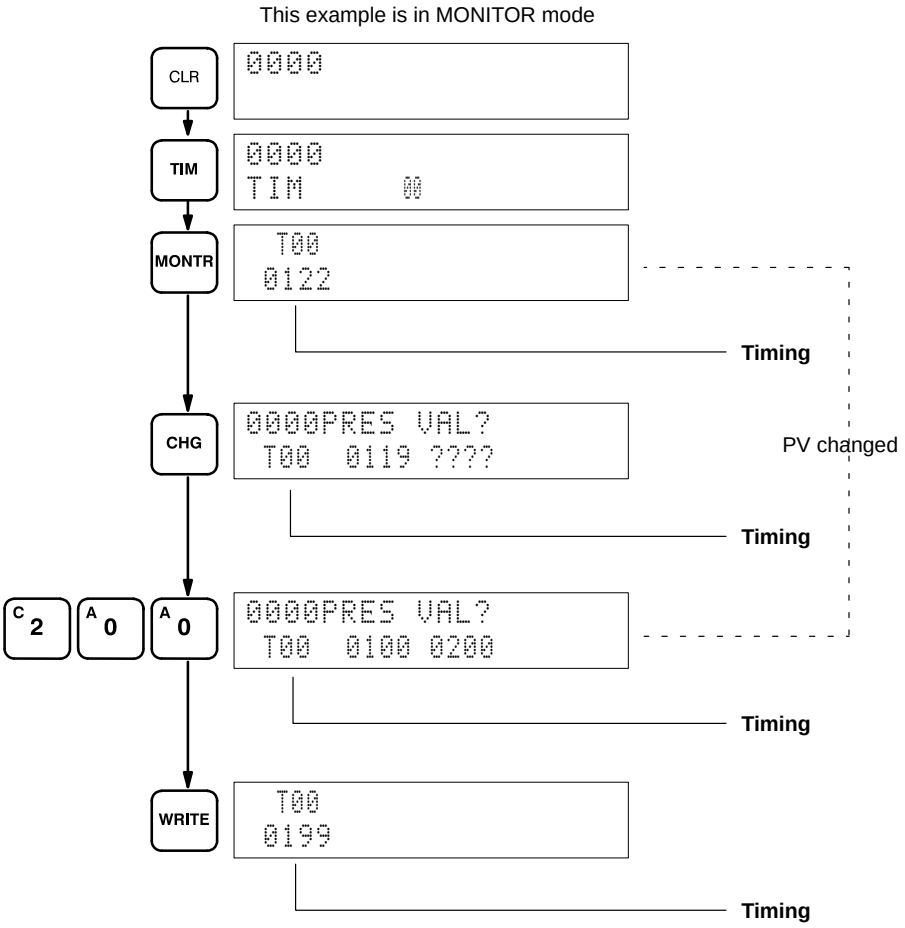
To change contents of the leftmost word address, press CHG, input the desired value, and press WRITE.

Key Sequence



Example

The following example shows the effects of changing the PV of a timer.



7-3-4 Changing Timer/Counter SV

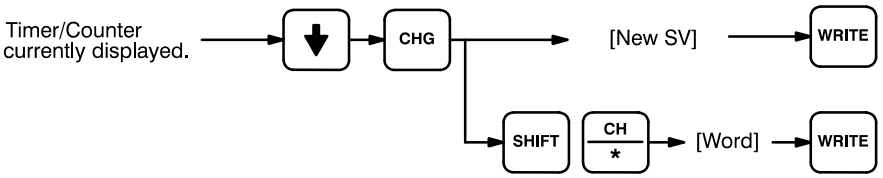
The SV of a timer or counter can be changed by inputting a new value numerically when in MONITOR mode. The SV can be changed while the program is being executed.

To change the SV, first display the address of the timer or counter whose SV is to be changed, press the down key, and then press CHG. The new value can then be input numerically and WRITE pressed to change the SV.

When changing the SV of timers or counters while operation is stopped, use PROGRAM mode and follow the procedure outlined in *4-6-2 Inputting or Overwriting Programs*.

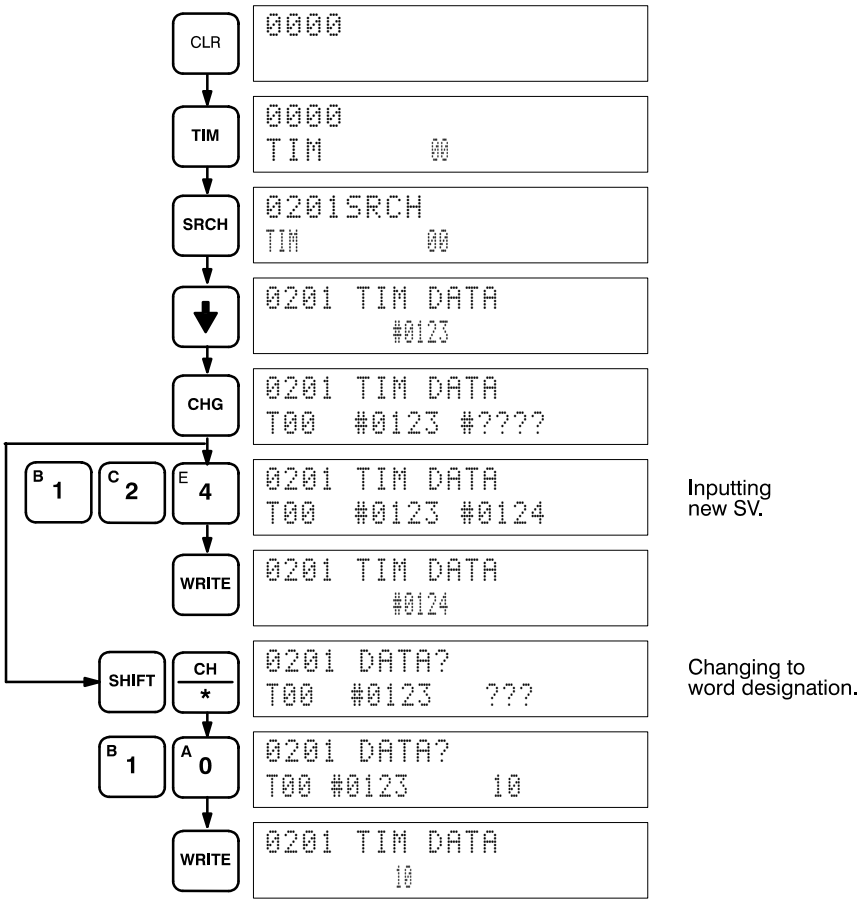
This operation can be used to change a SV from designation as a constant to a word address designation or from a word address to a constant designation.

Key Sequence



Example The following example shows inputting a new constant and changing from a constant to a word designation.

Inputting New SV



7-4 Program Backup and Restore Operations

Program Memory (UM) can be backed-up on a standard commercially available cassette tape recorder. Any kind of dependable magnetic tape of adequate length will suffice. To save a 16K-word program, the tape must be 30 minutes long. Always allow about 5 seconds of blank tape leader before the taped data begins. Store only one program on a single side of a tape; there is no way to identify separate programs stored on the same side of the tape. If a program is longer than will fit on one side, it can be split onto two sides.

Be sure to label the contents of all cassette tapes clearly.

Use patch cords to connect the cassette recorder earphone (or LINE-OUT) jack to the Programming Console EAR jack and the cassette recorder microphone (or LINE-IN) jack to the Programming Console MIC jack. Set the cassette recorder volume and tone controls to maximum levels.

The PC must be in PROGRAM mode for all cassette tape operations.

While the operation is in progress, the cursor will blink and the block count will be incremented on the display.

Cassette tape operations may be halted at any time by pressing the CLR key.

Error Messages

The following error messages may appear during cassette tape operations.

Message	Meaning and appropriate response
0000 ERR ******* FILE NO.********	File number on cassette and designated file number are not the same. Repeat the operation using the correct file number.
**** MT VER ERR	Cassette tape contents differs from that in the PC. Check content of tape and/or the PC.
**** MT ERR	Cassette tape is faulty. Replace it with another.

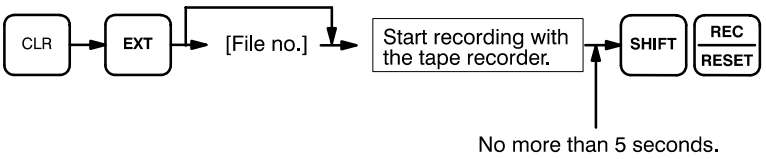
7-4-1 Saving Program Memory Data

This operation is used to copy the content of Program Memory to a cassette tape. The procedure is as follows:

1, 2, 3... Press EXT.
Input a file number for the data that is to be saved.
Start cassette tape recording.
Within 5 seconds, press the SHIFT and REC/RESET keys.

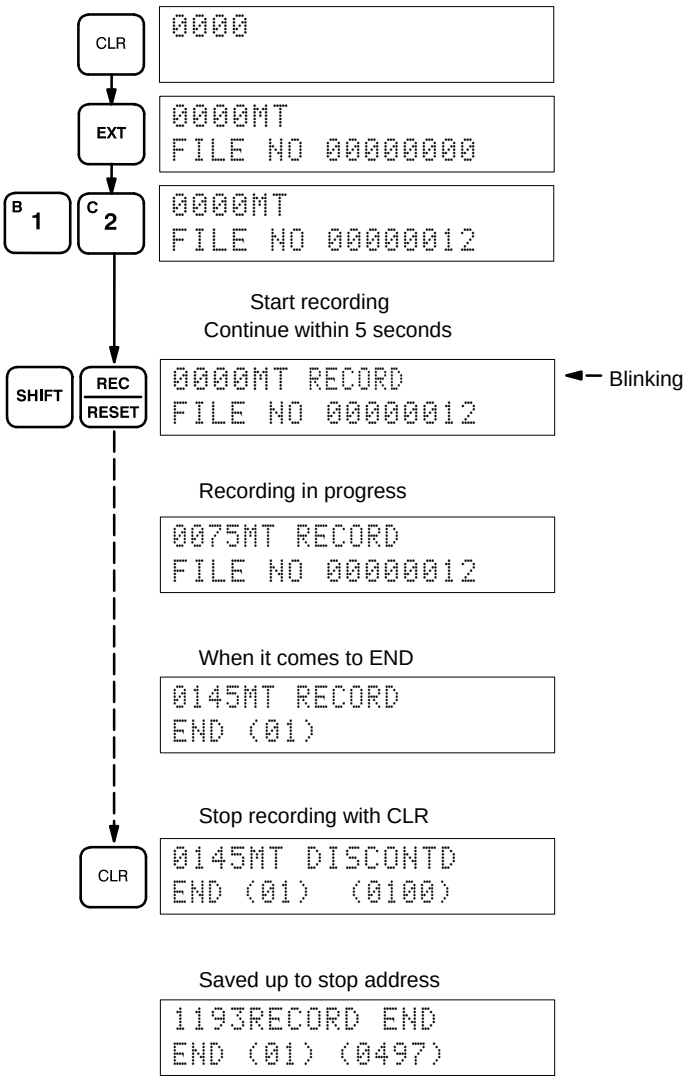
Program saving continues until END(01) or the final address is reached. Cancel by pressing the CLR key.

Key Sequence



(Cancel with the CLR key.)

Example



7-4-2 Restoring or Comparing Program Memory Data

This operation is used to restore Program Memory data from a cassette tape or to compare Program Memory data with the contents on a cassette tape. The procedure is as follows:

1, 2, 3... Press EXT.

Specify the number of the file to be restored or compared.

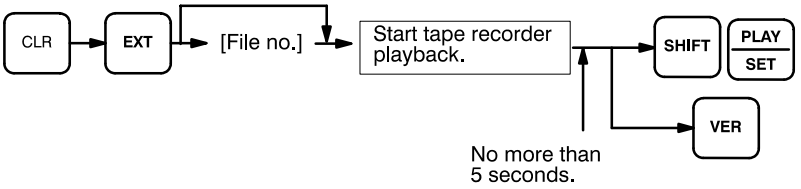
Start playing the cassette tape.

Within 5 seconds, press SHIFT and PLAY/SET to restore data or VER to compare data.

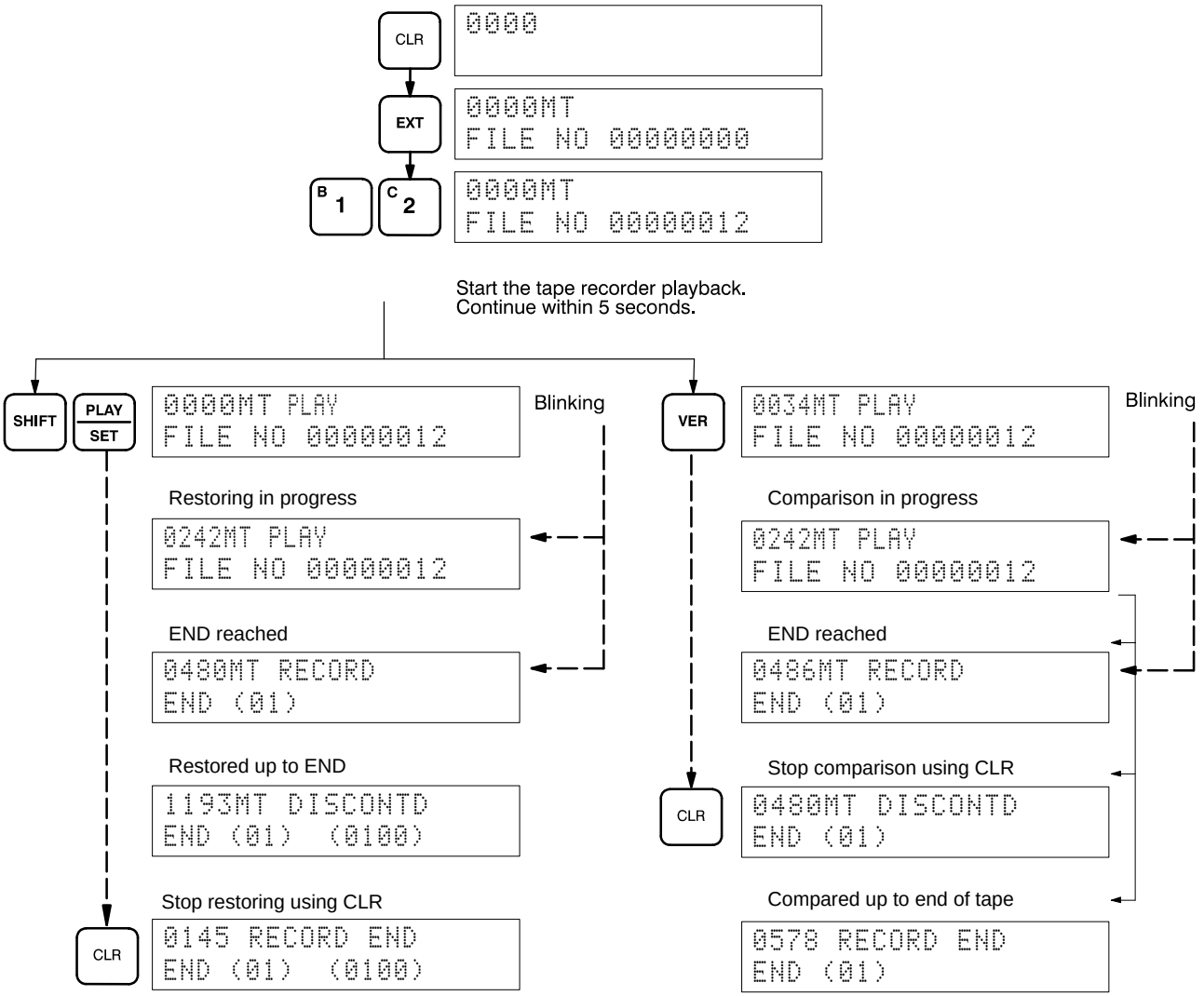
Program restoration or comparison continues until the final address or END(01) is reached or until the tape is finished. Cancel by pressing the CLR key.

To restore or compare program data recorded on two sides of a tape or on two or more tapes, begin restoring or comparing from the lowest address.

Key Sequence



Example



SECTION 8
Troubleshooting

8-1 Introduction . . . . . .
8-2 Reading and Clearing Errors and Messages . . . . . .
8-3 Error Messages . . . . . .
8-4 Error Flags . . . . . .

8-1 Introduction

The K-type PCs provide self-diagnostic functions to identify many types of abnormal system conditions. These functions minimize downtime and enable quick, smooth error correction.

This section provides information on hardware and software errors that occur during PC operation. Program input and program syntax errors are described in *Section 4 Writing and Inputting the Program*. Although described in *Section 3 Memory Areas*, flags and other error information provided in SR areas are listed in *8-4 Error Flags*.

There are two indicators on the front of the CPU that provide visual indication of an abnormality in the PC. The error indicator (ERR) indicates fatal errors (i.e., ones that will stop PC operation); the alarm indicator (ALARM) indicates nonfatal ones. These indicators are shown in *2-2 Indicators*.

⚠ DANGER The PC will turn ON the error indicator (ERR), stop program execution, and turn OFF all outputs from the PC for most hardware errors, or certain 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 and other system and/or user-programmed error indications can be used to program proper actions.

8-2 Reading and Clearing Errors and Messages

System error messages can be displayed on the Programming Console or any other Programming Device.

On the Programming Console, press the CLR, FUN, and MONTR keys. If there are multiple error messages stored by the system, the MONTR key can be pressed again to access the next message. If the system is in PROGRAM mode, pressing the MONTR key will clear the error message, so be sure to write down all message errors as you read them out. (It is not possible to clear an error or a message while in RUN or MONITOR mode; the PC must be in PROGRAM mode.) When all messages have been cleared, "ERR CHK OK" will be displayed.

Details on accessing error messages from the Programming Console are provided in *7-3 Monitoring Operation and Modifying Data*. Procedures for the GPC, LSS, and FIT are provided in the relevant *Operation Manual*.

8-3 Error Messages

There are basically two types of errors for which messages are displayed: non-fatal operating 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, 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.

Non-fatal Operating Errors

The following error messages appear for errors that occur after program execution has been started. PC operation and program execution will continue after one or more of these error have occurred. The POWER, RUN, and ALARM indicators will be lit and the ERR indicator will not be lit for any of these errors. The RUN output will be ON.

Error and message	Probable cause	Possible correction
Cycle time overrun SCAN TIME OVER	Watchdog timer has exceeded 100 ms.	Program cycle time is longer then desirable. Reduce cycle time if possible.
Battery error ATT LOW	Backup battery is missing or it's voltage has dropped.	Check battery and replace if necessary.

Fatal Operating Errors

The following error messages appear for errors that occur after program execution has been started. PC operation and program execution will stop and all outputs from the PC will be turned OFF when any of the following errors occur. All CPU indicators will not be lit for the power interruption error. For all other fatal operating errors, the POWER, and ERR indicators will be lit and the RUN indicator will not be lit. The RUN output will be OFF.

Error and message	Probable cause	Possible correction
Power interruption No message	Power has been interrupted for at least 10 ms.	Check power supply voltage and power lines. Try to power-up again.
CPU error No message	Watchdog timer has exceeded maximum setting (default setting: 130 ms).	Restart system in PROGRAM mode and check program. Reduce cycle time or reset watchdog timer if longer time required. (Consider effects of longer cycle time before resetting).
Memory error MEMORY ERR	Memory Unit is incorrectly mounted or missing or parity error has occurred.	Check Memory Unit to make sure it is mounted and backed up properly. Perform a Program Check Operation to locate cause of error. If error not correctable, try inputting program again.
No END(01) instruction NO END INST	END(01) is not written anywhere in program.	Write END(01) at the final address of the program.
I/O bus error I O US ERR	Error has occurred in the bus line between the CPU and Units.	Check the CPU Left/Right selector switch on the Expansion I/O Unit. Check cable connections between the Units and Racks.

8-4 Error Flags

The following table lists the flags and other information provided in the SR area that can be used in troubleshooting. Details are provided in *3-4 Special Relay (SR) Area*.

SR Area

Address	Function
1808	Battery Alarm Flag
1809	Cycle Time Error Flag
1903	Instruction Execution Error (ER) Flag

Other Error Messages

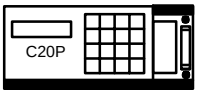
A number of other error messages are detailed within this manual. Errors in program input and debugging can be examined in *4-6-2 Inputting and Overwriting Programs* and *4-6-3 Checking the Program* and errors in cassette tape operation are detailed in *7-4 Program Backup and Restore Operations*.

Appendix A
Standard Models

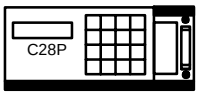
There are four K-type C-series CPUs. A CPU can be combined with any of six types of Expansion I/O Unit and/or an Analog Timer, Analog I/O Unit, or I/O Link Unit.

CPUs

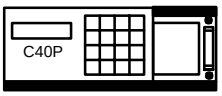
C20K-C□□-□



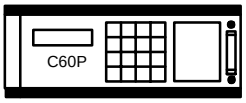
C28K-C□□-□



C40K-C□□-□



C60K-C□□-□



To order cable separately, specify C4K-CN502

5 cm or 40 cm
One included with each Expansion I/O Unit.

Expansion I/O Units

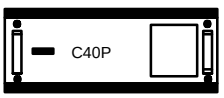
C4K-I□/O□□

C16P-I□-□/O□-□

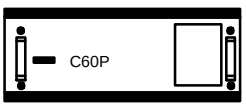
C20P-E□□-□

C28P-E□□-□

C40P-E□□-□



C60P-E□□-□



C4K-CN502
(included with Unit)

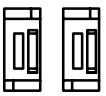
Cable (70 cm)
C20P-CN711
(ordered separately)

Analog Timer Unit

C4K-TM

Analog I/O Units

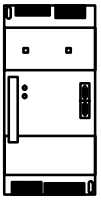
C1K-AD/DA



C4K-AD

I/O Link Unit

C20-LK011/LK011-P



CPUs

Name	Power supply	Inputs	Outputs		Model number	Standards
C20K	100 to 240 VAC	24 VDC, 12 pts	Relay w/socket	8 pts	C20K-CDR-A	U, C
			Transistor, 1 A		C20K-CDT1-A	U, C
			Triac, 1 A		C20K-CDS1-A	U, C
		24 VDC, 2 pts	Relay w/socket		C20K-CAR-A	U, C, N, L
		100 to 120 VAC, 10 pts	Triac, 1A		C20K-CAS1-A	U, C
	24 VDC	24 VDC, 12 pts	Relay w/socket		C20K-CDR-D	U, C
			Transistor, 1 A		C20K-CDT1-D	U
C28K	100 to 240 VAC	24 VDC, 16 pts	Relay w/socket	12 pts	C28K-CDR-A	U, C
			Transistor, 1 A		C28K-CDT1-A	U, C
			Triac, 1 A		C28K-CDS1-A	U, C
		24 VDC, 2 pts	Relay w/socket		C28K-CAR-A	U, C
		100 to 120 VAC, 14 pts	Triac, 1A		C28K-CAS1-A	U, C
	24 VDC	24 VDC, 16 pts	Relay w/socket		C40K-CDR-D	U, C
			Transistor, 1 A		C28K-CDT1-D	U, C
C40K	100 to 240 VAC	24 VDC, 16 pts	Relay w/socket	16 pts	C40K-CDR-A	U, C
			Transistor, 1 A		C40K-CDT1-A	U, C
			Triac, 1 A		C40K-CDS1-A	U, C
		24 VDC, 2 pts	Relay w/socket		C40K-CAR-A	U, C
		100 VAC, 22 pts	Triac, 1 A		C40K-CAS1-A	U, C
	24 VDC	24 VDC, 24 pts	Relay w/socket		C40K-CDR-D	U, C
			Transistor, 1 A		C40K-CDT1-D	U, C
C60K	100 to 240 VAC	24 VDC, 32 pts	Relay w/socket	28 pts	C60K-CDR-A	U, C
			Transistor, 1 A		C60K-CDT1-A	U, C
			Triac, 1 A		C60K-CDS1-A	U, C
		24 VDC, 2 pts	Relay w/socket		C60K-CAR-A	U, C
		100 VAC, 30 pts	Triac, 1 A		C60K-CAS1-A	U, C
	24 VDC	24 VDC, 32 pts	Relay w/socket		C60K-CDR-D	U, C
			Transistor, 1 A		C60K-CDT1-D	---

● U: UL, C: CSA, N: NK, L: LLOYD
See Omron sales representatives concerning operating conditions under which UL, CSA, and NK standards were met (Aug. 1991).

I/O Units

Name	Power Supply	Inputs	Outputs		Model number	Standards
C4K I/O Unit	---	24 VDC, 4 pts	---		C4K-ID	U, C
		100 to 120 VAC, 4 pts	---		C4K-IA	U, C
		---	Relay w/socket	4 pts	C4K-OR2	U, C
			Transistor, 1 A		C4K-OT2	U, C
			Triac, 1A		C4K-OS2	U, C
C16P I/O Unit	100 to 240 VAC	24 VDC, 16 pts	---		C16P-ID-A	U, C
		---	Relay w/socket	16 pts	C16P-OR-A	U, C
			Transistor, 1 A		C16P-OT1-A	U, C
			Triac, 1A		C16P-OS1-A	U, C
	---	24 VDC, 16 pts	---		C16P-ID	U, C
		100 to 120 VAC, 16 pts	---		C16P-IA	U, C
	24 VDC	---	Relay w/socket	16 pts	C16P-OR-D	U, C
			Transistor, 1 A		C16P-OT1-D	U
C20P I/O Unit	100 to 240 VAC	24 VDC, 12 pts	Relay w/socket	8 pts	C20P-EDR-A	U, C, N, L
			Transistor, 1 A		C20P-EDT1-A	U, C, N, L
			Triac, 1A		C20P-EDS1-A	U, C, N, L
		100 to 120 VAC, 12 pts	Relay w/socket		C20P-EAR-A	U, C, N, L
			Triac, 1A		C20P-EAS1-A	U, C, N, L
	24 VDC	24 VDC, 12 pts	Relay w/socket		C20P-EDR-D	U, C, N, L
			Transistor, 1 A		C20P-EDT1-D	U, C, N, L
C28P I/O Unit	100 to 240 VAC	24 VDC, 16 pts	Relay w/socket	12 pts	C28P-EDR-A	U, C, N, L
			Transistor, 1 A		C28P-EDT1-A	U, C, N, L
			Triac, 1A		C28P-EDS1-A	U, C, N, L
		100 to 120 VAC, 16 pts	Relay w/socket		C28P-EAR-A	U, C, N, L
			Triac, 1A		C28P-EAS1-A	U, C, N, L
	24 VDC	24 VDC, 16 pts	Relay w/socket		C28P-EDR-D	U, C, N, L
			Transistor, 1 A		C28P-EDT1-D	U, C, N, L
C40P I/O Unit	100 to 240 VAC	24 VDC, 24 pts	Relay w/socket	16 pts	C40P-EDR-A	U, C, N, L
			Transistor, 1 A		C40P-EDT1-A	U, C, N, L
			Triac, 1A		C40P-EDS1-A	U, C, N, L
		100 to 120 VAC, 24 pts	Relay w/socket		C40P-EAR-A	U, C, N, L
			Triac, 1A		C40P-EAS1-A	U, C, N, L
	24 VDC	24 VDC, 24 pts	Relay w/socket		C40P-EDR-D	U, C, N, L
			Transistor, 1 A		C40P-EDT1-D	U, C, N, L
C60P I/O Unit	100 to 240 VAC	24 VDC, 32 pts	Relay w/socket	28 pts	C60P-EDR-A	U, C
			Transistor, 1 A		C60P-EDT1-A	---
			Triac, 1A		C60P-EDS1-A	U, C
		100 VAC, 32 pts	Relay w/socket		C60P-EAR-A	U, C
			Triac, 1A		C60P-EAS1-A	U, C
	24 VDC	24 VDC, 32 pts	Relay w/socket		C60P-EDR-D	U, C
			Transistor, 1 A		C60P-EDT1-D	---

Special Units

Name	Specifications		Model number	Standards
Analog Timer Unit	Settings: 0.1 s to 10 min (one cable, C4K-CN502, included)		C4K-TM	U, C
Analog Timer External Connector	2-m cable and connector		C4K-CN223	---
Analog Input Unit	1 input; input ranges: 4 to 20 mA, 1 to 5 V		C1K-AD	U, C
	4 inputs; input ranges: 4 to 20 mA, 1 to 5 V		C4K-AD	U, C
Analog Output Unit	1 output; output ranges: 4 to 20 mA, 1 to 5 V		C1K-DA	U, C
Host Link Unit	RS-232C	C20/C20K/C28K/C40K/C60K	3G2C7-LK201-EV1	---
	RS-422		3G2C7-LK202-EV1	---
I/O Link Unit	APF/PCF		C20-LK011-P	U, C
	PCF		C20-LK011	U, C
I/O Connecting Cable	For horizontal mounting; cable length: 5 cm (for maintenance)		C20P-CN501	---
	For vertical mounting; cable length: 40 cm (for maintenance)		C20P-CN411	---
I/O Connecting Cable	For horizontal mounting; connects to C4K I/O, Analog Timer, or Analog I/O Units (for maintenance)	Cable length: 5 cm	C4K-CN502	---
		Cable length: 50 cm	C4K-CN512	---
		Cable length: 1 m	C4K-CN122	---
I/O Link Connecting Cable	Cable length: 70 cm; for I/O Link Units only		C20P-CN711	---
EPROM	2764		ROM-H	L
Battery Set	Built into CPU (same for all C-Series PCs)		3G2A9-BAT08	---
Relay	24-VDC contact relay		G6B-1174P-FD-US	U, C
	24-VDC transistor relay		G3SD-Z01P-PD-US	U, C
	24-VDC triac relay		G3S-201PL-PD-US	U, C

- U: UL, C: CSA, N: NK, L: LLOYD

See Omron sales representatives concerning operating conditions under which UL, CSA, and NK standards were met (Aug. 1991).

Mounting Rail and Accessories

Name	Specifications		Model number	Standards
DIN Track	Length: 50 cm	Not usable with C60K	PFP-50N	
	Length: 1 m		PFP-100N	---
			PFP-100N2	
End Plate	---		PFP-M	
Spacer	---		PFP-S	

Factory Intelligent Terminal (FIT)

Name	Specifications	Model number	Standards
FIT	1. FIT Computer 2. SYSMATE Ladder Pack (2 system disks, 1 data disk) 3. MS-DOS 4. GPC Communications Adapter (C500-IF001) 5. Peripheral Connecting Cable (3G2A2-CN221) 6. Power Cord and 3-pin/2-pin plug 7. Carrying Case	FIT10-SET11-E	---

Graphic Programming Console (GPC)

Name	Specifications		Model number	Standards
GPC (LCD display)	W/battery; power supply: 32 kw, 100 VAC; w/comments; System Memory Cassette ordered separately.		3G2C5-GPC03-E	
	W/battery; power supply: 32 kw, 200 VAC; w/comments; System Memory Cassette ordered separately.		3G2C5-GPC04-E	
GPC Carrying Case	W/side pocket for accessories		C500-CS001	
GPC System Memory Cassette	For K-Type PCs	W/comments	3G2C5-MP304-EV3	---
Cassette Interface Unit	Used to load programs in V8, M1R, M5R, POR, or S6 cassettes into the GPC and print them out through a Printer Interface Unit.		3G2A5-CMT01-E	

Peripheral Devices

Name	Specifications		Model number	Standards
Programming Console	Vertical, with backlight		3G2A5-PRO13-E	U, C
	Horizontal, with backlight		3G2A6-PRO15-E	---
	Hand-Held, with backlight. The Programming Console Adapter AP003 and connecting cable CN222/CN422 are necessary. They are sold separately.		C200H-PR027-E	U, C
Programming Console Mounting Bracket	Used to attach Hand-held Programming Console to a panel.		C200H-ATT01	---
Programming Console Connecting Cables	For C20K/C28K/C40K/C60K	1 m	3G2C7-CN122	---
		50 cm	3G2C7-CN512	---
	For Hand-held Programming Console	2 m	C200H-CN222	U, C
		4 m	C200H-CN422	U, C
Programming Console Adapter	Attached to PC when connecting Programming Console via cable (for 3G2A5-PRO13-E or 3G2A6-PRO15-E).		3G2A5-AP001-E	---
	Required to use Hand-held Programming Console.		3G2A5-AP003	---
Programming Console Base	Attached to Programming Console when connecting Programming Console via cable.		3G2A5-BP001	---
Cassette Recorder Connecting Cable	Used to connect Programming Console, GPC, or Cassette Deck Interface Unit to a cassette deck; length: 1 m.		SCYPOR-PLG01	---
PROM Writer	Used for all K-type PCs.		C500-PRW06	---
Printer Interface Unit	Interface for X-Y plotter or printer; System Memory Cassette ordered separately.		3G2A5-PRT01-E	---
Memory Rack	K-type PCs w/comment printing function		C500-MP102-EV3	
	K-type PCs		C20-MP009-EV3	
Printer Connecting Cable	2 m (also used for X-Y plotter)		SCY-CN201	---
Floppy Disk Interface Unit	C20K/C28K/C40K/C60K. GPC required; with comment file; able to connect to NEC floppy disk controller		3G2C5-FDI03-E	---
Peripheral Interface Unit	To connect GPC or FIT to K-type PCs		3G2C7-IP002-V2	
Connecting Cable	Used to connect FIT or GPC to Peripheral Interface Unit and to connect Programming Console Adapter and Programming Console Base.	2 m	3G2A2-CN221	---
		5 m	3G2A5-CN523	
		10 m	3G2A5-CN131	
		20 m	3G2A5-CN231	
		30 m	3G2A5-CN331	
		40 m	3G2A5-CN431	
		50 m	3G2A5-CN531	

● U: UL, C: CSA, N: NK
See Omron sales representatives concerning operating conditions under which UL, CSA, and NK standards were met (Aug. 1988).

Appendix B
Programming Instructions and Execution Times

Function code	Name	Mnemonic	Page
-	LOAD	LD	73
-	LOAD NOT	LD NOT	73
-	AND	AND	73
-	AND NOT	AND NOT	73
-	OR	OR	73
-	OR NOT	OR NOT	73
-	AND LOAD	AND LD	74
-	OR LOAD	OR LD	74
-	OUTPUT	OUT	75
-	OUTPUT NOT	OUT NOT	75
-	TIMER	TIM	83
-	COUNTER	CNT	90
00	NO OPERATION	NOP	81
01	END	END	81
02	INTERLOCK	IL	78
03	INTERLOCK CLEAR	ILC	78
04	JUMP	JMP	80
05	JUMP END	JME	80
08	STEP DEFINE	STEP	128
09	STEP START	SNXT	128
10	SHIFT REGISTER	SFT	106
11	KEEP	KEEP	77
12	REVERSIBLE COUNTER	CNTR	93
13	DIFFERENTIATE UP	DIFU	75
14	DIFFERENTIATE DOWN	DIFD	75
15	HIGH-SPEED TIMER	TIMH	86
16	WORD SHIFT	WSFT	110
20	COMPARE	CMP	112
21	MOVE	MOV	111
22	MOVE NOT	MVN	112
23	BCD-TO-BINARY	BIN	115
24	BINARY-TO-BCD	BCD	115
30	BCD ADD	ADD	120
31	BCD SUBTRACT	SUB	122
32	BCD MULTIPLY	MUL	123
33	BCD DIVIDE	DIV	124
40	SET CARRY	STC	125
41	CLEAR CARRY	CLC	125
60	REVERSIBLE DRUM COUNTER	RDM	103

Function code	Name	Mnemonic	Page
61	HIGH-SPEED DRUM COUNTER	HDM	94
62	END WAIT	ENDW	135
63	NOTATION INSERT	NETW	136
76	4-TO-16 DECODER	MLPX	116
77	16-TO-4 ENCODER	DMPX	118
84	REVERSIBLE SHIFT REGISTER	SFTR	109
91	SUBROUTINE ENTER	SBS	126
92	SUBROUTINE DEFINE	SBN	126
93	RETURN	RET	126
97	I/O REFRESH	IORF	135

Instruction Execution Times

This following table lists the execution times for all instructions that are available for the K-types. The maximum and minimum execution times and the conditions which cause them are given where relevant.

Execution times for most instructions depend on whether they are executed with an ON or an OFF execution condition. The OFF execution time for an instruction can also vary depending on the circumstances, i.e., whether it is in an interlocked program section and the execution condition for IL is OFF, whether it is between JMP(04) 00 and JME(05) 00 and the execution condition for JMP(04) 00 is OFF, or whether it is reset by an OFF execution condition. “R,” “IL,” and “JMP” are used to indicate these three times.

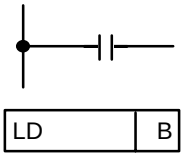
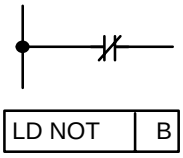
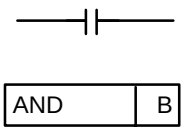
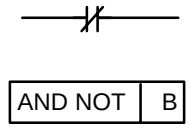
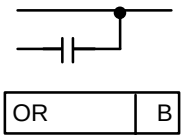
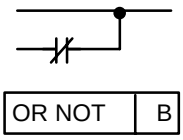
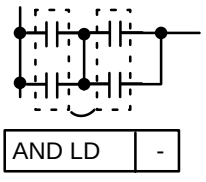
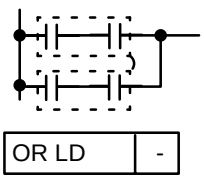
Execution times are expressed in microseconds except where noted.

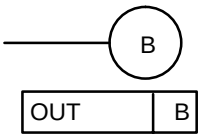
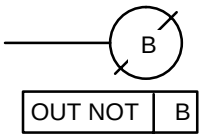
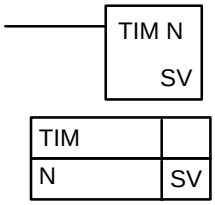
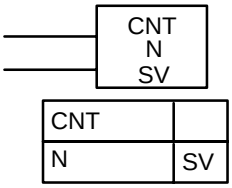
Function code	Instruction	Execution time(μs)	Conditions
---	LD	12	Always
	LD NOT	12	Always
	AND	11.5	Always
	AND NOT	11.5	Always
	OR	11.5	Always
	OR NOT	11.5	Always
	AND LD	4	Always
	OR LD	4	Always
	OUT	17	When outputting logical “1” (ON)
		17.5	When outputting logical “0” (OFF)
	OUT NOT	19	When outputting logical “1” (ON)
		17.5	When outputting logical “0” (OFF)

Function code	Instruction	Execution time(μs)	Conditions
---	TIM	95	When timing
		95.5 to 186.5	When reset
	CNT	80.5	When counting
		91.5 TO 184	When reset
00	NOP	2	Always
01	END	—	Refer to Cycle Time Calculation Example.
02	IL	2.5	Always
03	ILC	3	Always
04	JMP	94	Always
05	JME	38	Always
08	STEP	60 to 127	Always
09	SNXT	100	Always
10	SFT	102	When shifting 1 word
		248	When shifting 13 words
		90 to 254	When reset (1 to 13 words)
11	KEEP	19	When set
		20	When reset
12	CNTR	95	When counting DOWN
		190.5	When counting UP (word specified)
13	DIFU	60.5	When input = 1
		56.5	When input = 0
14	DIFD	59	When input = 1
		62.5	When input = 0
15	TIMH	94.5	When timing
		97 to 187.5	When reset
16	WSFT	97	When shifting DM by 1 word
		825.5	When shifting DM by 64 words
20	CMP	121.5	When comparing a constant with word data
		212	When comparing a TIM/CNT with word data
21	MOV	109	When transferring a constant to a word
		196	When transferring a TIM/CNT to a word
22	MVN	108.5	When inverting & transferring a constant to a word
		196	When inverting & transferring a TIM/CNT to a word
23	BIN	115	When converting & transferring a TIM/CNT to a word
		193.5	When converting & transferring a word to a word
24	BCD	194	When converting & transferring DM to DM
		202.5	When converting & transferring data in other areas
30	ADD	233	When adding two words
		352	When adding a TIM/CNT to a constant
31	SUB	237.5	When subtracting a word from a word
		356.5	When subtracting a constant from a TIM/CNT
32	MUL	655	When multiplying a DM word by a DM word

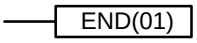
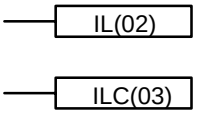
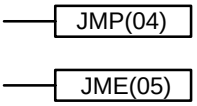
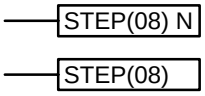
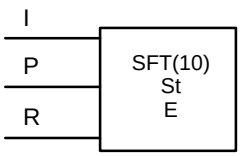
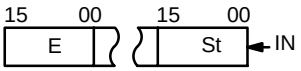
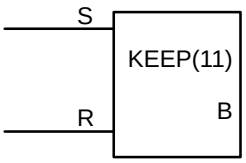
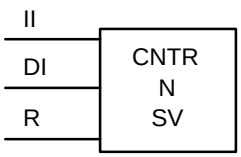
Function code	Instruction	Execution time(μs)	Conditions
33	DIV	572	When dividing a DM word by a DM word
40	STC	16	Always
41	CLC	16	Always
60	RDM	695	At reset
61	HDM	734	Always
62	ENDW	197	With DM word
63	NETW	58	Always
76	MLPX	212.5	Word, 1 digit (constant) —> word
		288	Word, 4 digits (constant) —> word
		355	TIM/CNT, 1 digit (TIM/CNT) —> word
		431	TIM/CNT, 4 digits (TIM/CNT) —> word
77	DMPX	298.5	Word, 1 digit (constant) —> word
		658.5	Word, 4 digits (constant) —> word
		456	TIM/CNT, 1 digit (TIM/CNT) —> word
		1,080	TIM/CNT, 4 digits (TIM/CNT) —> word
		145	When shifting one word
		743	When shifting 64 DM words
84	SFTR	136 to 668	When resetting 1 to 64 DM words
		44	NOP
		42	IL
91	SBS	75	Always
92	SBN	26	Always
93	RET	49	Always
97	IORF	108	When refreshing 1 word

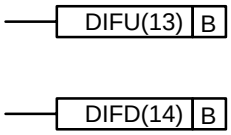
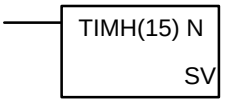
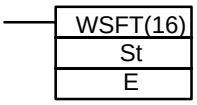
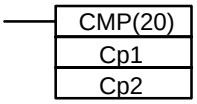
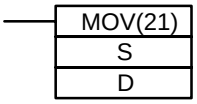
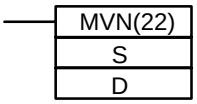
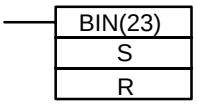
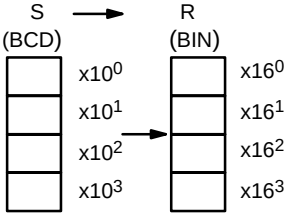
Ladder Diagram Instructions

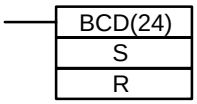
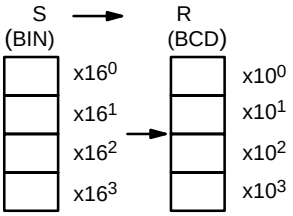
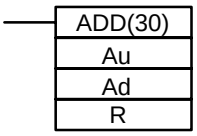
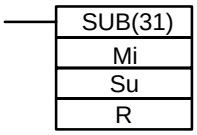
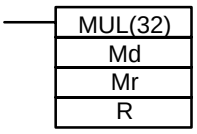
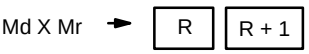
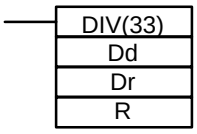
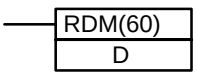
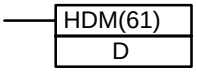
Name Mnemonic	Symbol	Function	Operands
LOAD LD		Used to start instruction block with status of designated bit.	**B:** IR SR HR TC TR
LOAD NOT LD NOT		Used to start instruction block with inverse of designated bit.	**B:** IR SR HR TC TR
AND AND		Logically ANDs status of designated bit with execution condition.	**B:** IR SR HR TC TR
AND NOT AND NOT		Logically ANDs inverse of designated bit with execution condition.	**B:** IR SR HR TC TR
OR OR		Logically ORs status of designated bit with execution condition.	**B**: IR SR HR TC TR
OR NOT OR NOT		Logically ORs inverse of designated bit with execution condition.	**B:** IR SR HR TC TR
AND LOAD AND LD		Logically ANDs results of preceding blocks.	None
OR LOAD OR LD		Logically ORs results of preceding blocks.	None
Refer to table at beginning of Appendix B for page references.			

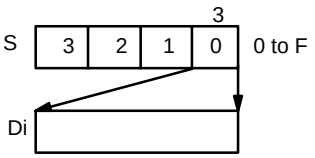
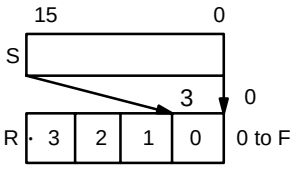
Name Mnemonic	Symbol	Function	Operands
OUTPUT OUT		Turns ON designated bit.	**B:** IR HR TR
OUTPUT NOT OUT NOT		Turns OFF designated bit.	**B:** IR HR TR
TIMER TIM		ON-delay (decrementing) timer operation. Set value: 999.9 s; accuracy: +0.0/-0.1 s. Same TC bit cannot be assigned to more than one timer/counter. The TC bit is input as a constant.	**N:** TC **SV**: IR HR #
COUNTER CNT		A decrementing counter. SV: 0 to 9999; CP: count pulse; R: reset input. The TC bit is input as a constant.	**N:** TC **SV:** IR HR #
Refer to table at beginning of Appendix B for page references.			

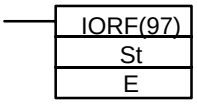
Special Instructions

Name Mnemonic	Symbol	Function	Operands
NO OPERATION NOP (00)	None	Nothing is executed and next instruction is moved to.	None
END END(01)		Required at the end of the program.	None
INTERLOCK IL(02) **INTERLOCK CLEAR** ILC(03)		If interlock condition is OFF, all outputs are turned OFF and all timer PVs reset between this IL(02) and the next ILC(03). Other instructions are treated as NOP; counter PV are maintained.	None
JUMP JMP(04) **JUMP END** JME(05)		Cause all instructions between JMP(04) and the corresponding JME(05) to be ignored. Corresponding JME is next one in program; only 8 JMP-JME pairs allowed per program.	None
STEP DEFINE STEP(08)		Is used in the definition of program sections. STEP N marks the beginning of the section identified by N. STEP without an operand indicates the end of a series of program sections.	**N:** HR
STEP START SNXT(09)		SNXT resets the timers and clears the data areas used in the previous program section. SNTX must also be present at the end of a series of program sections.	**N:** HR
SHIFT REGISTER SFT(10)		Creates a bit shift register from the starting word (St) through the ending word (E). I: input bit; P: shift pulse; R: reset input. St must be less than or equal to E and Bg and E must be in the same data area. 	**St/E:** IR HR
KEEP KEEP(11)		Defines a bit (B) as a latch controlled by set (S) and reset (R) inputs.	**B:** IR HR
REVERSIBLE COUNTER CNTR (12)		Increases or decreases PV by one whenever the increment input (II) or decrement input (DI) signal goes from OFF to ON. SV: 0 to 9999; R: reset input. Must not access the same TC bit as another timer/counter. The TC bit is input as a constant.	**N:** TC **SV:** IR HR #
Refer to table at beginning of Appendix B for page references.			

Name Mnemonic	Symbol	Function	Operands
DIFFERENTIATE UP DIFU(13) **DIFFERENTIATE DOWN** DIFD(14)		DIFU turns ON the designated bit (B) for one cycle on the rising edge of the input signal; DIFD turns ON the bit for one cycle on the trailing edge. The maximum number of DIFU/DIFDs is 48.	**B:** IR HR
HIGH-SPEED TIMER TIMH(15)		A high-speed ON-delay (decrementing) timer. SV: 0.02 to 99.99 s. Must not be assigned the same TC bit as another timer/counter. The TC bit is input as a constant.	**N:** TC **SV**: IR HR #
WORD SHIFT WSFT(16)		Shifts data between the start and end words in word units.	**St/E:** IR HR DM
COMPARE CMP(20)		Compares two sets of four-digit hexadecimal data (Cp1 and Cp2) and outputs result to GR, EQ, and LE.	**Cp1/Cp2:** IR SR HR TC DM #
MOVE MOV(21)		Transfers source data (S) (word or four-digit constant) to destination word (D).	**S:** IR SR HR TC DM # **D:** IR HR DM
MOVE NOT MVN(22)		Inverts source data (S) (word or four-digit constant) and then transfers it to destination word(D).	**S:** IR SR HR TC DM # **D:** IR HR DM
BCD-TO-BINARY BIN(23)		Converts four-digit, BCD data in source word (S) into 16-bit binary data, and outputs converted data to result word (R). 	**S:** IR SR HR TC DM **R:** IR HR DM
Refer to table at beginning of Appendix B for page references.			

Name Mnemonic	Symbol	Function	Operands
BINARY-TO-BCD BCD(24)		Converts binary data in source word (S) into BCD, and outputs converted data to result word (R). 	**S:** IR SR HR DM **R:** IR HR DM
BCD ADD ADD(30)		Adds two four-digit BCD values (Au and Ad) and content of CY, and outputs result to specified result word (R). 	**Au/Ad** IR SR HR TC DM # **R:** IR HR DM
BCD SUBTRACT SUB(31)		Subtracts both four-digit BCD subtrahend (Su) and content of CY from four-digit BCD minuend (Mi) and outputs result to specified result word (R). 	**Mi/Su:** IR SR HR TC DM # **R:** IR HR DM
BCD MULTIPLY MUL(32)		Multiplies a words data or a four-digit BCD value(Md) and another words data (Mr) and outputs the result to a specified result word (R). 	**Md/Mr** IR SR HR TC DM # **R:** IR HR DM
BCD DIVIDE DIV(33)		Divides a words data or a four-digit BCD dividend (Dd) and another words data (Dr) and outputs result to specified result word (R). 	**Dd/Dr:** IR SR HR TC DM # **R:** IR HR DM
SET CARRY STC(40)		Sets carry flag (i.e., turns CY ON).	None
CLEAR CARRY CLC(41)		CLC clears carry flag (i.e, turns CY OFF).	None
REVERSIBLE DRUM COUNTER RDM(60)		High-speed UP-DOWN counter operation.	**D:** IR HR DM
HIGH-SPEED DRUM COUNTER HDM(61)		A 2-kHz counter with both software and hardware resets.	**D:** IR HR DM
Refer to table at beginning of Appendix B for page references.			

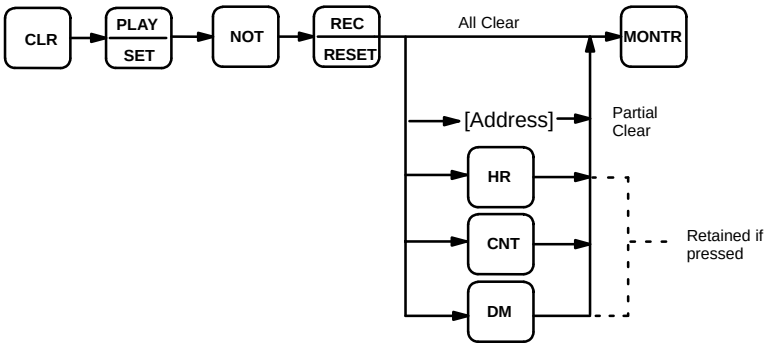
Name Mnemonic	Symbol	Function	Operands
END WAIT ENDW(62)	ENDW(62) N	Used to force a cycle time longer than normal causing the CPU to wait.	**N:** IR HR TC DM #
NOTATION INSERT NETW(63)	NETW(63) C1 C2	Used to leave comments in the program.	#
4-TO-16 DECODER MLPX(76)	MLPX(76) S Di R	Converts up to four hexadecimal digits in source word (S) into decimal values from 0 to 15 and turns ON, in result word(s) (R), bit(s) corresponding to converted value. Digits designated in Di digits (rightmost digit: first digit to be converted; next digit to left: number of digits to be converted minus 1). 	**S:** IR SR HR TC DM **Di:** IR HR TC DM # **R:** IR HR DM
16-TO-4 ENCODER DMPX(77)	DMPX(77) S R Di	Determines position of highest ON bit in source word(s) (starting word: S) and turns ON corresponding bit(s) in result word (R). Digit designations made with Di digits (rightmost digit: first digit to receive converted value; next digit to left: number of words to be converted minus 1). 	**S:** IR SR HR TC DM **R:** IR HR DM **Di:** IR HR TC DM #
REVERSIBLE SHIFT REGISTER SFT(84)	SFTR(84) C St E	Shifts data in a specified word or series of words one bit to either the left or the right.	**St/E:** IR HR DM **C:** IR HR DM
SUBROUTINE ENTER SBS(91)	SBS(91) N	Transfers control of a program over to a subroutine N.	**N:** 00 to 15
SUBROUTINE DEFINE SBN(92)	SBN(92) N	Indicates the beginning of a subroutine definition.	**N:** 00 to 15
Refer to table at beginning of Appendix B for page references.			

Name Mnemonic	Symbol	Function	Operands
RETURN RET(93)		Indicates the end of a subroutine definition.	None
I/O REFRESH IORF(97)		Refreshes I/O words between a specified range. Refreshes words in word units.	**St/E:** 00 to 09
Refer to table at beginning of Appendix B for page references.			

Appendix C
Programming Console Operations

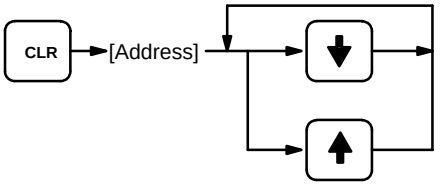
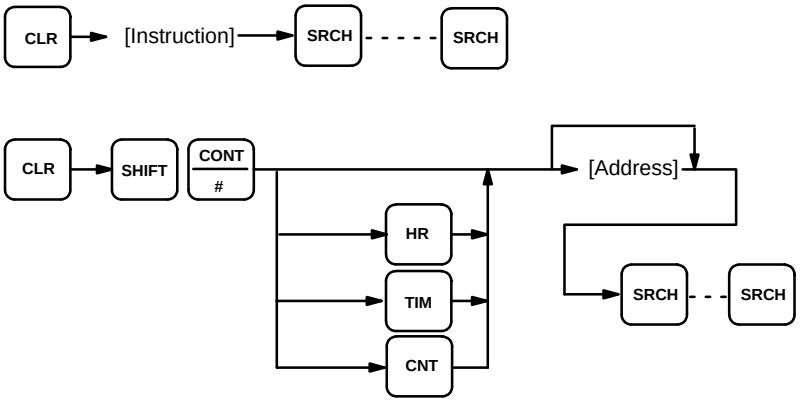
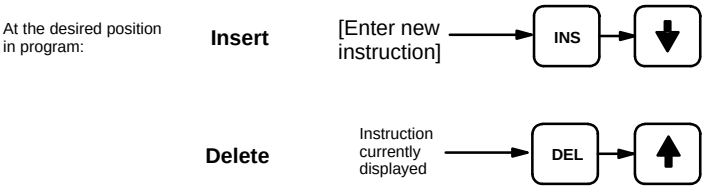
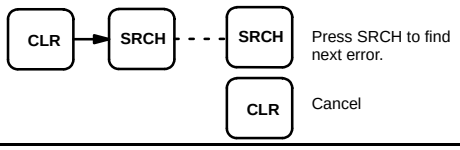
Name	Function	Page
Data Clear	Used to erase data, either selectively or totally, from the Program Memory and the IR, AR, HR, DM, and TC areas.	47
Address Designation	Displays the specified address.	50
Program Search	Searches a program for the specified data address or instruction.	55
Instruction Insert Instruction Delete	Allows a new instruction to be inserted before the displayed instruction, or deletes the displayed instruction (respectively).	57
Program Check	Checks the completed program for syntax errors (up to three levels in H-type PCs).	53
Error Message Read	Displays error messages in sequence, starting with the most severe messages.	148
Bit/Word Monitor	Displays the specified address whose operand is to be monitored. In RUN or MONTR mode it will show the status of the operand for any bit or word in any data area.	150
Forced Set/Reset	Set: Used to turn ON bits or timers, or to increment counters currently displayed on the left of the screen. Reset: Used to turn OFF bits, or to reset timers or counters.	153
Hex/BCD Data Change	Used to change the value of the leftmost BCD or hexadecimal word displayed during a Bit/Word Monitor operation.	155
SV Change SV Reset	Alters the SV of a timer or counter either by incrementing or decrementing the value, or by overwriting the original value with a new one.	156
Program Memory Save	Saves Program Memory to tape.	158
Program Memory Restore	Reads Program Memory from tape.	159
Program Memory Compare	Compares Program Memory data on tape with that in the Program Memory area.	159

System Operations

Operation/Description	Modes*	Key sequence
Data Clear Unless otherwise specified, this operation will clear all erasable memory in Program Memory and IR, HR, AR, DM, and TC areas. To clear EPROM memory the write enable switch must be ON (i.e., enabled). The branch lines shown are used only when performing a partial memory clear, with each of the memory areas entered being retained. Specifying an address will result in the Program Memory after and including that address being deleted. All memory up to that address will be retained.	P	

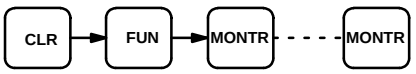
*Modes in which the given instruction is applicable: R = RUN, M = MONITOR, P = PROGRAM

Programming Operations

Operation/Description	Modes*	Key sequence
Address Designation Displays the specified address. Can be used to start programming from a non-zero address or to access an address for editing. Leading zeros need not be entered. The contents of the address will not be displayed until the down key is pressed. The up and down keys can then be used to scroll through the Program Memory.	R P M	
Program Search Allows the program to be searched for occurrences of any designated instruction or data area address. To designate a bit address, press SHIFT, CONT/#, and then input the address. Then press SRCH. Pressing SRCH again will find the next occurrence. In RUN or MONITOR mode, the ON/OFF status of each monitored bit will also be displayed. Applicable data areas vary according to the PC being used.	R P M	
Instruction Insert and Instruction Delete The displayed instruction can be deleted, or another instruction can be inserted before it. Care should be taken to avoid inadvertent deletions as there is no way of recovering the instructions other than to re-enter them. When an instruction is deleted all subsequent instruction addresses are automatically adjusted so that there are no empty addresses, or instructions without addresses.	P	
Program Check Once a program has been entered, it should be checked for errors. The address where the error was generated will be displayed.	P	

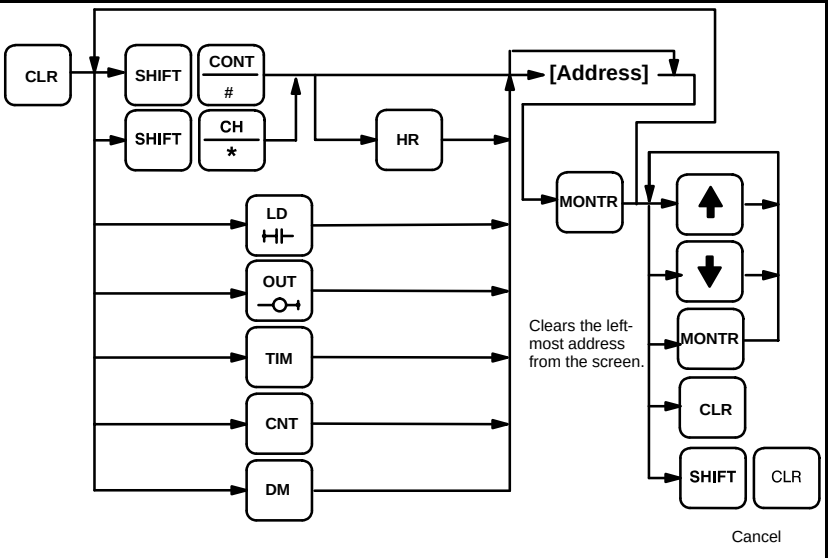
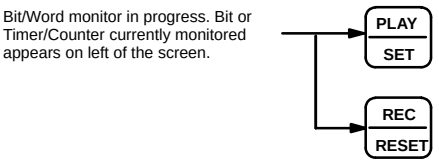
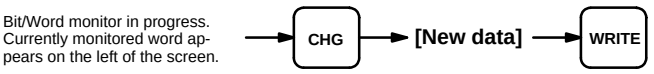
*Modes in which the given instruction is applicable: R = RUN, M = MONITOR, P = PROGRAM

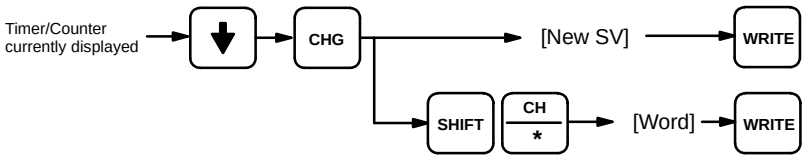
Debugging Operations

Operation/Description	Modes*	Key sequence
Error Message Read Displays error messages in sequence with most severe messages displayed first. Press monitor to access remaining messages. In PROGRAM mode, pressing MONTR clears the displayed message from memory and the next message is displayed.	R P M	

*Modes in which the given instruction is applicable: R = RUN, M = MONITOR, P = PROGRAM

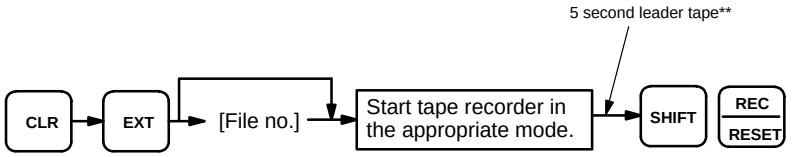
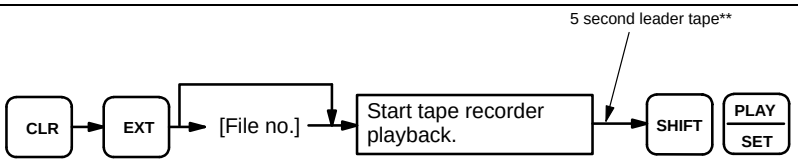
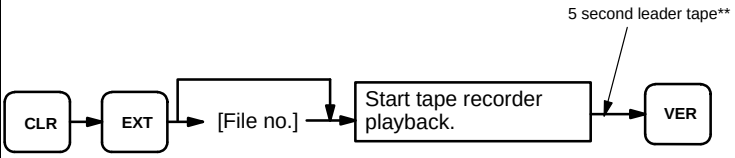
Monitoring and Data Changing Operations

Operation/Description	Modes*	Key sequence
Bit/Word Monitor Up to six memory addresses, containing either words or bits, or a combination of the two, can be monitored at once. Only three can be displayed at any one time. If operated in RUN or MONITOR mode, the status of monitored bits will also be displayed. The operation can be started from a cleared display by entering the address of the first word or bit to be monitored and pressing MONTR, or from any address in the program by displaying the address of the bit or word to be monitored and pressing MONTR. When a timer or counter is monitored, its PV will be displayed and a box is displayed in the bottom left hand corner if the Completion Flag is ON. Applicable data areas vary according to the PC being used.	R P M	
Forced Set/Reset If a bit, timer, or counter address is leftmost on the screen during a Bit/Word Monitor operation, pressing PLAY/SET will turn ON the bit, start the timer, or increment the counter. Pressing REC/RESET will turn OFF the bit, or reset the timer or counter. These force-sets and force-resets are effective for one cycle. Timers will not operate in PROGRAM mode. SR bits are not affected by this operation.	P M	
Hex/BCD Data Change Used to edit the leftmost BCD or hexadecimal value displayed during a Bit/Word Monitor operation. If a timer or counter is leftmost on the display, the PV will be the value displayed and affected by this operation. It can only be changed in MONITOR mode and only while the timer or counter is operating. SR words cannot be changed using this operation.	P M	

Operation/Description	Modes*	Key sequence
SV Change, **SV Reset** There are three ways of modifying the SVs for timers and counters. One method is to enter a new value. The second is to increment or decrement the existing SV. In MONITOR mode the SV can be changed while the program is being executed. Incrementing and decrementing can only be carried out if the SV has been entered as a constant. The third method is to change the value properties from that of a constant to a word address, or vice versa. Note that the display clears after pressing the CHG key and the subsequent keystrokes determine whether the new data will be entered as a word address (pressing SHIFT CH/* plus Word address) or a constant (entering data only).	P M M	

*Modes in which the given instruction is applicable: R = RUN, M = MONITOR, P = PROGRAM

Cassette Tape Operations

Operation/Description	Modes*	Key sequence
Program Memory Save Copies data from the Program Memory to tape. The file no. specified in the instructions provides an identifying address for the information within the tape. Each file number should be used only once per tape. If only a part of the Program Memory is to be stored, the appropriate start and stop addresses must be entered. Each C60 tape can store approximately 16K words on each side of the tape. When the start address is entered, the maximum stop address is set as the default. Do not set a stop address greater than this one. If you wish to record past this address the additional information will need to be recorded either on the flip side of the tape or on a separate tape. After starting the tape recorder, wait about 5 seconds before pressing SHIFT REC/RESET. This is to allow the leader tape to pass before the data transmission starts.	P	
Program Memory Restore To read Program Memory data which has been recorded on a cassette tape, the keystrokes are as given here. The file number must be the same as the one entered when the data was recorded. The read operation will proceed from the specified start address up to the end of the tape, unless halted by a CLR command. The instruction must be completed before the required data is reached on the tape, i.e., usually before the leader tape finishes.	P	
Program Memory Compare The procedure to compare Program Memory data stored on a tape with that in the PC's Program Memory area is the same as that for reading it (see above), except that after starting the tape playback, VER should be pressed instead of SHIFT and PLAY/SET.	P	

*Modes in which the given instruction is applicable: R = RUN, M = MONITOR, P = PROGRAM

**These times take the cassette leader tape into consideration according to the following:
a) When recording to tape, the leader tape needs to be allowed to pass before the data transmission to the tape player starts.
b) When restoring from tape or comparing data, the Programming Console needs to be ready to receive data before the data is transfered from the tape.

Appendix D
Error and Arithmetic Flag Operation

The following table shows which instructions 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 subsections of *Section 5 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 TIM, CNT, and CNTR are executed when ER is ON, other 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 not shown do not affect any of the flags in the table.

Instructions	SR 1907 (LE)	SR 1906 (EQ)	SR 1905 (GR)	SR 1904 (CY)	SR 1903 (ER)
END(01)	OFF	OFF	OFF	OFF	OFF
TIM					↕
TIMH(15)					↕
CNT					↕
CNTR(12)					↕
WSFT(16)					↕
CMP(20)	↕	↕	↕		
MOV(21)		↕			
MVN(22)		↕			
BIN(23)		↕			↕
BCD(24)		↕			↕
ADD(30)		↕		↕	↕
SUB(31)		↕		↕	↕
MUL(32)		↕			↕
DIV(33)		↕			↕
STC(40)				ON	ON
CLC(41)				OFF	OFF
MLPX(76)					↕
DMPX(77)					↕
SFTR(84)				↕	↕
SBS(91)					↕

Note: ↕ means that the flag is affected by the result of instruction execution.

Appendix E
Binary–Hexadecimal–Decimal Table

Decimal	BCD	Hex	Binary
00	00000000	00	00000000
01	00000001	01	00000001
02	00000010	02	00000010
03	00000011	03	00000011
04	00000100	04	00000100
05	00000101	05	00000101
06	00000110	06	00000110
07	00000111	07	00000111
08	00001000	08	00001000
09	00001001	09	00001001
10	00010000	0A	00001010
11	00010001	0B	00001011
12	00010010	0C	00001100
13	00010011	0D	00001101
14	00010100	0E	00001110
15	00010101	0F	00001111
16	00010110	10	00010000
17	00010111	11	00010001
18	00011000	12	00010010
19	00011001	13	00010011
20	00100000	14	00010100
21	00100001	15	00010101
22	00100010	16	00010110
23	00100011	17	00010111
24	00100100	18	00011000
25	00100101	19	00011001
26	00100110	1A	00011010
27	00100111	1B	00011011
28	00101000	1C	00011100
29	00101001	1D	00011101
30	00110000	1E	00011110
31	00110001	1F	00011111
32	00110010	20	00100000

Appendix F
Word Assignment Recording Sheets

This appendix contains sheets that can be copied by the programmer to record I/O bit allocations and terminal assignments on the Racks, as well as details of work bits, data storage areas, timers, and counters.

Programmer: **Program:** **Date:** **Page:**

Word:	Unit:	
Bit	**Field device**	**Notes**
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

Word:	Unit:	
Bit	**Field device**	**Notes**
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

Word:	Unit:	
Bit	**Field device**	**Notes**
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

Word:	Unit:	
Bit	**Field device**	**Notes**
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

Programmer: **Program:** **Date:** **Page:**

Area:	Word:	
Bit	**Usage**	**Notes**
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

Area:	Word:	
Bit	**Usage**	**Notes**
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

Area:	Word:	
Bit	**Usage**	**Notes**
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

Area:	Word:	
Bit	**Usage**	**Notes**
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		
11		
12		
13		
14		
15		

Programmer: **Program:** **Date:** **Page:**

Word	Contents	Notes	Word	Contents	Notes

Programmer: **Program:** **Date:** **Page:**

TC address	T or C	Set value	Notes

TC address	T or C	Set value	Notes

Appendix G
Program Coding Sheet

The following page can be copied for use in coding ladder diagram programs. It is designed for flexibility, allowing the user to input all required addresses and instructions.

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.

Programmer: **Program:** **Date:** **Page:**

Address	Instruction	Operand(s)

Address	Instruction	Operand(s)

Address	Instruction	Operand(s)

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 designate the word and/or bit location. For the UM area, an address designates the instruction location (UM area); for the FM area, the block location (FM area), etc.

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 on Units.

Analog Timer Unit A dedicated timer that interfaces through analog signal externally and digital signals internally.

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. Although in a PC all data is ultimately stored in binary form, binary is used to refer to data that is numerically equivalent to the binary value. It is not used to refer to binary-coded decimal. Each four binary bits is equivalent to one hexadecimal digit.

binary-coded decimal A system used to represent numbers so that each four binary bits is numerically equivalent to one decimal digit.

bit The smallest unit of storage in a PC. The status of a bit is either ON or OFF. Four bits equal one digit; sixteen bits, one word. Different bits 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 designator An operand that is used to designate the bit or bits of a word to be used by an instruction.

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.

buffer A temporary storage space for data in a computerized device.

bus bar The line leading down the left and sometimes right side of a ladder diagram. Instruction execution follows down the bus bar, which is the starting point for all instruction lines.

call A process by which instruction execution shifts from the main program to a subroutine. The subroutine may be called by an instruction or by an interrupt.

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 operation.

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 bit are available with different pulse widths.

condition An 'instruction' placed along an instruction line to determine how terminal instruction 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 input directly and in place of a data memory address would hold the value to be used.

control bit A bit in a memory area that is set either from the program or from 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 affect the operation of the controlled system.

counter Either a dedicated number 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.

CPU Unit The CPU Unit contains the CPU and provides a certain number of I/O points.

cycle 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.

cycle time The time required for a single cycle of the ladder-diagram program.

data area An area in the PC's memory that is designed to hold a specific type of data, e.g., the SR area is designed to hold flags and control bits. Memory areas that hold programs are not considered data areas.

data area boundary The highest address available in a data area. When designating an operand that requires multiple words, it is necessary that the highest address in the data area is not exceeded.

debug A process by which a draft program is corrected until it operates as intended. Debugging includes both removal of syntax errors as well as fine-tuning of timing and coordination of control operations.

decimal A number system where all numbers are expressed to the base 10. Although in a PC all data is ultimately stored in binary form, four binary bits are often used to represent one decimal digit, a system called binary-coded decimal.

decrement Decreasing a numeric value by 1.

default A value assumed and automatically set by the PC when a specific value is not input by the user.

definer A number used as an operand for an instruction but that serves to define the instruction itself rather that the data on which the instruction is to operate. Definers include jump numbers, subroutine numbers, etc.

delay In tracing, a value that specifies where tracing 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 data of some sort in an instruction is to be placed as opposed to the location from which data is to be taken for use in the instruction. The location from which data is to be taken is called the source.

differentiation instruction An instruction used to ensure that the operand bit is never turned ON for more than one cycle 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.

digit designator An operand that is used to designate the digit or digits of a word to be used by an instruction.

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 a concept basic to PC Systems.

DM area A data area used to hold word data. A word in the DM area cannot be accessed by bit.

download The process of transferring a program or data from a higher-level computer to a lower-level computer or PC.

electrical noise Electrical ‘static’ that can disturb electronic communications. The ‘snow’ that can appear on a TV screen is an example of the effects of electrical noise.

error code A numeric code output to indicate the existence of and something about the nature of an error. Some error codes are generated by the system; other are defined in the program by the operator.

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 being executed.

execution time The time required for the CPU to execute either an individual instruction or an entire program.

Expansion I/O Unit An Expansion I/O Unit is connected to increase the number of I/O points available.

extended counter A counter created in a program that count higher that any of the standard counters provided by the individual instructions.

extented timer A timer created in a program that times longer that any of the standard timers provided by the individual instructions.

Factory Intelligent Terminal A Programming Device provided with advanced programming and debugging capabilities to facilitate PC operation. The Factory Intelligent Terminal also provides various interfaces for external devices, such as floppy disk drives.

fatal error An error that will stop PC operation and require correction before operation can be continued.

FIT Short for Factory Intelligent Terminal.

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 program.

flicker bit A bit that is programmed to turn ON and OFF at a specific interval.

force reset The process of artificially turning OFF a bit from a Programming Device. Bits are usually turned OFF as a result of program execution.

force set The process of artificially turning ON a bit from 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.

GPC Short for Graphic Programming Console.

Graphic Programming Console A Programming Device provided with advanced programming and debugging capabilities to facilitate PC operation. A Graphic Programming Console is provided with a large display onto which ladder-diagram programs can be written directly in ladder-diagram symbols for input into the PC without conversion to mnemonic form.

hardware error An error originating in the hardware structure of the PC, as opposed to a software error, which ordinates in software (i.e., programs).

hexadecimal A number system where all numbers are expressed to the base 16. Although in a PC all data is ultimately stored in binary form, displays on and inputs through Programming Devices are often expressed in hexadecimal to facilitate operation. Each four binary bits is numerically equivalent to one hexadecimal digit.

Host Link System — One or more host computers connected to one or more PCs through Host Link Units so that the host computer can be used to transfer data to and receive data from the PC(s). Host Link Systems enable centralized management and control of a PC System.

Host Link Unit — An interface used to connect a PC to a host computer in a Host Link System.

host computer — A computer that is used to transfer data or programs to or receive data or programs from a PC in a Host Link System. The host computer is used for data management and overall system control. Host computers are generally small personal or business computers.

HR area — A data area used to store and manipulate data and to preserve data when power to the PC is turned OFF.

I/O capacity — The number of inputs and outputs that a PC is able to handle. This number ranges from around one-hundred for smaller PCs to two-thousand for the largest ones.

I/O devices — The devices to which terminals on I/O Units, Special I/O Units, etc., 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 Link — Created in an Optical Remote I/O System to enable input/output of one or two IR words directly between PCs. The words are input/output between the PC controlling the Master and a PC connected to the Remote I/O System through an I/O Link Unit or an I/O Link Rack.

I/O Link Unit — A Unit used with certain PCs to create an I/O Link in an Optical Remote I/O System.

I/O point — The place at which an input signal enters the PC System or an output signal leaves the PC System. In physical terms, an I/O point corresponds to terminals or connector pins on a Unit; in terms of programming, an I/O point corresponds to an I/O bit in the IR area.

I/O response time — The time required for an output signal to be sent from the PC in response to and input signal received from an external device.

I/O Unit — The most basic type of Unit mounted to a backplane to create a Rack. I/O Units include Input Units and Output Units, each of which is available in a range of specifications. I/O Units do not include Special I/O Units, Link Units, etc.

I/O word — A word in the IR area that is allocated to a Unit in the PC System.

increment — Increasing a numeric value by 1.

initialization error — An error that occurs either in hardware or software before the PC System has actually begun operation, i.e., during initialization.

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. Input often is used abstractly or collectively to refer to incoming signals.

input bit A bit in the IR area that is allocated to hold the status of an input.

input device An external device that sends signal(s) into the PC System.

input point The point at which an input enters the PC System. An input point physically corresponds to terminals or connector pin(s).

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 an action to be carried out and the data to be used in carrying out the action. Instructions can 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 execution an instruction. The execution time for any one instruction can vary with the execution condition for the instruction and the operands used in 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.

interface An interface is the conceptual boundary between systems or devices and usually involves changes in the way the communicated data is represented. Interface devices perform operations as changing the coding, format, or speed of the data.

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.

IR area A data area whose principal function is to hold the status of inputs coming into the system and outputs that are to be set out of the system. Bits and words in the IR are that are used this way are called I/O bits and I/O words. The remaining bits in the IR area are work bits.

jump A type of programming where execution moves directly from one point in a program to a separate point in the program without sequentially executing the instruction in between. Jumps are usually conditional on an execution condition.

jump number A definer used with a jump that defines the points from which and to which a jump is to be made.

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 suggests 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 A hardware or software connection formed between two Units. “Link” can refer either to a part of the physical connection between two Units or a software connection created to data existing at another location (I/O Links).

load The processes of copying data either from an external device or from a storage area to an active portion of the system such as a display buffer. Also, an output device connected to the PC is called a load.

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.

LR area A data area that is used in a PC Link System so that data can be transferred between two or more PCs. If a PC Link System is not used, the LR area is available for use as work bits.

Master Short for Remote I/O Master Unit.

main program All of a program except for the subroutines.

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.

MONITOR mode A mode of PC operation in which normal program execution is possible but in which modification of data held in memory is still possible. Used for monitoring or debugging the PC.

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.

nest Programming one jump within another jump, programming a call to a subroutine from within another subroutine, etc.

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 open 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 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 low voltage or non-conductivity, but can be defined as the opposite of either.

OFF delay The delay produced between the time turning OFF a signal is initiated (e.g., by an input device or PC) and the time 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 high voltage or conductivity, but can be defined as the opposite of either.

ON delay The delay produced between the time a signal is initiated (e.g., by an input device or PC) and the the time 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 longer than one cycle.

operand A 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.

Optical Slave Rack A Slave Rack connected through an Optical Remote I/O Slave Unit.

OR A logic operation whereby the result is true if either one or both of the premises is 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. Output often is used abstractly or collectively to refer to outgoing signals.

output bit A bit in the IR area that is allocated to hold the status to be sent to an output device.

output device An external device that receives a signal(s) from the PC System.

output point The point at which an output leaves the PC System. An output point physically corresponds to terminals or connector pin(s).

output signal A change in the status of a connection leaving the PC. 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 System All of the Units connected to the CPU Unit up to, but not including the I/O devices. The limits of the PC System on the upper end is the PC and the program in its CPU and on the lower end, I/O Units, an I/O Link Unit, etc.

PCB An acronym for printed circuit board.

Peripheral Device Devices connected to a PC System to aid in system operation. Peripheral devices include printers, programming devices, external storage media, etc.

present value The current time left on a timer or the current count of a counter. Present value is abbreviated 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.

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 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 as CPU-mounting models.

Programming Device A peripheral device used to input a program into a PC or to alter or monitor a program already held in the PC. There are dedicated programming devices, such as Programming Consoles, and there are non-dedicated devices, such as a host computer.

PROGRAM mode A mode of operation that allows for inputting and debugging programs but that does not permit normal execution of the program.

PROM Writer A Peripheral Device used to write programs and other data into a ROM for permanent storage and application.

prompt A message or symbol that appears on a display to request input from the operator.

PV Short 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 wired to each other to form control circuits. In a PC, these are replaced by programmable circuits.

Remote I/O Master Unit The Unit in a Remote I/O System through which signals are sent to all other Remote I/O Units. The Remote I/O Master Unit is mounted either to a C200H, C500, C1000H, or C2000H CPU Rack or an Expansion I/O Rack connected to the CPU Rack. Remote I/O Master Unit is generally abbreviated to simply “Master.”

Remote I/O Slave Unit A Unit mounted to a C200H, C500, C1000H, or C2000H Backplane to form a Slave Rack. Remote I/O Slave Unit is generally abbreviated to simply “Slave.”

Remote I/O System A C200H, C500, C1000H, or C2000H system in which remote I/O points are controlled through a Master mounted to a CPU Rack or an Expansion I/O Rack connected to the CPU Rack. K-type PCs can be connected to Remote I/O Systems through I/O Link Units.

Remote I/O Unit Any of the Units in a Remote I/O System. Remote I/O Units include Masters, Slaves, Optical I/O Units, I/O Link Units, and Remote Terminals.

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 point from which the subroutine was called. A return is automatic upon completion of the subroutine and the return is always to.

reversible counter A counter that can be both incremented and decrement depending on a specified condition(s).

reversible shift register A shift register that can shift data in either direction depending on a specified condition(s).

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 time *See cycle time.*

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 a specific condition(s) different from the one that originally caused the bit to turn OFF or ON.

servicing The process whereby the PC provides data to or receives data from external devices or remote I/O or otherwise handles data transactions for Link Systems.

set The process of turning a bit or signal ON.

set value The count from which a counter starts counting down (or, in the case of a reversible counter, the maximum count) or the time from which a timer starts timing. Set value is abbreviated SV.

shift register One or more words in which data is shifted in bit, digit, or word units a specified number of units to the right or left.

Slave Short for Remote I/O Slave Unit.

Slave Rack A C200H, C500, C1000H, or C2000H Rack containing a Remote I/O Slave Unit and controlled through a Remote I/O Master Unit. Slave Racks are generally located away from the CPU Rack.

software error An error that occurs in the execution of a program.

software protects A software means of protecting data from being changed as opposed to a physical switch or other hardware setting.

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 location to which the result of an instruction is to be written is called the destination.

SR area A data area in a PC used mainly for flags, control bits, and other information provided about PC operation. The status of only certain SR bits may be controlled by the operator, i.e., most SR bits can only be read.

subroutine A group of instructions placed after the main program and executed only if called from the main program or activated by an interrupt.

subroutine number A definer used to identify the subroutine that a subroutine call or interrupt activates.

SV Short for set value.

switching capacity The voltage/current that a relay can 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 exit), mistakes in specifying operands within acceptable parameters (e.g., specifying unwritable SR bits as a destination), and mistakes in actual application of instructions (e.g., a call to a subroutine that does not exist).

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 condition on an instruction line(s).

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.

TM area A memory area used to store the results of a trace.

transmission distance The distance that a signal can be transmitted.

TR area A data area used to store execution conditions so that they can be reloaded later for use with other instructions.

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.

Unit In OMRON PC terminology, the word Unit is capitalized to indicate any product sold for a PC System. Though most of the names of these products end with the word Unit, not all do, e.g., a Remote Terminal is referred to in a collective sense as a Unit. Context generally makes any limitations of this word clear.

unit number A number assigned to some Link Units and Special I/O Units to assign words and sometimes other operating parameters to it.

watchdog timer A timer within the system that ensures that the cycle 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. Although a default value of 130 ms is automatically set for the basic time limit, this value can be extended by the program.

Wired Slave Rack A Slave Rack connected through a Wired Remote I/O Slave Unit.

word A unit of storage in memory that consists of 16 bits. All data areas consists of words. Some data areas can be accessed 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 in a work word.

work word A word that can be used for data calculation or other manipulation in programming, i.e., a ‘work space’ in memory. A large portion of the IR area is always reserved for work words. Parts of other areas not required for special purposes may also be used as work words, e.g., I/O words not allocated to I/O Units.

Index

Numbers

16–TO–4 ENCODER – DMPX(77). *See* instruction set

4–TO–16 DECODER – MLPX(76). *See* instruction set

A

ADD(30). *See* instruction set

Always ON/OFF flags. *See* data areas

Analog Timer Unit. *See* instruction set

AND. *See* instruction set

AND LD. *See* instruction set

AND LOAD. *See* instruction set

AND NOT. *See* instruction set

AND NOT – AND NOT. *See* instruction set

arithmetic operation flags. *See* data areas

B

backup. *See* cassette tape operation

Battery Alarm Flag. *See* data areas

BCD ADD – ADD(30). *See* instruction set

BCD DIVIDE – DIV(33). *See* instruction set

BCD MULTIPLY – MUL(32). *See* instruction set

BCD SUBTRACT – SUB(31). *See* instruction set

BCD(24). *See* instruction set

BCD–TO–BINARY CONVERSION – BIN(23). *See* instruction set

BIN(23). *See* instruction set

binary table,

BINARY–TO–BCD CONVERSION – BCD(24). *See* instruction set

bits
- force set/reset,
- monitor,
- self maintaining,

branching. *See* ladder diagram

C

Carry Flag. *See* data areas

cassette tape operation,
- comparing program memory data,
- error messages,
- restoring program memory data,
- saving program memory data,

CLC(41). *See* instruction set

CLEAR CARRY – CLC(41). *See* instruction set

Clock Pulse Bits. *See* data areas

CMP(20). *See* instruction set

CNTR(12). *See* instruction set

COMPARE – CMP(20). *See* instruction set

comparing program/memory data. *See* cassette tape operation

control bits, usage,

control system, definition of,

Counter – CNT. *See* instruction set

CPU indicators,

cycle time
- calculating,
- components,
- displaying,
- for instruction execution, ,
- long duration,

Cycle Time Error Flag. *See* data areas

D

data areas
- components,
- Data Memory area,
- HR,
- IR
  - I/O bits available
    - in CPUs,
    - in Expansion I/O Units,
  - word allocation,
- SR
  - Always OFF/ON Flags,
  - arithmetic flags, ,
  - Battery Alarm Flag,
  - Clock Pulse Bits,
  - Cycle Time Error Flag,
  - Error Flag, ,
  - First Cycle Flag,
  - high speed drum/counter reset,
  - Step Flag,
  - usage,
- structure,
- TC,
- TR,

digit, monitor,

DIV(33). *See* instruction set

DM area. *See* data areas

DMPX(77). *See* instruction set

E

END – END(01). *See* instruction set

END WAIT – ENDW(62). *See* instruction set

END(01). *See* instruction set

ENDW(62). *See* instruction set

Equal Flag. *See* data areas

Error Flag. *See* data areas

errors
- cassette tape operations,
- clearing messages,
- fatal,
- message tables,
- non–fatal,
- reading and clearing messages,
- SR area flags,

F–H

Factory Intelligent Terminal. *See* Peripheral Devices

First Cycle Flag. *See* instruction set

FIT. *See* Peripheral Devices

flags
- execution affect,
- usage,

Floppy Disk Interface Unit. *See* Peripheral Devices

GPC. *See* Peripheral Devices

Graphic Programming Console. *See* Peripheral Devices

Greater Than Flag. *See* data areas

HDM(61). *See* instruction set

high–speed drum counter reset. *See* data areas

HIGH-SPEED DRUM COUNTER – HDM(61). *See* instruction set

HIGH-SPEED TIMER – TIMH(15). *See* instruction set

HR area. *See* data areas

I

I/O bits available
- in CPUs,
- in Expansion I/O Units,

I/O REFRESH – IORF(97). *See* instruction set

I/O response time,

I/O Units. *See* Units

IL(02). *See* instruction set

ILC(03). *See* instruction set

input bit, definition of,

input devices, definition of,

input point, definition of,

input signal, definition of,

instruction set
- ADD(30),
- Analog Timer Unit,
- AND,
  - combining with OR,
  - use in ladder diagrams,
- AND LD,
  - combining with OR LD,
  - use in logic blocks,
- AND NOT,
  - use in ladder diagrams,
- BCD(24),
- BIN(23),
- CLC(41),
- CMP(20),
- CNT,
  - changing set value,
- CNTR(12),
- DIFD(14)
  - as a bit control instruction,
  - use in interlocks,
- DIFU(13)
  - as a bit control instruction,
  - use in interlocks,
- DIV(33),
- DMPX(77),
- END(01), ,
- ENDW(62),
- HDM(61),
- IL(02),
  - converting to mnemonic code,
  - use in branching,
- ILC(03),
  - converting to mnemonic code,
  - use in branching,
- IORF(97),
- JME(05),
- JMP(04),
- KEEP(11)
  - as a bit control instruction,
  - controlling bit status,
- LD,
  - use in ladder diagrams,
- LD NOT,
  - use in ladder diagrams,
- MLPX(76),
- MOV(21),
- MUL(32),
- MVN(22),
- NETW(63),
- NOP(00),
- NOT,
- OR,
  - combining with AND,
  - use in ladder diagrams,
- OR LD,
  - combining with AND LD,
  - use in logic blocks, ,
- OR NOT,
  - use in ladder diagrams,
- OUT,
  - using to control bit status,
- OUT NOT,
  - using to control bit status,

RDM(60),
RET(93),
SBN(92),
SBS(91),
SFT(10),
SFTR(84),
SNXT(09),
STC(40),
STEP(08),
SUB(31),
TIM,
changing set value,
TIMH(15),
WSFT(16),

INTERLOCK – IL(02). *See* instruction set

INTERLOCK CLEAR – ILC(03). *See* instruction set

IORF(97). *See* instruction set

IR area. *See* data areas

J–K

JME(05). *See* instruction set

JMP(04). *See* instruction set

JUMP – JMP(04). *See* instruction set

JUMP END – JME(05). *See* instruction set

KEEP(11). *See* instruction set

L

ladder diagram
branching
use of,
using IL(02) and ILC(03),
using JMP(04) and JME(05),
using TR bits,
converting to mnemonic code,
instructions
combining
AND and OR,
AND LD and OR LD,
controlling bit status
using DIFU(13) and DIFD(14),
using KEEP(11), ,
using OUT and OUT NOT, ,
format,
function of,
notation,
operands, function of,
structure of,
using logic blocks,

Ladder Support Software. *See* Peripheral Devices

LD. *See* instruction set

LD NOT. *See* instruction set

LEDs. *See* CPU indicators

leftmost, definition,

Less Than Flag. *See* data areas

Link Units. *See* Units

LOAD – LD. *See* instruction set

LOAD NOT – LD NOT. *See* instruction set

logic blocks. *See* ladder diagram

LSS. *See* Peripheral Devices

M

memory areas
data areas. *See* data areas
definition of,
program memory. *See* program memory

MLPX(76). *See* instruction set

mnemonic code, converting,

MOV(21). *See* instruction set

MOVE – MOV(21). *See* instruction set

MOVE NOT – MVN(22). *See* instruction set

MUL(32). *See* instruction set

MVN(22). *See* instruction set

N

NETW(63). *See* instruction set

NO OPERATION – NOP(00). *See* instruction set

NOP(00). *See* instruction set

NOT. *See* instruction set

NOTATION INSERT – NETW(63). *See* instruction set

O

OR. *See* instruction set

OR LD. *See* instruction set

OR LOAD. *See* instruction set

OR NOT. *See* instruction set

OR NOT – OR NOT. *See* instruction set

output bit, definition of,

output devices, definition of,

output points, definition of,

output signal, definition of,

P

Peripheral Devices,
  Factory Intelligent Terminal (FIT),
    standard models,
  Floppy Disk Interface Unit,
  Graphic Programming Console (GPC),
    standard models,
  Ladder Support Software (LSS),
  Printer Interface Unit,
  Programming Console, ,
    clearing memory,
    modes of,
    operation of,
    the keyboard,
  PROM Writer,
  standard models,

Printer Interface Unit. *See* Peripheral Devices

program execution,

Program Memory,
  See also memory areas
  structure,

programming
  console operations in table form,
  debugging,
  displaying and clearing error messages,
  inputting, modifying and checking,
  inserting and deleting instructions,
  instruction table,
  instructions. *See* Instruction set
  method,
  overwriting existing programs,
  precautions,
  searching the program,
  setting and reading from memory address,

Programming Console. *See* Peripheral Devices

PROM Writer. *See* Peripheral Devices

R

RDM(60). *See* instruction set

restore. *See* cassette tape operation

RET(93). *See* instruction set

RETURN – RET(93). *See* instruction set

REVERSIBLE COUNTER – CNTR(12). *See* instruction set

REVERSIBLE DRUM COUNTER – RDM(60). *See* instruction set

REVERSIBLE SHIFT REGISTER – SFTR(84). *See* instruction set

rightmost, definition,

S

SBN(92). *See* instruction set

SBS(91). *See* instruction set

SET CARRY – STC(40). *See* instruction set

SFT(10). *See* instruction set

SFTR(84). *See* instruction set

SHIFT REGISTER – SFT(10). *See* instruction set

SINGLE STEP – STEP(08). *See* instruction set

SNXT(09). *See* instruction set

Special I/O Units. *See* Units

SR area. *See* data areas

standard models
  DIN Units,
  Factory Intelligent Terminal,
  Graphic Programming Console,
  I/O Units,
  K–type CPUs,
  Peripheral Devices,
  Special Units,

status indicators. *See* CPU indicators

STC(40). *See* instruction set

Step Flag. *See* data areas

STEP START – SNXT(09). *See* instruction set

STEP(08). *See* instruction set

SUB(31). *See* instruction set

SUBROUTINE ENTER – SBS(91). *See* instruction set

SUBROUTINE START – SBN(92). *See* instruction set

T–W

TC area. *See* data areas

TIM. *See* instruction set

TIMH(15). *See* instruction set

TR area. *See* data areas

TR bits
  converting to mnemonic code,
  use in branching,

UM area. *See* program memory

Units
  definition of,
  I/O Units, definition of,
  Link Units, definition of,
  Special I/O Units, definition of,

words, I/O,

WORDSHIFT – WSFT(16). *See* instruction set

work bits, usage,

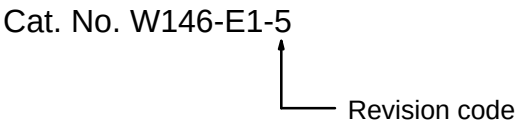
work words, usage,

writing a program
  See also programming
  steps,

WSFT(16). *See* instruction set

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
2	May 1989	Complete rewrite
3	March 1990	IR area portion of *Section 3* completely rewritten. Corrections made to content on the following pages: 2, 6, 7, 11, 12, 18, 36–39, 42, 47, 49, 51, 54–56, 60, 88, 89, 91, 92, 93, 95, 98, 100, 110, 111, 121, 122, 161 A Glossary has been added.
4	May 1992	*Sections 7–1* through *7–4* have been incorporated into *Section 4* and mnemonic code has been added throughout the manual for better clarity. Terms changed: Normal condition to normally open condition and inverse condition to normally closed condition. Instruction names placed in all-caps. Page 6: Data Access Console added. LSS/FIT interfacing requirements corrected. Page 14: Data structure description corrected. Page 15: Decimal point description corrected. Page 17, 18: Accessible bits corrected. Page 19: Number of connectable Units corrected. Page 38: Jump description clarified. Page 61: SV range changed. Accuracy figure removed. Section 5–18: Control bits changed to be consecutive within each program. Last graphic in last example in logic-block instruction description: LD 0500 changed to OR 0500 and LD 0002 changed to AND 0002. Section 5–11–2: Set value range changed from 00.00 and 99.99 to 00.02 and 99.99. Explanation also added. Page 109: Operand 1101 was changed to 1003 in the diagram. Page 116: Values within the top table have been changed. Pages 117, 118: Values on this page have been changed. Pages 178 through 181: Information relating to the standards has changed.
4A	May 1993	Scan time changed to cycle time throughout the manual. **Pages 14 and 15:** Accessible bits corrected. **Page 16:** Table of numbers of I/O points for various system configurations added. **Page 20:** Control bits related to counter instructions added. **Page 82:** HDM(61) and RDM(60) added to section introduction. **Pages 95, 103, 104:** IR bits corrected to SR bits. **Pages 95 to 103:** Counter number specified for HDM(61) in programming. **Pages 103, 104:** “High-speed” removed from description of RDM(60) and warning added that RDM(60) is not a high-speed counter. **Page 105:** Counter number specified for RDM(60) in programming. **Page 128:** Maximum number of steps in one program specified as being 16. **Page 170:** PROM Writer model number updated to C500-PRW06.
4B	December 1994	Address change.
5	July 1999	*Precautions* section added in front of Section 1.