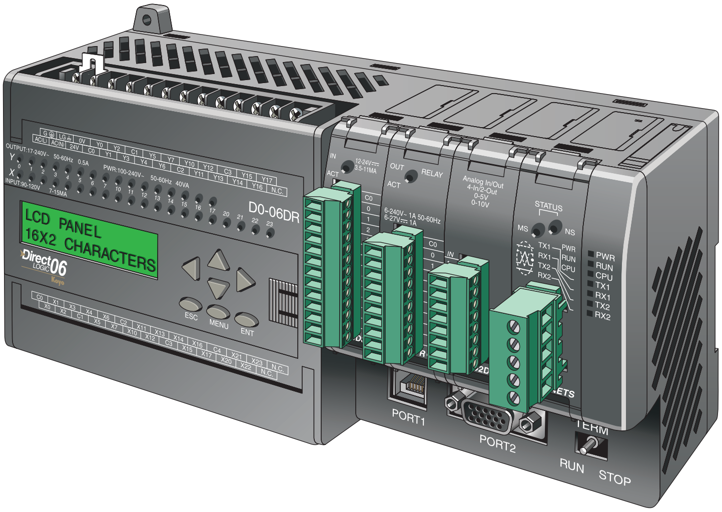
AUTOMATIONDIRECT.com

DL06 Micro PLC User Manual

Volume 1 of 2

Manual Number: D0-06USER-M



WARNING

Thank you for purchasing automation equipment from **Automationdirect.com®**, doing business as, **AutomationDirect**. We want your new automation equipment to operate safely. Anyone who installs or uses this equipment should read this publication (and any other relevant publications) before installing or operating the equipment.

To minimize the risk of potential safety problems, you should follow all applicable local and national codes that regulate the installation and operation of your equipment. These codes vary from area to area and usually change with time. It is your responsibility to determine which codes should be followed, and to verify that the equipment, installation, and operation is in compliance with the latest revision of these codes.

At a minimum, you should follow all applicable sections of the National Fire Code, National Electrical Code, and the codes of the National Electrical Manufacturer's Association (NEMA). There may be local regulatory or government offices that can also help determine which codes and standards are necessary for safe installation and operation.

Equipment damage or serious injury to personnel can result from the failure to follow all applicable codes and standards. We do not guarantee the products described in this publication are suitable for your particular application, nor do we assume any responsibility for your product design, installation, or operation.

Our products are not fault-tolerant and are not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the product could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). **AutomationDirect** specifically disclaims any expressed or implied warranty of fitness for High Risk Activities.

For additional warranty and safety information, see the Terms and Conditions section of our catalog. If you have any questions concerning the installation or operation of this equipment, or if you need additional information, please call us at 770-844-4200.

This publication is based on information that was available at the time it was printed. At **AutomationDirect** we constantly strive to improve our products and services, so we reserve the right to make changes to the products and/or publications at any time without notice and without any obligation. This publication may also discuss features that may not be available in certain revisions of the product.

Trademarks

This publication may contain references to products produced and/or offered by other companies. The product and company names may be trademarked and are the sole property of their respective owners. **AutomationDirect** disclaims any proprietary interest in the marks and names of others.

Copyright 2016, Automationdirect.com Incorporated
All Rights Reserved

No part of this manual shall be copied, reproduced, or transmitted in any way without the prior, written consent of **Automationdirect.com Incorporated**. **AutomationDirect** retains the exclusive rights to all information included in this document.

AVERTISSEMENT

Nous vous remercions d'avoir acheté l'équipement d'automatisation de **Automationdirect.comMC**, en faisant des affaires comme, **AutomationDirect**. Nous tenons à ce que votre nouvel équipement d'automatisation fonctionne en toute sécurité. Toute personne qui installe ou utilise cet équipement doit lire la présente publication (et toutes les autres publications pertinentes) avant de l'installer ou de l'utiliser.

Afin de réduire au minimum le risque d'éventuels problèmes de sécurité, vous devez respecter tous les codes locaux et nationaux applicables régissant l'installation et le fonctionnement de votre équipement. Ces codes diffèrent d'une région à l'autre et, habituellement, évoluent au fil du temps. Il vous incombe de déterminer les codes à respecter et de vous assurer que l'équipement, l'installation et le fonctionnement sont conformes aux exigences de la version la plus récente de ces codes.

Vous devez, à tout le moins, respecter toutes les sections applicables du Code national de prévention des incendies, du Code national de l'électricité et des codes de la National Electrical Manufacturer's Association (NEMA). Des organismes de réglementation ou des services gouvernementaux locaux peuvent également vous aider à déterminer les codes ainsi que les normes à respecter pour assurer une installation et un fonctionnement sûrs.

L'omission de respecter la totalité des codes et des normes applicables peut entraîner des dommages à l'équipement ou causer de graves blessures au personnel. Nous ne garantissons pas que les produits décrits dans cette publication conviennent à votre application particulière et nous n'assumons aucune responsabilité à l'égard de la conception, de l'installation ou du fonctionnement de votre produit.

Nos produits ne sont pas insensibles aux défaillances et ne sont ni conçus ni fabriqués pour l'utilisation ou la revente en tant qu'équipement de commande en ligne dans des environnements dangereux nécessitant une sécurité absolue, par exemple, l'exploitation d'installations nucléaires, les systèmes de navigation aérienne ou de communication, le contrôle de la circulation aérienne, les équipements de survie ou les systèmes d'armes, pour lesquels la défaillance du produit peut provoquer la mort, des blessures corporelles ou de graves dommages matériels ou environnementaux («activités à risque élevé»). La société **AutomationDirect** nie toute garantie expresse ou implicite d'aptitude à l'emploi en ce qui a trait aux activités à risque élevé.

Pour des renseignements additionnels touchant la garantie et la sécurité, veuillez consulter la section Modalités et conditions de notre documentation. Si vous avez des questions au sujet de l'installation ou du fonctionnement de cet équipement, ou encore si vous avez besoin de renseignements supplémentaires, n'hésitez pas à nous téléphoner au 770-844-4200.

Cette publication s'appuie sur l'information qui était disponible au moment de l'impression. À la société **AutomationDirect**, nous nous efforçons constamment d'améliorer nos produits et services. C'est pourquoi nous nous réservons le droit d'apporter des modifications aux produits ou aux publications en tout temps, sans préavis ni quelque obligation que ce soit. La présente publication peut aussi porter sur des caractéristiques susceptibles de ne pas être offertes dans certaines versions révisées du produit.

Marques de commerce

La présente publication peut contenir des références à des produits fabriqués ou offerts par d'autres entreprises. Les désignations des produits et des entreprises peuvent être des marques de commerce et appartiennent exclusivement à leurs propriétaires respectifs. **AutomationDirect** nie tout intérêt dans les autres marques et désignations.

Copyright 2016, Automationdirect.com Incorporated
Tous droits réservés

Nulle partie de ce manuel ne doit être copiée, reproduite ou transmise de quelque façon que ce soit sans le consentement préalable écrit de la société **Automationdirect.com Incorporated**. **AutomationDirect** conserve les droits exclusifs à l'égard de tous les renseignements contenus dans le présent document.

DL06 Micro PLC USER MANUAL

AUTOMATIONDIRECT.com

Please include the Manual Number and the Manual Issue, both shown below, when communicating with Technical Support regarding this publication.

Manual Number:	**D0-06USER-M**
Issue:	**3rd Edition, Rev. E**
Issue Date:	**8/18**

Publication History		
Issue	**Date**	**Description of Changes**
First Edition	7/02	Original
Rev. A	10/02	Updated drawing images and made minor corrections.
Rev. B	6/03	Added new PLC and made numerous corrections.
2nd Edition	3/04	Added two appendices, removed discrete module data and made numerous corrections.
3rd Edition	3/07	Corrected all tables, many corrections to Chapters 2, 3, 4, 5, 6, and 7; Chapter 3 (HSIO) was moved to the Appendices and Chapter 4 was divided into Chapters 3 & 4; added DS5 Intelligent Boxes to Chapter 5; added Ramp/Soak example to Chapter 8; Numbering Systems and Serial Communications were added to Appendices; many minor corrections were made throughout manual.
Rev. A	5/07	Minor corrections and updates.
Rev. B	6/11	Updated Chapter 5 with current ***Direct***SOFT dialog views, corrected number of registers needed to use the print message instruction, removed fuses and corrected I/O wiring drawings, and other minor corrections and updates.
Rev. C	2/13	Added H0-CTRIO2 references. Minor corrections and updates. Added transient suppression for inductive loads.
Rev. D	6/16	Corrections and updates.
Rev. E	8/18	Corrections, updates, removed mention of the DA-EU-M from Appendix J.

Notes

TABLE OF CONTENTS

Chapter 1: Getting Started

Chapter 2: Installation, Wiring, and Specifications

Chapter 3: CPU Specifications and Operation

Chapter 4: System Design and Configuration

Chapter 5: Standard RLL Instructions

Chapter 6: Drum Instruction Programming

Volume Two: Table of Contents

Chapter 7: RLL^{PLUS} Stage Programming

Chapter 8: PID Loop Operation

Chapter 9: Maintenance and Troubleshooting

Chapter 10: LCD Display Panel

Appendix A: Auxiliary Functions

Appenedix B: DL06 Error codes

Appendix C: Instruction Execution Times

Appendix D: Special Relays

Appendix E: High-speed Input and Pulse Output Features

Appendix F: PLC Memory

Appendix G: ASCII Table

Appendix H: Product Weights

Appendix I: Numbering Systems

Appendix J: European Union Directives (CE)

Appendix K: Introduction to Serial Communications

Index

Notes

Chapter 1

Getting Started

In This Chapter...

Introduction

The Purpose of this Manual

Thank you for purchasing a DL06 Micro PLC. This manual shows you how to install, program, and maintain all PLCs in the DL06 family. It also helps you understand how to interface them to other devices in a control system. This manual contains important information for personnel who will install DL06 PLCs and for the PLC programmer. This user manual will provide the information you need to get and keep your system up and running.

Supplemental Manuals

The D0–OPTIONS–M manual contains technical information about the option cards available for the DL06 PLCs. This information includes specifications and wiring diagrams that will be indispensable if you use any of the optional I/O or communications cards. If you have purchased one of our operator interface panels or ***Direct***SOFT™ programming software, you will want to refer to the manuals that are written for these products.

Technical Support

We strive to make our manuals the best in the industry. We rely on your feedback to let us know if we are reaching our goal. If you cannot find the solution to your particular application, or, if for any reason you need technical assistance, please call us at

770–844–4200

Our technical support group will work with you to answer your questions. They are available Monday through Friday from 9:00 A.M. to 6:00 P.M. Eastern Time. We also encourage you to visit our web site where you can find technical and non-technical information about our products and our company.

http://www.automationdirect.com

If you have a comment, question or suggestion about any of our products, services, or manuals, please fill out and return the **Suggestions** card included with this manual.

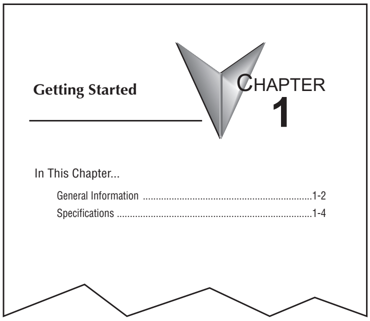
Conventions Used

When you see the notepad icon in the left-hand margin, the paragraph to its immediate right will be a **special note**. Notes represent information that may make your work quicker or more efficient.
The word **NOTE** in boldface type will mark the beginning of the text.

When you see the exclamation point icon in the left-hand margin, the paragraph to its immediate right will be a **warning**. This information could prevent injury, loss of property, or even death in extreme cases. Any warning in this manual should be regarded as critical information that should be read in its entirety.
The word **WARNING** in boldface type will mark the beginning of the text.

Key Topics for Each Chapter

The beginning of each chapter will list the key topics that can be found in that chapter.



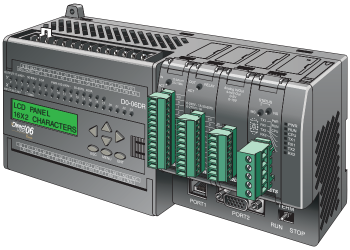
Getting Started

CHAPTER
1

In This Chapter...

DL06 Micro PLC Overview

The DL06 micro PLC family is a versatile product line that combines powerful features and a very compact footprint. The DL06 PLCs offer expandable I/O, high-speed counter, floating point, PID, etc. There are a number of communication options and an optional LCD display.



The DL06 PLC Features

The DL06 Micro PLC family includes nine different versions. All have the same appearance and CPU performance. The CPU offers an instruction set very similar to our powerful new DL260 CPU including new easy to use ASCII and MODBUS instructions. All DL06 PLCs have two built-in communications ports that can be used for programming, operator interface, networking, etc.

Units with DC inputs have selectable high-speed input features on four input points. Units with DC outputs offer selectable pulse output capability on the first and second output points. Details of these features and more are covered in Chapter 3, CPU Specifications and Operation. There are nine versions of the DL06 PLC. The most common industrial I/O types and power supply voltages are available. Consult the following table to find the model number of the PLC that best fits your application.

DL06 Micro PLC Family					
DL06 Part Number	**Discrete Input Type**	**Discrete Output Type**	**External Power**	**High-Speed Input**	**Pulse Output**
D0–06AA	AC	AC	95–240 VAC	No	No
D0–06AR	AC	Relay	95–240 VAC	No	No
D0–06DA	DC	AC	95–240 VAC	Yes	No
D0–06DD1	DC	DC Sinking	95–240 VAC	Yes	Yes
D0–06DD2	DC	DC Sourcing	95–240 VAC	Yes	Yes
D0–06DR	DC	Relay	95–240 VAC	Yes	No
D0–06DD1–D	DC	DC Sinking	12–24 VDC	Yes	Yes
D0–06DD2–D	DC	DC Sourcing	12–24 VDC	Yes	Yes
D0–06DR–D	DC	Relay	12–24 VDC	Yes	No

*Direct*SOFT Programming for Windows™

The DL06 Micro PLC can be programmed with ***Direct*SOFT**, a Windows-based software package that supports familiar features such as cut-and-paste between applications, point-and-click editing, viewing and editing multiple application programs at the same time, floating views, intelligent boxes, etc. Firmware version 2.10 is needed in order to use the intelligent boxes.

*Direct*SOFT (part number PC-DSOFTx) supports the *Direct*LOGIC CPU families. You can use *Direct*SOFT 5 to program the DL05, DL06, DL105, DL205, DL305, and DL405 CPUs. A separate manual discusses *Direct*SOFT programming software. Earlier programming software versions such as *Direct*SOFT32, version 4.0 can also be used to program the DL06.

Handheld Programmer

All DL06 Micro PLCs have a built-in programming port for use with the handheld programmer (D2–HPP), the same programmer used with the DL05, DL105 and DL205 families. The handheld programmer can be used to create, modify and debug your application program. A separate manual discusses the Handheld Programmer. Only D2–HPPs with firmware version 2.0 or later will program the DL06.

NOTE: *Not all instructions are available to use with the HPP - the real number instructions, for example.* ***Direct****SOFT will be needed to program instructions such as these.*

I/O Quick Selection Guide

The nine versions of the DL06 have input/output circuits which can interface to a wide variety of field devices. In several instances a particular input or output circuit can interface to either DC or AC voltages, or both sinking and sourcing circuit arrangements. Check this guide to find the proper DL06 Micro PLC to interface to the field devices in your application.

I/O Selection Guide						
DL06 Part Number	INPUTS			OUTPUTS		
	I/O type/ commons	Sink/Source	Voltage Ranges	I/O type/ commons	Sink/Source	Voltage/ Current Ratings*
D0–06AA	AC / 5	–	90 – 120 VAC	AC / 4	–	17 – 240 VAC, 50/60 Hz 0.5A
D0–06AR	AC / 5	–	90 – 120 VAC	Relay / 4	Sink or Source	6 – 27VDC, 2A 6 – 240 VAC, 2A
D0–06DA	DC / 5	Sink or Source	12 – 24 VDC	AC / 4	–	17 – 240 VAC, 50/60 Hz 0.5A
D0–06DD1	DC / 5	Sink or Source	12 – 24 VDC	DC / 4	Sink	6 – 27 VDC, 0.5A (Y0–Y1) 6 – 27 VDC, 1.0A (Y2–Y17)
D0–06DD2	DC / 5	Sink or Source	12 – 24 VDC	DC / 4	Source	12 – 24 VDC, 0.5A (Y0–Y1) 12 – 24 VDC, 1.0A (Y2–Y17)
D0–06DR	DC / 5	Sink or Source	12 – 24 VDC	Relay / 4	Sink or Source	6 – 27VDC, 2A 6 – 240 VAC, 2A
D0–06DD1–D	DC / 5	Sink or Source	12 – 24 VDC	DC / 4	Sink	6 – 27 VDC, 0.5A (Y0–Y1) 6 – 27 VDC, 1.0A (Y2–Y17)
D0–06DD2–D	DC / 5	Sink or Source	12 – 24 VDC	DC / 4	Source	12 – 24 VDC, 0.5A (Y0–Y1) 12 – 24 VDC, 1.0A (Y2–Y17)
D0–06DR–D	DC / 5	Sink or Source	12 – 24 VDC	Relay / 4	Sink or Source	6 – 27 VDC, 2A 6 – 240 VAC, 2A

* See Chapter 2, Specifications for more information about a particular DL06 version.

Quick Start

This example is not intended to tell you everything you need to know about programming and starting up a complex control system. It is only intended to give you an opportunity to demonstrate to yourself and others the basic steps necessary to power up the PLC and confirm its operation. Please look for warnings and notes throughout this manual for important information you will not want to overlook.

Step 1: Unpack the DL06 Equipment

Unpack the DL06 and gather the parts necessary to build this demonstration system. The recommended components are:

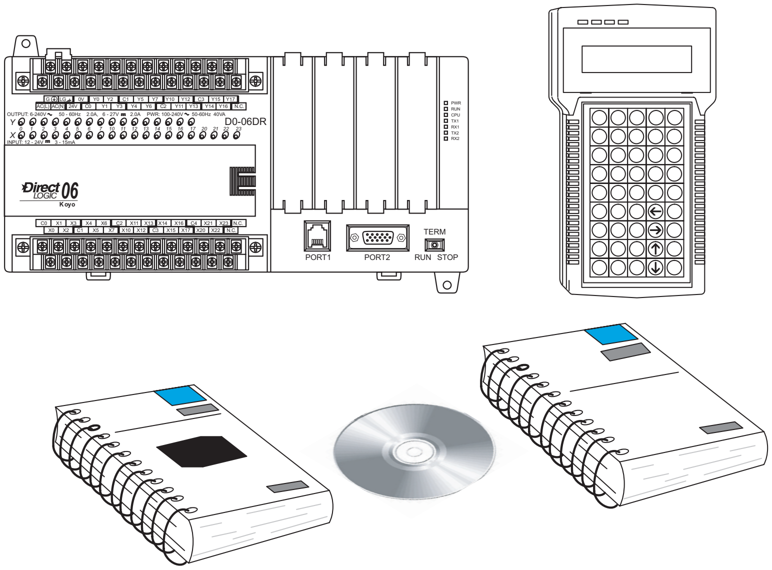
- DL06 Micro PLC
- AC power cord or DC power supply
- Toggle switches (see Step 2 on next page)
- Hook-up wire, 16-22 AWG
- DL06 User Manual (this manual)
- A small screwdriver, 5/8” flat or #1 Philips type

You will need at least one of the following programming options:

- ***Direct*SOFT** Programming Software V5.0 or later (PC-DSOFTx), ***Direct*SOFT** Programming Software Manual (included with the software), and a programming cable (D2-DSCBL connects the DL06 to a personal computer).

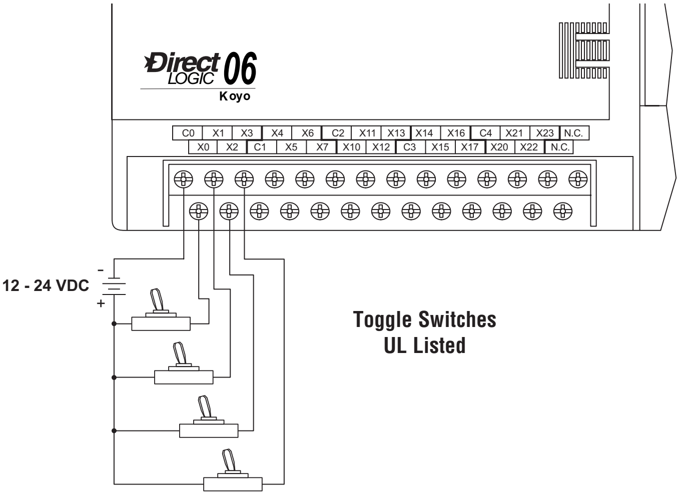
or

- D2-HPP Handheld Programmer, firmware version 2.0 or later, (comes with programming cable). Please purchase Handheld Programmer Manual D2-HPP-M separately.

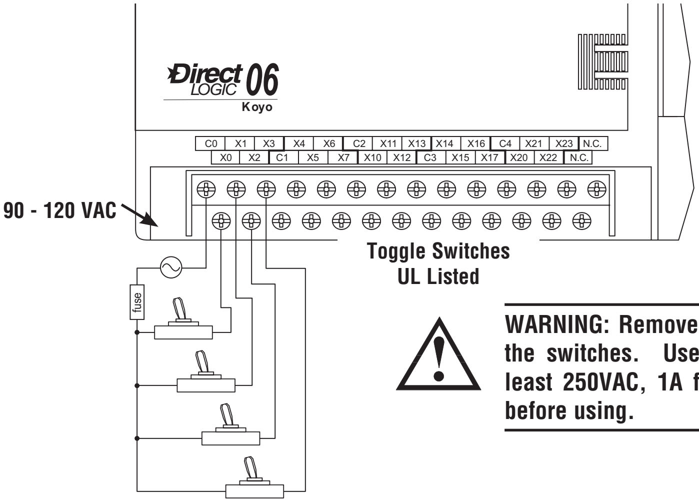


Step 2: Connect Switches to Input Terminals

To proceed with this quick-start exercise or to follow other examples in this manual, you will need to connect one or more input switches as shown below. If you have DC inputs on an AC-supply DL06, you can use the auxiliary 24VDC supply on the output terminal block or other external 12-24VDC power supply. Be sure to follow the instructions in the accompanying WARNING on this page.


DirectLOGIC 06
Koyo
C0 X1 X3 X4 X6 C2 X11 X13 X14 X16 C4 X21 X23 N.C.
X0 X2 C1 X5 X7 X10 X12 C3 X15 X17 X20 X22 N.C.
12 - 24 VDC
-
+
Toggle Switches
UL Listed

D0-06DA, D0-06DD1, D0-06DD2, D0-06DR, D0-DD1-D, and D0-06DR1-D DC Input

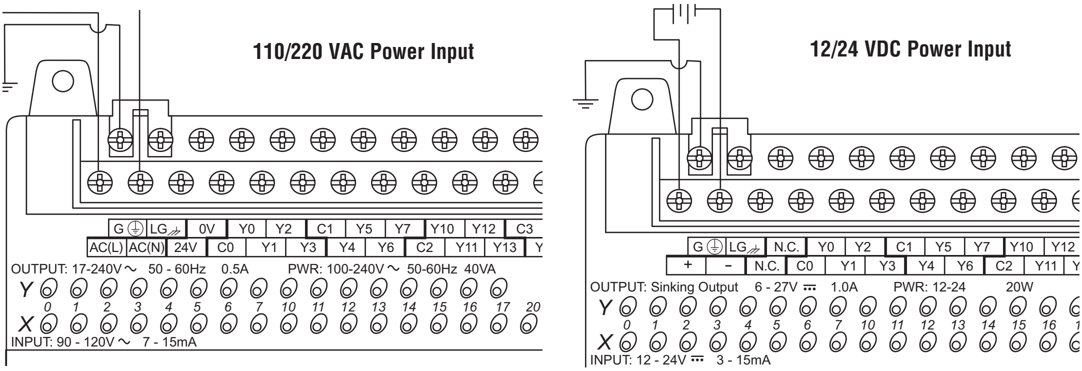

DirectLOGIC 06
Koyo
C0 X1 X3 X4 X6 C2 X11 X13 X14 X16 C4 X21 X23 N.C.
X0 X2 C1 X5 X7 X10 X12 C3 X15 X17 X20 X22 N.C.
90 - 120 VAC
fuse
Toggle Switches
UL Listed

D0-06AA and D0-06AR AC input only

WARNING: Remove power and unplug the DL06 when wiring the switches. Use only UL-approved switches rated for at least 250VAC, 1A for AC inputs. Firmly mount the switches before using.

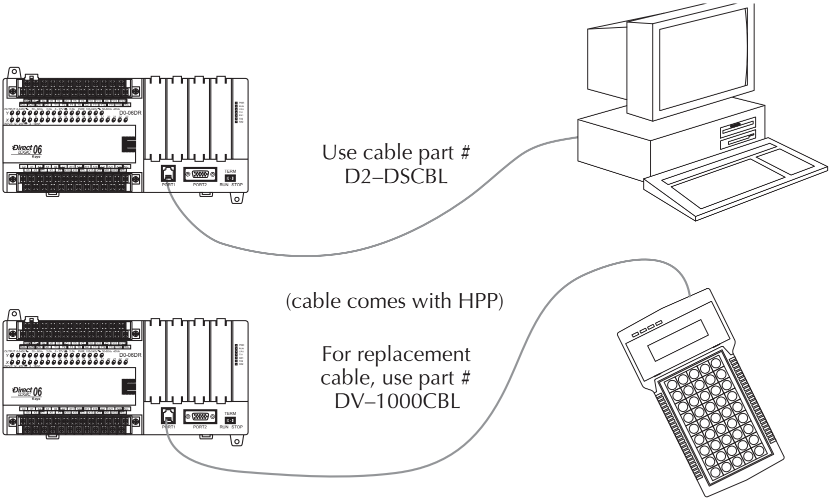
Step 3: Connect the Power Wiring

Connect the power input wiring for the DL06. Observe all precautions stated earlier in this manual. For more details on wiring, see Chapter 2 on Installation, Wiring, and Specifications. When the wiring is complete, close the connector covers. Do not apply power at this time.


110/220 VAC Power Input
G LG 0V Y0 Y2 C1 Y5 Y7 Y10 Y12 C3
AC(L) AC(N) 24V C0 Y1 Y3 Y4 Y6 C2 Y11 Y13 Y
OUTPUT: 17-240V ~ 50 - 60Hz 0.5A PWR: 100-240V ~ 50-60Hz 40VA
Y 0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20
X
INPUT: 90 - 120V ~ 7 - 15mA
12/24 VDC Power Input
G LG N.C. Y0 Y2 C1 Y5 Y7 Y10 Y12
+ – N.C. C0 Y1 Y3 Y4 Y6 C2 Y11 Y
OUTPUT: Sinking Output 6 - 27V ⎓ 1.0A PWR: 12-24 20W
Y 0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 1
X
INPUT: 12 - 24V ⎓ 3 - 15mA

Step 4: Connect the Programming Device

Most programmers will use *Direct*SOFT programming software, installed on a personal computer. An alternative, if you need a compact portable programming device, is the Handheld Programmer (firmware version 2.20 or later). Both devices will connect to COM port 1 of the DL06 via the appropriate cable.



Use cable part # D2–DSCBL

(cable comes with HPP)

For replacement cable, use part # DV–1000CBL

NOTE: *The Handheld Programmer cannot create or access LCD, ASCII or MODBUS instructions.*

Step 5: Switch on the System Power

Apply power to the system and ensure the PWR indicator on the DL06 is on. If not, remove power from the system and check all wiring and refer to the troubleshooting section in Chapter 9 for assistance.

Step 6: Initialize Scratchpad Memory

It’s a good precaution to always clear the system memory (scratchpad memory) on a new DL06. There are two ways to clear the system memory:

- In *Direct*SOFT, select the PLC menu, then Setup and Initialize Scratch Pad. Initializing Scratch Pad will return secondary comm port settings and retentive range settings to default. If you have made any changes to these, you will need to note these changes and re-enter them after initializing Scratchpad.
- For the Handheld Programmer, use the AUX key and execute AUX 54.

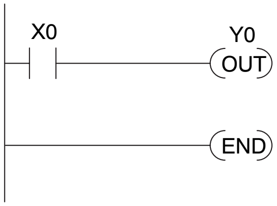
See the Handheld Programmer Manual for additional information.

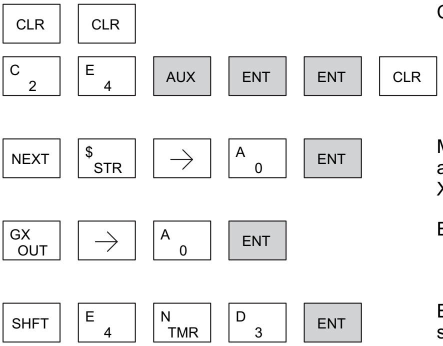
Step 7: Enter a Ladder Program

At this point, *Direct*SOFT programmers need to refer to Chapter 2 (Quick Start) in the *Direct*SOFT Programming Software Manual. There you will learn how to establish a communications link with the DL06 PLC, change CPU modes to Run or Program, and enter a program.

If you are learning how to program with the Handheld Programmer, make sure the CPU is in Program Mode (the RUN LED on the front of the DL06 should be off). If the RUN LED is on, use the MODE key on the Handheld PRogrammer to put the PLC in Program Mode, then switch to TERM.

Equivalent ***Direct*** SOFT display





Clear the Program

Move to the first address and enter X0 contact

Enter output Y0

Enter the END statement

Enter the following keystrokes on the Handheld Programmer.

After entering the simple example program, put the PLC in Run mode by using the Mode key on the Handheld Programmer.

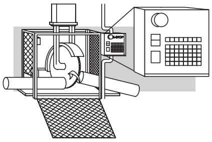
The RUN indicator on the PLC will illuminate, indicating the CPU has entered the Run mode. If not, repeat this step, ensuring the program is entered properly or refer to the troubleshooting guide in chapter 9.

After the CPU enters the run mode, the output status indicator for Y0 should follow the switch status on input channel X0. When the switch is on, the output will be on.

Steps to Designing a Successful System

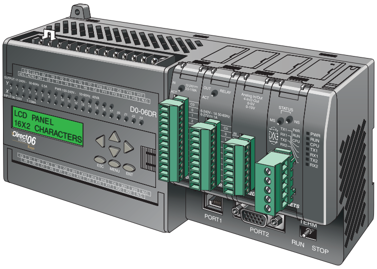
Step 1: Review the Installation Guidelines

Always make safety the first priority in any system design. Chapter 2 provides several guidelines that will help you design a safer, more reliable system. This chapter also includes wiring guidelines for the various versions of the DL06 PLC.



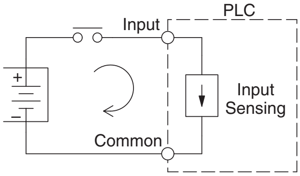
Step 2: Understand the PLC Setup Procedures

The PLC is the heart of your automation system. Make sure you take time to understand the various features and setup requirements.



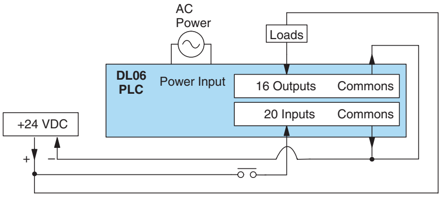
Step 3: Review the I/O Selection Criteria

There are many considerations involved when you select your I/O type and field devices. Take time to understand how the various types of sensors and loads can affect your choice of I/O type.



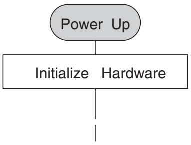
Step 4: Choose a System Wiring Strategy

It is important to understand the various system design options that are available before wiring field devices and field-side power supplies to the Micro PLC.



Step 5: Understand the System Operation

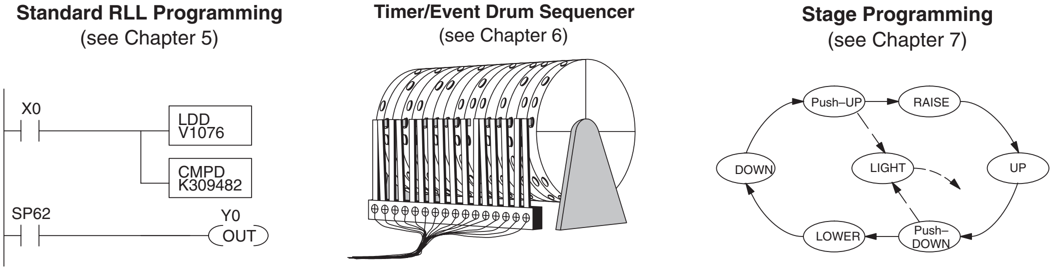
Before you begin to enter a program, it is very helpful to understand how the DL06 system processes information. This involves not only program execution steps, but also involves the various modes of operation and memory layout characteristics.



Step 6: Review the Programming Concepts

The DL06 PLC instruction set provides for three main approaches to solving the application program, depicted in the figure below.

- RLL diagram-style programming is the best tool for solving boolean logic and general CPU register/ accumulator manipulation. It includes dozens of instructions, which will also be needed to augment drums and stages.
- The Timer/Event Drum Sequencer features up to 16 steps and offers both time and/or event-based step transitions. The DRUM instruction is best for a repetitive process based on a single series of steps.
- Stage programming (also called RLL^{PLUS}) is based on state-transition diagrams. Stages divide the ladder program into sections which correspond to the states in a flow chart you draw for your process.



Standard RLL Programming (see Chapter 5)

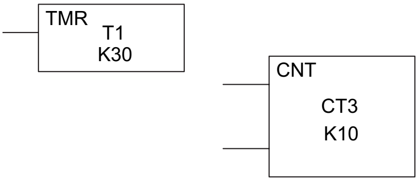
Timer/Event Drum Sequencer (see Chapter 6)

Stage Programming (see Chapter 7)

After reviewing the programming concepts above, you'll be equipped with a variety of tools to write your application program.

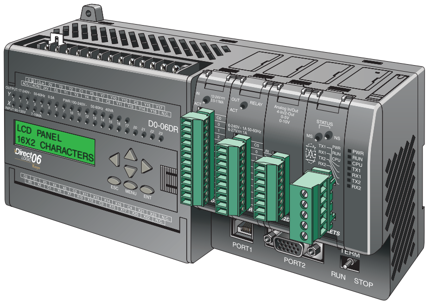
Step 7: Choose the Instructions

Once you have installed the Micro PLC and understand the main programming concepts, you can begin writing your application program. At that time you will begin to use one of the most powerful instruction sets available in a small PLC.



Step 8: Understand the Maintenance and Troubleshooting Procedures

Sometimes equipment failures occur when we least expect it. Switches fail, loads short and need to be replaced, etc. In most cases, the majority of the troubleshooting and maintenance time is spent trying to locate the problem. The DL06 Micro PLC has many built-in features, such as error codes, that can help you quickly identify problems.



Questions and Answers about DL06 Micro PLCs

Q. What is the instruction set like?

A. The instruction set is very close to that of our DL260 CPU. The DL06 instructions include the drum sequencing instruction, networking, ASCII, MODBUS, LCD, intelligent boxes and High-Speed I/O capabilities. High-Speed inputs are available on units with DC inputs only; high-speed outputs are available on units with DC outputs only.

Q. Do I have to buy the full *Direct*SOFT programming package to program the DL06?

A. Yes. The part number for *Direct*SOFT (PC-DSOFT6) is now used for all PLCs in the *Direct*LOGIC family, and the price is very affordable.

Q. Is the DL06 expandable?

A. Yes, the DL06 series function as stand-alone PLCs. However, option card slots allow you to expand the system without changing the footprint.

Q. Does the DL06 have motion control capability?

A. Yes, the DL06 has limited motion control capabilities. The High-Speed I/O features offer either encoder inputs with high-speed counting and presets with interrupt, or a pulse/direction output for stepper control. Three types of motion profiles are available, which are explained in Appendix E. The H0-CTRIO(2) option module can also be used to provide more motion functionality.

Q. Are the ladder programs stored in a removable EEPROM?

A. No. The DL06 contains a non-removable FLASH memory for program storage, which may be written and erased thousands of times. You may transfer programs to/from *Direct*SOFT on a PC.

Q. Does the DL06 contain fuses for its outputs?

A. There are no output circuit fuses. Therefore, we recommend fusing each channel, or fusing each common. See Chapter 2 for I/O wiring guidelines.

Q. Is the DL06 Micro PLC U.L. approved?

A. The Micro PLC has met the requirements of UL (Underwriters' Laboratories, Inc.), and CUL (Canadian Underwriters' Laboratories, Inc.). See our website, ***www.Automationdirect.com***, for complete details.

Q. Does the DL06 Micro PLC comply with European Union (EU) Directives?

A. The Micro PLC has met the requirements of the European Union Directives (CE). See our website, ***www.Automationdirect.com***, for complete details.

Q. Which devices can I connect to the communication ports of the DL06?

A. Port 1: The port is RS-232C, fixed at 9600 baud, odd parity, address 1, and uses the proprietary K-sequence protocol. The DL06 can also connect to MODBUS RTU and DirectNET networks as a slave device through port 1. The port communicates with the following devices:

- DV-1000 Data Access Unit, C-more, DirectTouch, LookoutDirect, DSData or Optimation Operator interface panels
- ***Direct***SOFT (running on a personal computer)
- D2-HPP handheld programmer
- Other devices which communicate via K-sequence, Directnet, MODBUS RTU protocols should work with the DL06 Micro PLC. Contact the vendor for details.

A. Port 2: This is a multi-function port. It supports RS-232C, RS422, or RS485, with selective baud rates (300 - 38,400 bps), address and parity. It also supports the proprietary K-sequence protocol as well as DirectNet and MODBUS RTU, ASCII In/Out and non-sequence/print protocols.

Q. Can the DL06 accept 5VDC inputs?

A. No. 5 volts is lower than the DC input ON threshold. However, many TTL logic circuits can drive the inputs if they are wired as open collector (sinking) inputs. See Chapter 2 for I/O wiring guidelines.

Notes

Installation, Wiring, and Specifications

Chapter 2

In This Chapter...

Safety Guidelines

***NOTE: Products with CE marks** perform their required functions safely and adhere to relevant standards as specified by CE directives, provided they are used according to their intended purpose, and the instructions in this manual are strictly followed. The protection provided by the equipment may be impaired if this equipment is used in a manner not specified in this manual. A listing of our international affiliates is available on our Web site: http://www.automationdirect.com*

WARNING: Providing a safe operating environment for personnel and equipment is your responsibility and should be your primary goal during system planning and installation. Automation systems can fail and may result in situations that can cause serious injury to personnel and/ or damage equipment. Do not rely on the automation system alone to provide a safe operating environment. Sufficient emergency circuits should be provided to stop the operation of the PLC or the controlled machine or process, either partially or totally. These circuits should be routed outside the PLC in the event of controller failure, so that independent and rapid shutdown are available. Devices, such as mushroom switches or end of travel limit switches, should operate motor starter, solenoids, or other devices without being processed by the PLC. These emergency circuits should be designed using simple logic with a minimum number of highly reliable electromechanical components. Every automation application is different, so there may be special requirements for your particular application. Make sure all national, state, and local government requirements are followed for the proper installation and use of your equipment.

Plan for Safety

The best way to provide a safe operating environment is to make personnel and equipment safety part of the planning process. You should examine every aspect of the system to determine which areas are critical to operator or machine safety. If you are not familiar with PLC system installation practices, or your company does not have established installation guidelines, you should obtain additional information from the following sources.

- NEMA — The National Electrical Manufacturers Association, located in Washington, D.C., publishes many different documents that discuss standards for industrial control systems. You can order these publications directly from NEMA. Some of these include:
  ICS 1, General Standards for Industrial Control and Systems
  ICS 3, Industrial Systems
  ICS 6, Enclosures for Industrial Control Systems
- NEC — The National Electrical Code provides regulations concerning the installation and use of various types of electrical equipment. Copies of the NEC Handbook can often be obtained from your local electrical equipment distributor or your local library.
- Local and State Agencies — many local governments and state governments have additional requirements above and beyond those described in the NEC Handbook. Check with your local Electrical Inspector or Fire Marshall office for information.

Three Levels of Protection

The publications mentioned provide many ideas and requirements for system safety. At a minimum, you should follow these regulations. Also, you should use the following techniques, which provide three levels of system control.

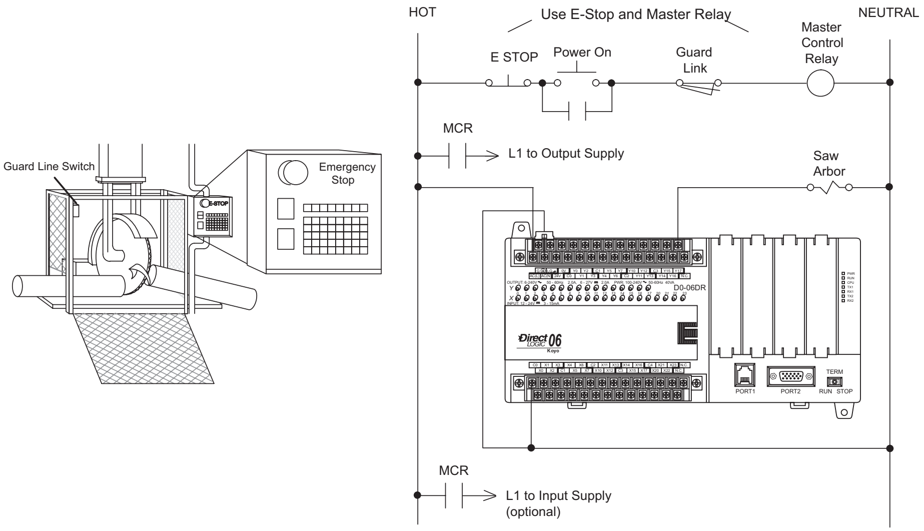
- Emergency stop switch for disconnecting system power
- Mechanical disconnect for output module power
- Orderly system shutdown sequence in the PLC control program

Emergency Stops

It is recommended that emergency stop circuits be incorporated into the system for every machine controlled by a PLC. For maximum safety in a PLC system, these circuits must not be wired into the controller, but should be hardwired external to the PLC. The emergency stop switches should be easily accessed by the operator and are generally wired into a master control relay (MCR) or a safety control relay (SCR) that will remove power from the PLC I/O system in an emergency.

MCRs and SCRs provide a convenient means for removing power from the I/O system during an emergency situation. By de-energizing an MCR (or SCR) coil, power to the input (optional) and output devices is removed. This event occurs when any emergency stop switch opens. However, the PLC continues to receive power and operate even though all its inputs and outputs are disabled.

The MCR circuit could be extended by placing a PLC fault relay (closed during normal PLC operation) in series with any other emergency stop conditions. This would cause the MCR circuit to drop the PLC I/O power in case of a PLC failure (memory error, I/O communications error, etc.).



Emergency Power Disconnect

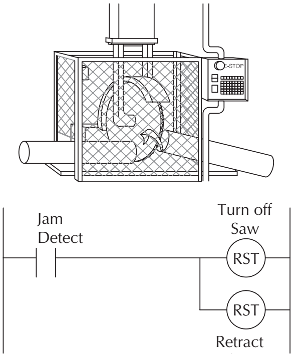
A properly rated emergency power disconnect should be used to power the PLC controlled system as a means of removing the power from the entire control system. It may be necessary to install a capacitor across the disconnect to protect against a condition known as **outrush**. This condition occurs when the output Triacs are turned off by powering off the disconnect, thus causing the energy stored in the inductive loads to seek the shortest distance to ground, which is often through the Triacs.

After an emergency shutdown or any other type of power interruption, there may be requirements that must be met before the PLC control program can be restarted. For example, there may be specific register values that must be established (or maintained from the state prior to the shutdown) before operations can resume. In this case, you may want to use retentive memory locations, or include constants in the control program to insure a known starting point.

Orderly System Shutdown

Ideally, the first level of fault detection is the PLC control program, which can identify machine problems. Certain shutdown sequences should be performed. The types of problems are usually things such as jammed parts, etc., that do not pose a risk of personal injury or equipment damage

WARNING: The control program must not be the only form of protection for any problems that may result in a risk of personal injury or equipment damage.



Class 1, Division 2 Approval

This equipment is suitable for use in Class 1, Zone 2, Division 2, groups A, B, C and D or non-hazardous locations only.

WARNING: Explosion Hazard! Substitution of components may impair suitability for Class 1, Division 2.

WARNING: Explosion Hazard! Do not disconnect equipment unless power has been switched off or the area is known to be non-hazardous.

WARNING: All models used with connector accessories must use R/C (ECBT2) mating plug for all applicable models. All mating plugs shall have suitable ratings for device.

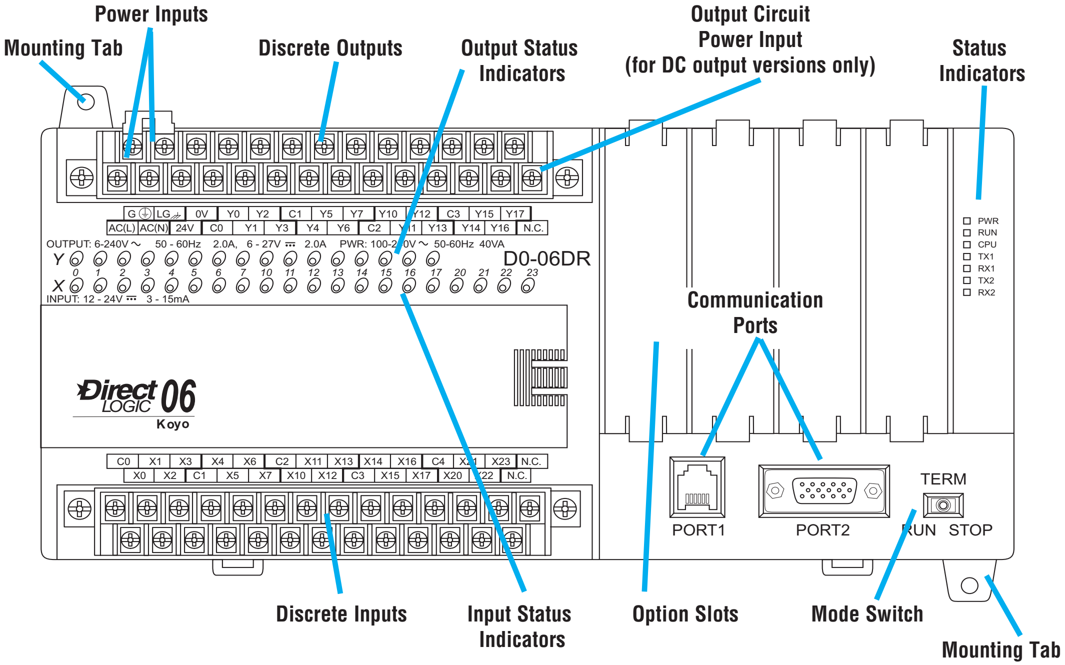
WARNING: This equipment is designed for use in Pollution Degree 2 environments (installed within an enclosure rated at least IP54).

WARNING: Transient suppression must be provided to prevent the rated voltage from being exceeded by 140%.

Orientation to DL06 Front Panel

Most connections, indicators and labels on the DL06 Micro PLCs are located on its front panel. The communication ports are located on front of the PLC, as are the option card slots and the mode selector switch. Please refer to the drawing below.

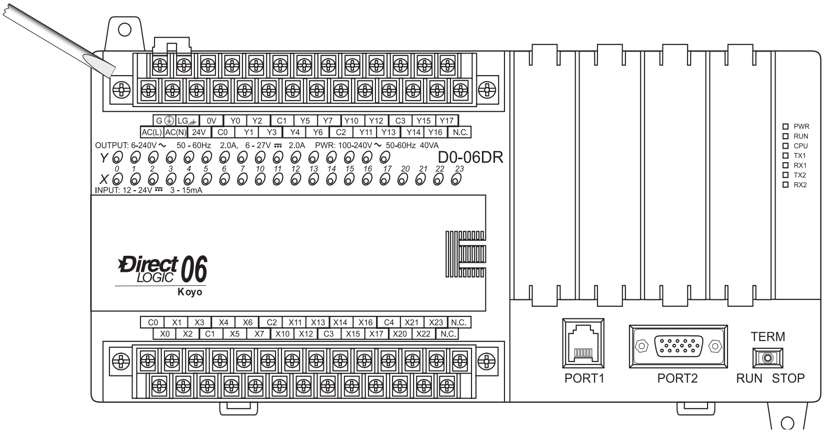
The output and power connector accepts external power and logic and chassis ground connections on the indicated terminals. The remaining terminals are for connecting commons and output connections Y0 through Y17. The sixteen output terminals are numbered in octal, Y0-Y7 and Y10-Y17. On DC output units, the end terminal on the right accepts power for the output stage. The input side connector provides the location for connecting the inputs X0 and X23 and the associated commons.


Power Inputs
Mounting Tab
Discrete Outputs
Output Status Indicators
Output Circuit Power Input (for DC output versions only)
Status Indicators
G LG 0V Y0 Y2 C1 Y5 Y7 Y10 Y12 C3 Y15 Y17
AC(L) AC(N) 24V C0 Y1 Y3 Y4 Y6 C2 Y11 Y13 Y14 Y16 N.C.
OUTPUT: 6-240V ~ 50 - 60Hz 2.0A, 6 - 27V ⎓ 2.0A PWR: 100-240V ~ 50-60Hz 40VA
Y
0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20 21 22 23
X
INPUT: 12 - 24V ⎓ 3 - 15mA
D0-06DR
Direct LOGIC 06
Koyo
PWR
RUN
CPU
TX1
RX1
TX2
RX2
Communication Ports
C0 X1 X3 X4 X6 C2 X11 X13 X14 X16 C4 X21 X23 N.C.
X0 X2 C1 X5 X7 X10 X12 C3 X15 X17 X20 X22 N.C.
PORT1
PORT2
TERM
RUN STOP
Discrete Inputs
Input Status Indicators
Option Slots
Mode Switch
Mounting Tab

WARNING: For some applications, field device power may still be present on the terminal block even though the Micro PLC is turned off. To minimize the risk of electrical shock, check all field device power before you expose or remove either connector.

Terminal Block Removal

The DL06 terminals are divided into two groups. Each group has its own terminal block. The outputs and power wiring are on one block, and the input wiring is on the other. In some instances, it may be desirable to remove the terminal block for easy wiring. The terminal block is designed for easy removal with just a small screwdriver. The drawing below shows the procedure for removing one of the terminal blocks.

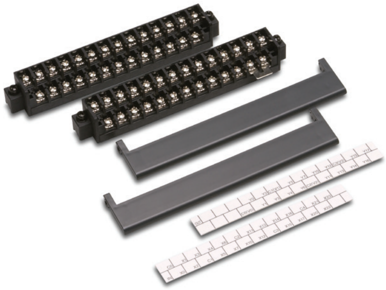


1. Loosen the retention screws on each end of the connector block.

2. From the center of the connector block, pry upward with the screwdriver until the connector is loose.

The terminal blocks on DL06 PLCs have regular (m3 size) screw terminals, which will accept either standard blade-type or #1 Philips screwdriver tips. Use No. 16 to 22 AWG solid/stranded wire. Be careful not to over-tighten; maximum torque is 0.882 to 1.020 N·m (7.806 to 9.028 in·lbs).

Spare terminal blocks are available in an accessory kit. Please refer to part number D0-ACC-2. You can find this and other accessories on our web site.



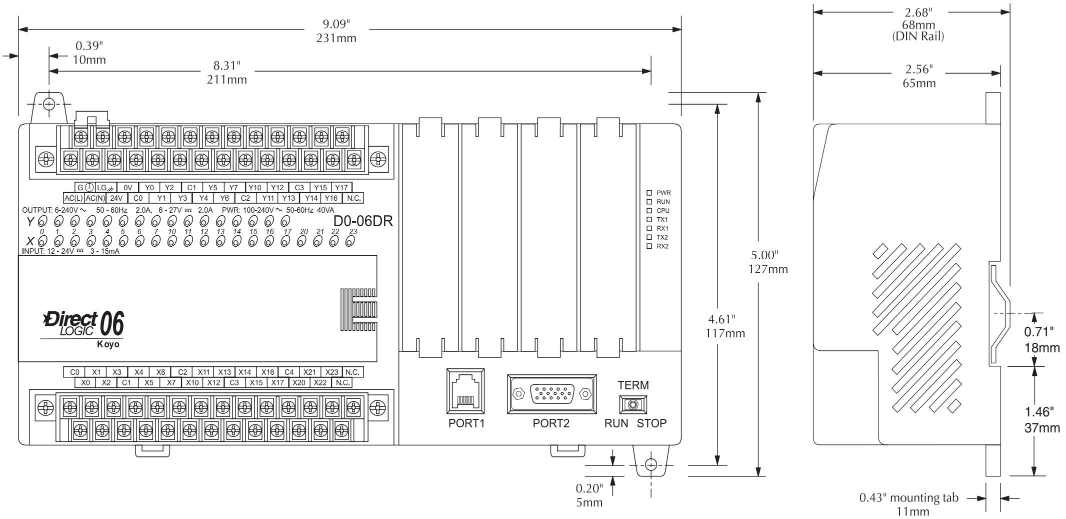
Mounting Guidelines

In addition to the panel layout guidelines, other specifications can affect the installation of a PLC system. Always consider the following:

- Environmental Specifications
- Power Requirements
- Agency Approvals
- Enclosure Selection and Component Dimensions

Unit Dimensions

The following diagram shows the outside dimensions and mounting hole locations for all versions of the DL06. Make sure you follow the installation guidelines to allow proper spacing from other components.


9.09"
231mm
0.39"
10mm
8.31"
211mm
G LG 0V Y0 Y2 C1 Y5 Y7 Y10 Y12 C3 Y15 Y17
AC(L) AC(N) 24V C0 Y1 Y3 Y4 Y6 C2 Y11 Y13 Y14 Y16 N.C.
OUTPUT: 6-240V ~ 50 - 60Hz 2.0A, 6 - 27V ⎓ 2.0A PWR: 100-240V ~ 50-60Hz 40VA
Y
0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20 21 22 23
X
INPUT: 12 - 24V ⎓ 3 - 15mA
D0-06DR
PWR
RUN
CPU
TX1
RX1
TX2
RX2
Direct LOGIC 06
Koyo
C0 X1 X3 X4 X6 C2 X11 X13 X14 X16 C4 X21 X23 N.C.
X0 X2 C1 X5 X7 X10 X12 C3 X15 X17 X20 X22 N.C.
PORT1
PORT2
TERM
RUN STOP
5.00"
127mm
4.61"
117mm
0.20"
5mm
2.68"
68mm
(DIN Rail)
2.56"
65mm
0.71"
18mm
1.46"
37mm
0.43" mounting tab
11mm

Enclosures

Your selection of a proper enclosure is important to ensure safe and proper operation of your DL06 system. Applications of DL06 systems vary and may require additional features. The minimum considerations for enclosures include:

- Conformance to electrical standards
- Protection from the elements in an industrial environment
- Common ground reference
- Maintenance of specified ambient temperature
- Access to equipment
- Security or restricted access
- Sufficient space for proper installation and maintenance of equipment

Panel Layout & Clearances

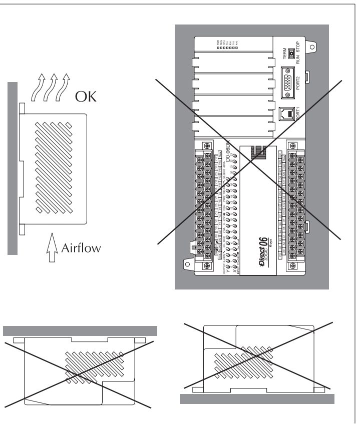
There are many things to consider when designing the panel layout. The following items correspond to the diagram shown. **Note:** there may be additional requirements, depending on your application and use of other components in the cabinet.

1. Mount the PLCs horizontally as shown below to provide proper ventilation. You ***cannot*** mount the DL06 units vertically, upside down, or on a flat horizontal surface. If you place more than one unit in a cabinet, there must be a minimum of 7.2” (183 mm) between the units.

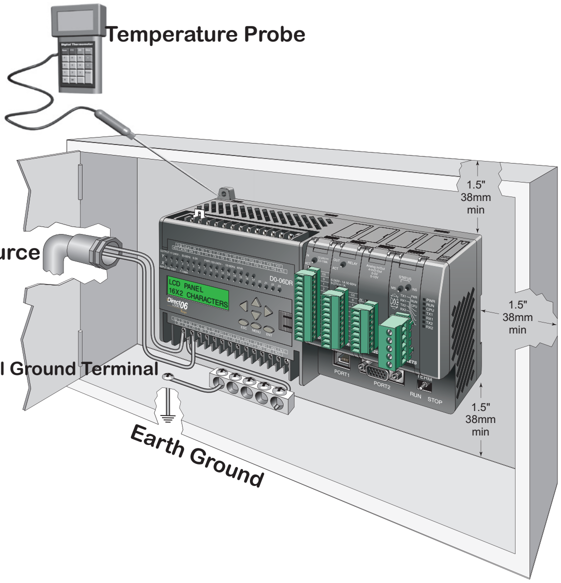
2. Provide a minimum clearance of 1.5” (38 mm) between the unit and all sides of the cabinet. **Remember to allow for any operator panels or other items mounted in the door.**

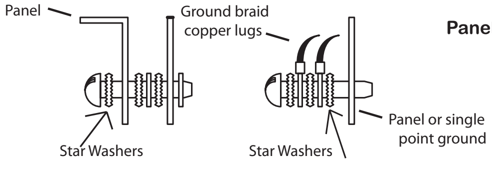
3. There should also be at least 3” (78 mm) of clearance between the unit and any wiring ducts that run parallel to the terminals.

4. The ground terminal on the DL06 base must be connected to a single point ground. Use copper stranded wire to achieve a low impedance. Copper eye lugs should be crimped and soldered to the ends of the stranded wire to ensure good surface contact.



NOTE*: There is a minimum cleara requirement of 1.5” (38 mm) between the panel door (or any devices mounted in the panel doo and the nearest DL06 component.*





5. There must be a single point ground (i.e., copper bus bar) for all devices in the panel requiring an earth ground return. The single point of ground must be connected to the panel ground termination. The panel ground termination must be connected to earth ground. Minimum wire sizes, color coding, and general safety practices should comply with appropriate electrical codes and standards for your area.

6. A good common ground reference (Earth ground) is essential for proper operation of the DL06. One side of all control and power circuits and the ground lead on flexible shielded cable must be properly connected to Earth ground. There are several methods of providing an adequate common ground reference, including:

a) Installing a ground rod as close to the panel as possible
b) Connection to incoming power system ground

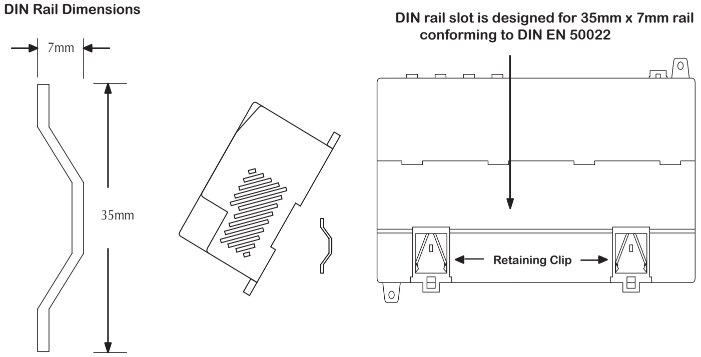
7. Evaluate any installations where the ambient temperature may approach the lower or upper limits of the specifications. If you suspect the ambient temperature will not be within the operating specification for the DL06 system, measures such as installing a cooling/heating source must be taken to get the ambient temperature within the range of specifications.

8. The DL06 systems are designed to be powered by 95–240 VAC or 12–24 VDC normally available throughout an industrial environment. Electrical power in some areas where the PLCs are installed is not always stable and storms can cause power surges. Due to this, power line filters are recommended for protecting the DL06 PLCs from power surges and EMI/RFI noise. The Automationdirect power line filter, for use with 120VAC and 240VAC, 1–5 Amps, is an excellent choice (locate at automationdirect.com); however, you can use a filter of your choice. These units install easily between the power source and the PLC.

NOTE: *If you are using other components in your system, make sure you refer to the appropriate manual to determine how those units can affect mounting dimensions.*

Using Mounting Rails

DL06 Micro PLCs can be secured to a panel by using mounting rails. We recommend rails that conform to DIN EN standard 50022. They are approximately 35 mm high, with a depth of 7 mm. If you mount the Micro PLC on a rail, do consider using end brackets on each side of the PLC. The end bracket helps keep the PLC from sliding horizontally along the rail, reducing the possibility of accidentally pulling the wiring loose. On the bottom of the PLC are two small retaining clips. To secure the PLC to a DIN rail, place it onto the rail and gently push up on the clips to lock it onto the rail. To remove the PLC, pull down on the retaining clips, lift up on the PLC slightly, then pull it away from the rail.


DIN Rail Dimensions
7mm
35mm
DIN rail slot is designed for 35mm x 7mm rail conforming to DIN EN 50022
Retaining Clip

NOTE: *Refer to our catalog or web site for a complete listing of* ***DINnector*** *connection systems.*

Environmental Specifications

The following table lists the environmental specifications that generally apply to DL06 Micro PLCs. The ranges that vary for the Handheld Programmer are noted at the bottom of this chart. Certain output circuit types may have derating curves, depending on the ambient temperature and the number of outputs ON. Please refer to the appropriate section in this chapter pertaining to your particular DL06 PLC.

Environmental Specifications	
Specification	**Rating**
Storage temperature	–4°F to 158°F (–20°C to 70°C)
Ambient operating temperature*	32°F to 131°F (0°C to 55°C)
Ambient humidity**	5% – 95% relative humidity (non–condensing)
Vibration resistance	MIL STD 810C, Method 514.2
Shock resistance	MIL STD 810C, Method 516.2
Noise immunity	NEMA (ICS3–304)
Atmosphere	No corrosive gases
Agency approvals	UL, CE (C1D2), FCC class A

*** Operating temperature for the Handheld Programmer and the DV–1000 is 32° to 122°F (0° to 50°C) Storage temperature for the Handheld Programmer and the DV–1000 is –4° to 158°F (–20° to 70°C).**
****Equipment will operate down to 5% relative humidity; however, static electricity problems occur much more frequently at low humidity levels (below 30%). Make sure you take adequate precautions when you touch the equipment. Consider using ground straps, anti-static floor coverings, etc. if you use the equipment in low-humidity environments.**

Agency Approvals

Some applications require agency approvals for particular components. The DL06 Micro PLC agency approvals are listed below:

- UL (Underwriters' Laboratories, Inc.)
- CUL (Canadian Underwriters' Laboratories, Inc.)
- CE (European Economic Union)

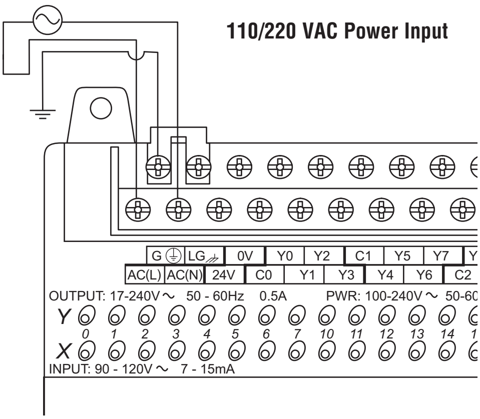
Marine Use

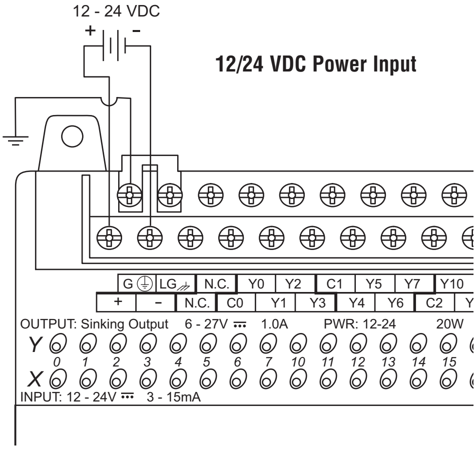
American Bureau of Shipping (ABS) certification requires flame-retarding insulation as per 4-8-3/5.3.6(a). ABS will accept Navy low smoke cables, cable qualified to NEC **Plenum rated** (fire resistant level 4), or other similar flammability resistant rated cables. Use cable specifications for your system that meet a recognized flame retardant standard (i.e., UL, IEEE, etc.), including evidence of cable test certification (i.e., tests certificate, UL file number, etc.).

NOTE: *Wiring must be* ***low smoke*** *per the above paragraph. Teflon coated wire is also recommended.*

Wiring Guidelines

Connect the power input wiring for the DL06. Observe all precautions stated earlier in this manual. When the wiring is complete, close the connector covers. Do not apply power at this time.





WARNING: Once the power wiring is connected, secure the terminal block cover in the closed position. There is a risk of electrical shock if you accidentally touch the connection terminals or power wiring when the cover is open.

External Power Source

The power source must be capable of suppling voltage and current complying with individual Micro PLC specifications, according to the following specifications:

NOTE: *The rating between all internal circuits is BASIC INSULATION ONLY.*

Power Source Specifications		
Item	**DL06 AC Powered Units**	**DL06 DC Powered Units**
Input Voltage Range	110/220 VAC (100–240 VAC/50-60 Hz)	12–24 VDC (10.8–26.4 VDC)
Maximum Inrush Current	13A, 1ms (100–240 VAC) 15A, 1ms (240–264 VAC)	10A
Maximum Power	40VA	20W
Voltage Withstand (dielectric)	1 minute @ 1500VAC between primary, secondary, field ground	
Insulation Resistance	> 10MΩ at 500VDC	

NOTE: *Recommended wire size for field devices is 16–22 AWG solid/stranded. Tighten terminal screws to 7.81 lb·in (0.882 N·m) to 9.03 lb·in (1.02 N·m).*

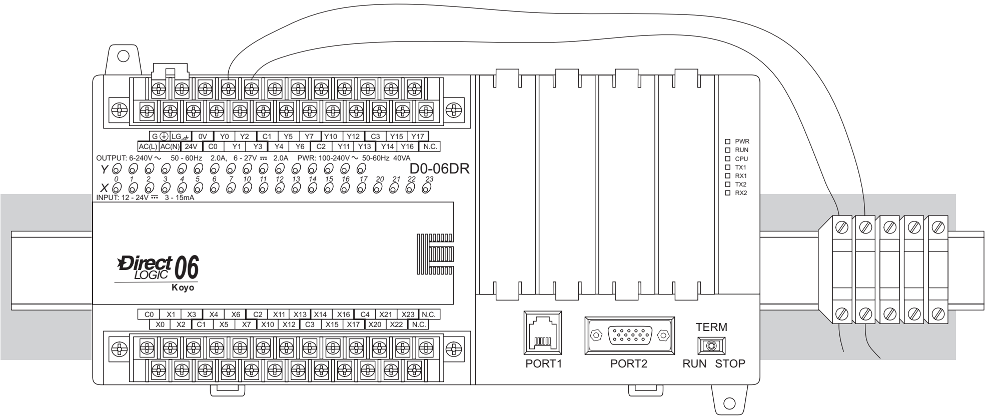
Planning the Wiring Routes

The following guidelines provide general information on how to wire the I/O connections to DL06 Micro PLCs. Refer to the corresponding specification sheet which appears later in this chapter for specific information on wiring a particular PLC .

1. Each terminal connection of the DL06 PLC can accept one 16 AWG wire or two 18 AWG size wires. Do not exceed this recommended capacity.
2. Always use a continuous length of wire. Do not splice wires to attain a needed length.
3. Use the shortest possible wire length.
4. Use wire trays for routing where possible.
5. Avoid running wires near high energy wiring.
6. Avoid running input wiring close to output wiring where possible.
7. To minimize voltage drops when wires must run a long distance, consider using multiple wires for the return line.
8. Avoid running DC wiring in close proximity to AC wiring where possible.
9. Avoid creating sharp bends in the wires.
10. Install the recommended power line filter to reduce power surges and EMI/RFI noise.

Fuse Protection for Input and Output Circuits

Input and Output circuits on DL06 Micro PLCs do not have internal fuses. In order to protect your Micro PLC, we suggest you add external fuses to your I/O wiring. A fast-blow fuse, with a lower current rating than the I/O bank's common current rating, can be wired to each common. Or, a fuse with a rating of slightly less than the maximum current per output point can be added to each output. Refer to the Micro PLC specification sheets further in this chapter to find the maximum current per output point or per output common. Adding the external fuse does not guarantee the prevention of Micro PLC damage, but it will provide added protection.



I/O Point Numbering

All DL06 Micro PLCs have a fixed I/O configuration. It follows the same octal numbering system used on other *Direct*Logic family PLCs, starting at X0 and Y0. The letter X is always used to indicate inputs and the letter Y is always used for outputs.

The I/O numbering always starts at zero and does not include the digits 8 or 9. The addresses are typically assigned in groups of 8 or 16, depending on the number of points in an I/O group. For the DL06, the twenty inputs use reference numbers X0 – X23. The sixteen output points use references Y0 – Y17.

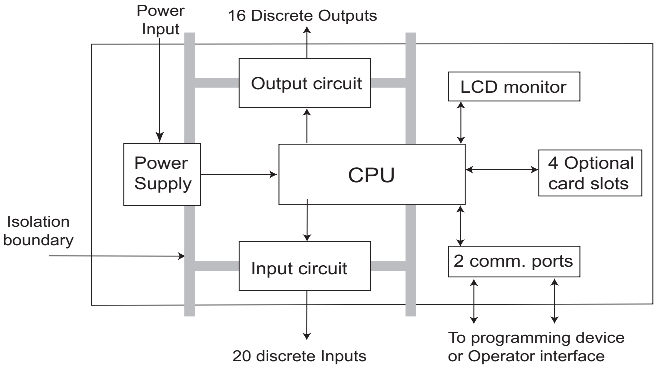
Additional I/O modules can be installed in the four option slots. See the DL05/06 Option Modules User Manual, D0-OPTIONS-M, for a complete selection of modules and how to address them in the DL06. This manual can either be ordered from Automationdirect or downloaded from our website.

System Wiring Strategies

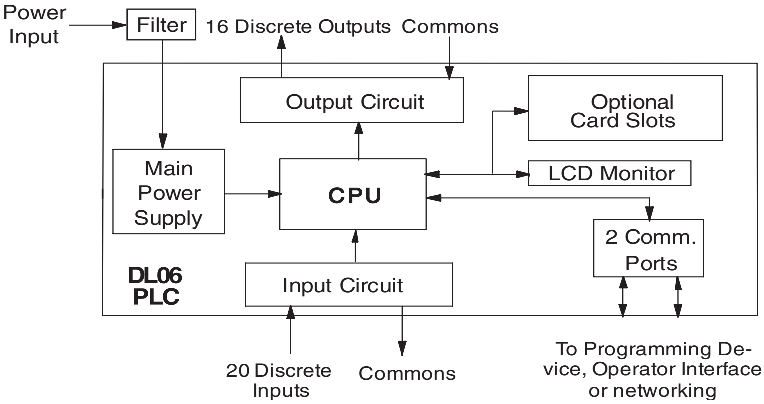
The DL06 Micro PLC is very flexible and will work in many different wiring configurations. By studying this section before actual installation, you can probably find the best wiring strategy for your application. This will help to lower system cost and wiring errors, and avoid safety problems.

PLC Isolation Boundaries

PLC circuitry is divided into three main regions separated by isolation boundaries, shown in the drawing below. Electrical isolation provides safety, so that a fault in one area does not damage another. A power line filter will provide isolation between the power source and the power supply. A transformer in the power supply provides magnetic isolation between the primary and secondary sides. Opto-couplers provide optical isolation in Input and Output circuits. This isolates logic circuitry from the field side, where factory machinery connects. Note that the discrete inputs are isolated from the discrete outputs, because each is isolated from the logic side. Isolation boundaries protect the operator interface (and the operator) from power input faults or field wiring faults. *When wiring a PLC, it is extremely important to avoid making external connections that connect logic side circuits to any other.*

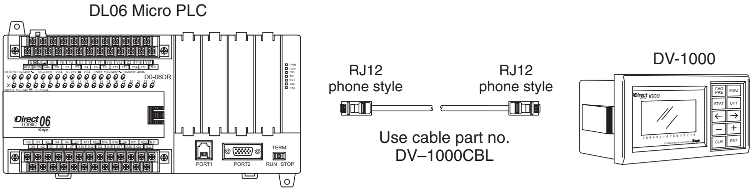


The next figure shows the internal layout of DL06 PLCs, as viewed from the front panel.

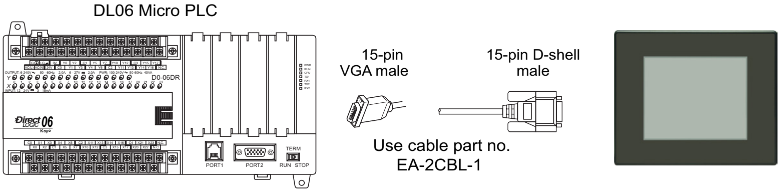


Connecting Operator Interface Devices

Operator interfaces require data and power connections. Some operator interfaces usually require separate AC power. However, other operator interface devices like the popular DV-1000 Data Access Unit may be powered directly from the DL06 Micro PLC. Connect the DV-1000 to communication port 1 on the DL06 Micro PLC using the cable shown below. A single cable contains transmit/receive data wires and +5V power.

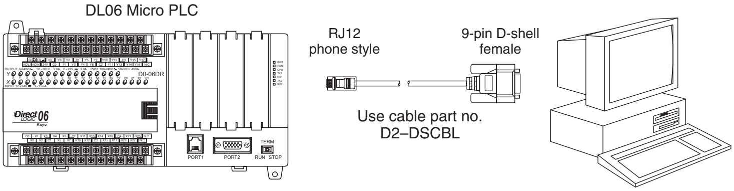


C-more operator interface touch panels use a provided 24 VDC plug-in power supply. Connect the DL06 to the serial connector on the rear of the C-more panel using the cable shown below.

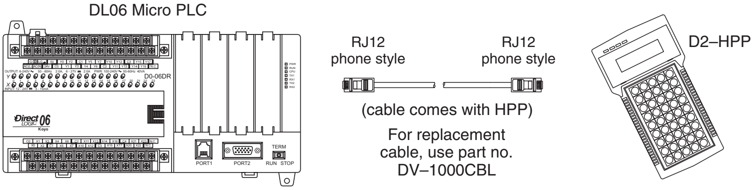


Connecting Programming Devices

DL06 Micro PLCs can be programmed with either a handheld programmer or with *Direct*SOFT on a PC. Connect the DL06 to a PC using the cable shown below.



The D2-HPP Handheld Programmer comes with a communications cable. For a replacement part, use the cable shown below.



Sinking / Sourcing Concepts

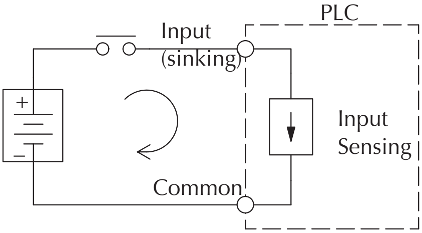
Before going further in our presentation of wiring strategies, we need to introduce the concepts of **sinking** and **sourcing**. These terms apply to typical input or output circuits. It is the goal of this section to make these concepts easy to understand. First, we give the following short definitions, followed by practical applications.

Sinking = Path to supply ground (–)

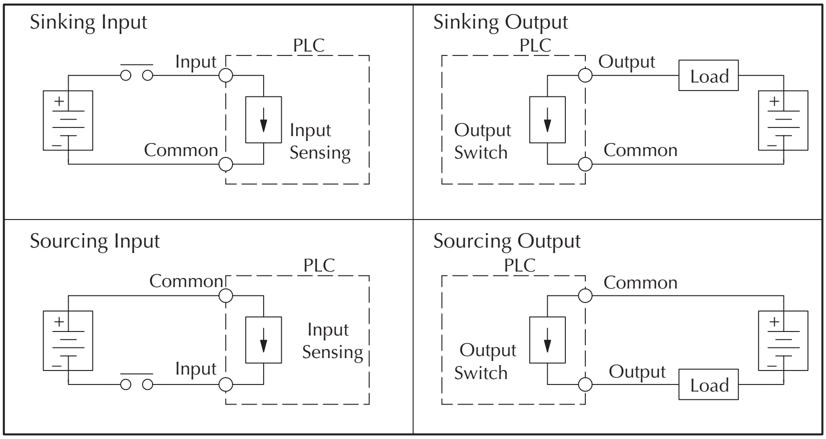
Sourcing = Path to supply source (+)

Notice the reference to (+) and (–) polarities. S*inking and sourcing terminology applies only to DC input and output circuits.* Input and output points that are either sinking or sourcing can conduct current in only one direction. This means it is possible to connect the external supply and field device to the I/O point with current trying to flow in the wrong direction, and the circuit will not operate. However, we can successfully connect the supply and field device every time by understanding **sourcing** and **sinking**.

For example, the figure to the right depicts a **sinkin**g input. To properly connect the external supply, we just have to connect it so the input *provides a path to ground (–)*. So, we start at the PLC input terminal, follow through the input sensing circuit, exit at the common terminal, and connect the supply (–) to the common terminal. By adding the switch, between the supply (+) and the input, we have completed the circuit. Current flows in the direction of the arrow when the switch is closed.

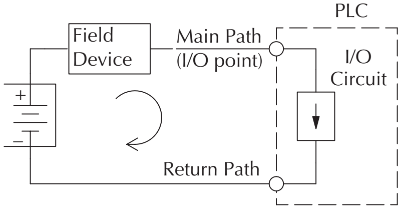


By applying the circuit principle above to the four possible combinations of input/output **sinking/sourcing** types, we have the four circuits as shown below. The DC-powered DL06 Micro PLCs have selectable **sinking** or **sourcing** inputs and either **sinking** or **sourcing** outputs. Any pair of input/output circuits shown below is possible with one of the DL06 models.

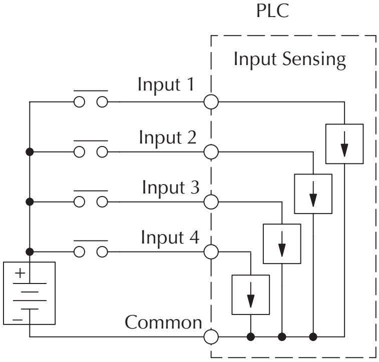


I/O Common Terminal Concepts

In order for a PLC I/O circuit to operate, current must enter at one terminal and exit at another. This means at least two terminals are associated with every I/O point. In the figure to the right, the input or output terminal is the *main path* for the current. One additional terminal must provide the *return path* to the power supply.



Most input or output point groups on PLCs share the return path among two or more I/O points. The figure to the right shows a group (*or bank*) of 4 input points which share a **common** return path. In this way, the four inputs require only five terminals instead of eight.



***NOTE**: In the circuit to the right, the current in the common path is 4 times any channel's input current when all inputs are energized. This is especially important in output circuits, where heavier gauge wire is sometimes necessary on commons.*

Most DL06 input and output circuits are grouped into banks that share a common return path. The best indication of I/O common grouping is on the wiring label. The I/O common groups are separated by a bold line. A thinner line separates the inputs associated with that common. To the right, notice that X0, X1, X2, and X3 share the common terminal C0, located to the left of X1.

C0		X1		X3	
	X0		X2		

The following complete set of labels shows five banks of four inputs and four banks of four outputs. One common is provided for each bank.

	G ⏚		LG		0V		Y0		Y2		C1		Y5		Y7		Y10		Y12		C3		Y15		Y17		
AC(L)		AC(N)		24V		C0		Y1		Y3		Y4		Y6		C2		Y11		Y13		Y14		Y16		N.C.	

C0		X1		X3		X4		X6		C2		X11		X13		X14		X16		C4		X21		X23		N.C.	
	X0		X2		C1		X5		X7		X10		X12		C3		X15		X17		X20		X22		N.C.		

This set of labels is for DC (sinking) output versions such as the D0-06DD1 and D0-06DD1-D. One common is provided for each group of four outputs, and one designated terminal on the output side accepts power for the output stage.

	G ⏚		LG		0V		Y0		Y2		C1		Y5		Y7		Y10		Y12		C3		Y15		Y17		
AC(L)		AC(N)		24V		C0		Y1		Y3		Y4		Y6		C2		Y11		Y13		Y14		Y16		+V	

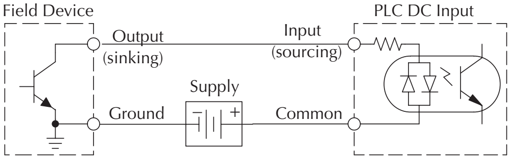
C0		X1		X3		X4		X6		C2		X11		X13		X14		X16		C4		X21		X23		N.C.	
	X0		X2		C1		X5		X7		X10		X12		C3		X15		X17		X20		X22		N.C.		

Connecting DC I/O to Solid State Field Devices

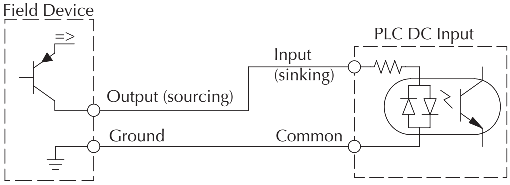
In the previous section on sinking and sourcing concepts, we discussed DC I/O circuits that only allow current to flow one way. This is also true for many of the field devices which have solid-state (transistor) interfaces. In other words, field devices can also be sourcing or sinking. *When connecting two devices in a series DC circuit (as is the case when wiring a field device to a PLC DC input or output), one must be wired as sourcing and the other as sinking.*

Solid State Input Sensors

The DL06's DC inputs are flexible in that they detect current flow in either direction, so they can be wired as either sourcing or sinking. In the following circuit, a field device has an open-collector NPN transistor output. It sinks current from the PLC input point, which sources current. The power supply can be the included auxiliary 24VDC power supply or another supply (+12VDC or +24VDC), as long as the input specifications are met.



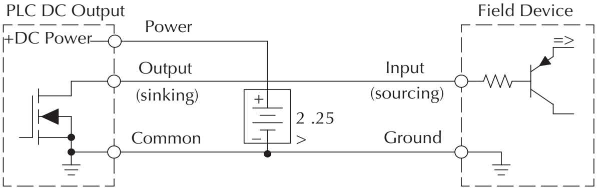
In the next circuit, a field device has an open-emitter PNP transistor output. It sources current to the PLC input point, which sinks the current back to ground. Since the field device is sourcing current, no additional power supply is required between the device and the PLC DC Input.



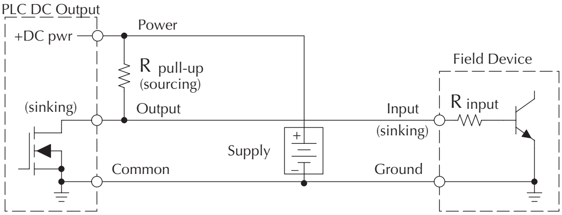
Solid State Output Loads

Sometimes an application requires connecting a PLC output point to a solid state input on a device. This type of connection is usually made to carry a low-level signal, not to send DC power to an actuator.

The DL06 PLC family offers DC outputs that are sinking only or DC outputs that are sourcing. All sixteen outputs have the same electrical common, even though there are four common terminal screws. In the following circuit, the PLC output point sinks current to the output common when energized. It is connected to a sourcing input of a field device input.



In the next example we connect a PLC DC output point to the sinking input of a field device. This is a bit tricky, because both the PLC output and field device input are sinking type. Since the circuit must have one sourcing and one sinking device, we add sourcing capability to the PLC output by using a pull-up resistor. In the circuit below, we connect $R_{pull-up}$ from the output to the DC output circuit power input.



NOTE: *DO NOT attempt to drive a heavy load (>25mA) with this pull-up method.*

NOTE: *Using the pull-up resistor to implement a sourcing output has the effect of inverting the output point logic. In other words, the field device input is energized when the PLC output is OFF, from a ladder logic point-of-view. Your ladder program must comprehend this and generate an inverted output. Or, you may choose to cancel the effect of the inversion elsewhere, such as in the field device.*

It is important to choose the correct value of $R_{pull-up.}$ In order to do so, we need to know the nominal input current to the field device (I_{input}) when the input is energized. If this value is not known, it can be calculated as shown (a typical value is 15 mA). Then use I_{input} and the voltage of the external supply to compute $R_{pull-up}$. Then calculate the power $P_{pull-up}$ (in watts), in order to size $R_{pull-up}$ properly.

$$I_{input} = \frac{V_{input\ (turn–on)}}{R_{input}}$$

$$R_{pull-up} = \frac{V_{supply} - 0.7}{I_{input}} - R_{input} \qquad P_{pull-up} = \frac{{V_{supply}}^2}{R_{pullup}}$$

Relay Output Wiring Methods

The D0–06AR and the D0–06DR models feature relay outputs. Relays are best for the following applications:

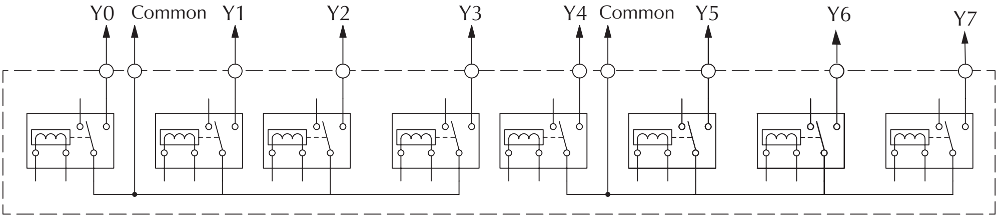
- Loads that require higher currents than the solid-state DL06 outputs can deliver
- Cost-sensitive applications
- Some output channels need isolation from other outputs (such as when some loads require AC while others require DC)

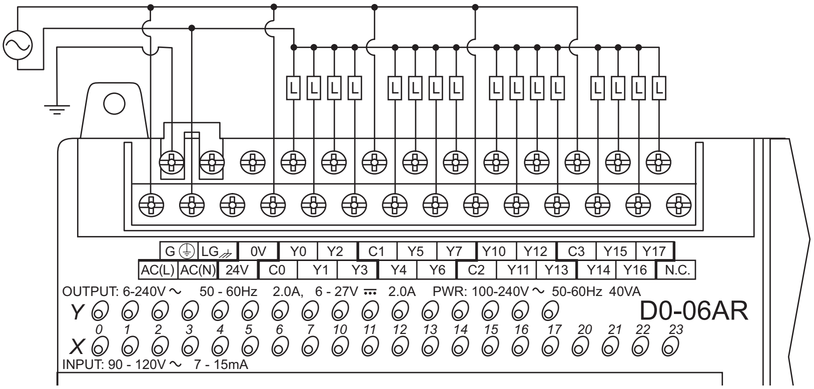
Some applications in which NOT to use relays:

- Loads that require currents under 10mA
- Loads which must be switched at high speed and duty cycle

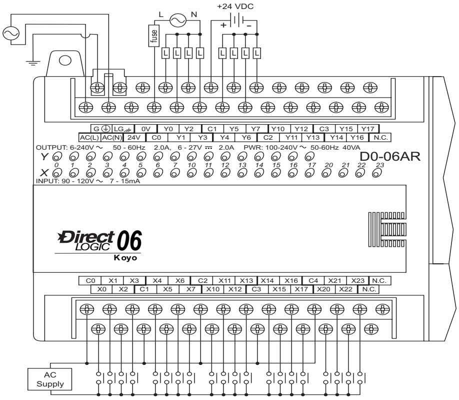
This section presents various ways to wire relay outputs to the loads. The relay output DL06s have sixteen normally-open SPST relays available. They are organized with four relays per common. The figure below shows the relays and the internal wiring of the PLC. Note that each group is isolated from the other group of outputs.

In the circuit below, all loads use the same AC power supply which powers the DL06 PLC. In this example, all commons are connected together.





In the circuit on the following page, loads for Y0 – Y3 use the same AC power supply which powers the DL06 PLC. Loads for Y4 – Y7 use a separate DC supply. In this example, the commons are separated according to which supply powers the associated load.



Relay Outputs – Transient Suppression for Inductive Loads in a Control System

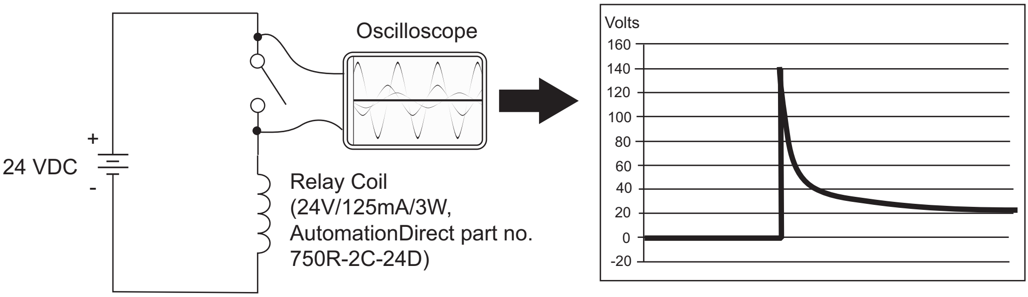
The following pages are intended to give a quick overview of the negative effects of transient voltages on a control system and provide some simple advice on how to effectively minimize them. The need for transient suppression is often not apparent to the newcomers in the automation world. Many mysterious errors that can afflict an installation can be traced back to a lack of transient suppression.

What is a Transient Voltage and Why is it Bad?

Inductive loads (devices with a coil) generate transient voltages as they transition from being energized to being de-energized. If not suppressed, the transient can be many times greater than the voltage applied to the coil. These transient voltages can damage PLC outputs or other electronic devices connected to the circuit, and cause unreliable operation of other electronics in the general area. Transients must be managed with suppressors for long component life and reliable operation of the control system.

This example shows a simple circuit with a small 24V/125mA /3W relay. As you can see, when the switch is opened, thereby de-energizing the coil, the transient voltage generated across the switch contacts peaks at 140V.

Example: Circuit with no Suppression



In the same circuit on the previous page, replacing the relay with a larger 24V/290mA/7W relay will generate a transient voltage exceeding 800V (not shown). Transient voltages like this can cause many problems, including:

- Relay contacts driving the coil may experience arcing, which can pit the contacts and reduce the relay's lifespan.
- Solid state (transistor) outputs driving the coil can be damaged if the transient voltage exceeds the transistor's ratings. In extreme cases, complete failure of the output can occur the very first time a coil is de-energized.
- Input circuits, which might be connected to monitor the coil or the output driver, can also be damaged by the transient voltage.

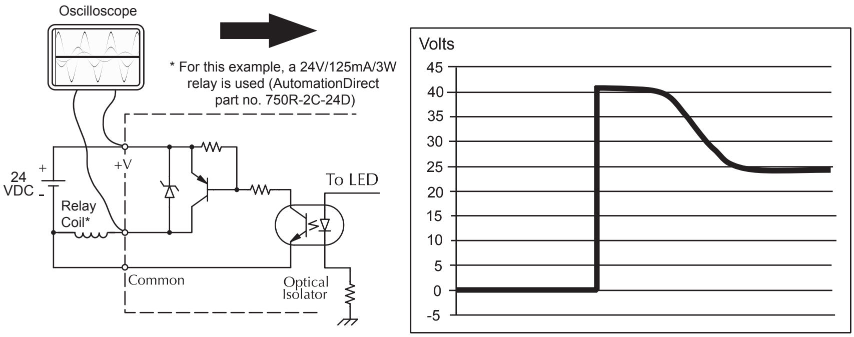
A very destructive side-effect of the arcing across relay contacts is the electromagnetic interference (EMI) it can cause. This occurs because the arcing causes a current surge, which releases RF energy. The entire length of wire between the relay contacts, the coil, and the power source carries the current surge and becomes an antenna that radiates the RF energy. It will readily couple into parallel wiring and may disrupt the PLC and other electronics in the area. This EMI can make an otherwise stable control system behave unpredictably at times.

PLC's Integrated Transient Suppressors

Although the PLC's outputs typically have integrated suppressors to protect against transients, they are not capable of handling them all. It is usually necessary to have some additional transient suppression for an inductive load.

Here is another example using the same 24V / 125mA / 3W relay used earlier. This example measures the PNP transistor output of a D0-06DD2 PLC, which incorporates an integrated Zener diode for transient suppression. Instead of the 140V peak in the first example, the transient voltage here is limited to about 40V by the Zener diode. While the PLC will probably tolerate repeated transients in this range for some time, the 40V is still beyond the module's peak output voltage rating of 30V.

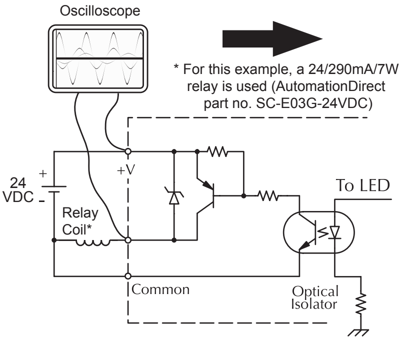
Example: Small Inductive Load with Only Integrated Suppression



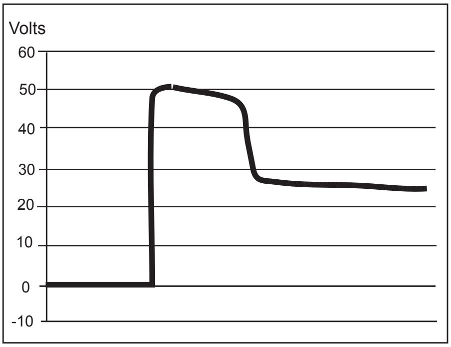
* For this example, a 24V/125mA/3W relay is used (AutomationDirect part no. 750R-2C-24D)

The next example uses the same circuit as above, but with a larger 24V / 290mA / 7W relay, thereby creating a larger inductive load. As you can see, the transient voltage generated is much worse, peaking at over 50V. Driving an inductive load of this size without additional transient suppression is very likely to permanently damage the PLC output.

Example: Larger Inductive Load with Only Integrated Suppression



* For this example, a 24/290mA/7W relay is used (AutomationDirect part no. SC-E03G-24VDC)



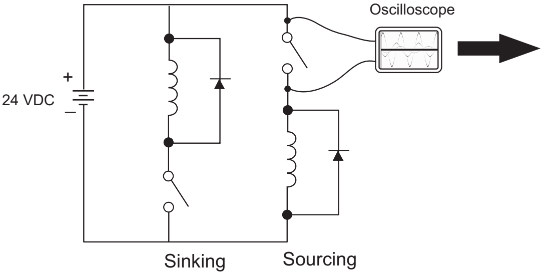
Additional transient suppression should be used in both these examples. If you are unable to measure the transients generated by the connected loads of your control system, using additional transient suppression on all inductive loads would be the safest practice.

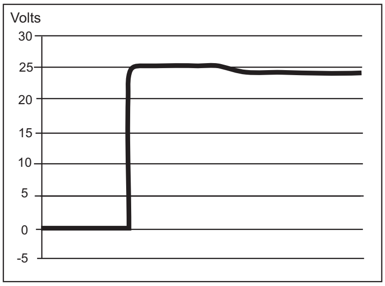
Types of Additional Transient Protection

DC Coils:

The most effective protection against transients from a DC coil is a flyback diode. A flyback diode can reduce the transient to roughly 1V over the supply voltage, as shown in this example.

DC Flyback Circuit

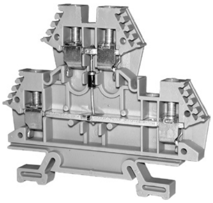




Many AutomationDirect socketed relays and motor starters have add-on flyback diodes that plug or screw into the base, such as the AD-ASMD-250 protection diode module and 784-4C-SKT-1 socket module shown below. If an add-on flyback diode is not available for your inductive load, an easy way to add one is to use AutomationDirect's DN-D10DR-A diode terminal block, a 600 VDC power diode mounted in a slim DIN rail housing.

AD-ASMD-250
Protection Diode Module

784-4C-SKT-1
Relay Socket



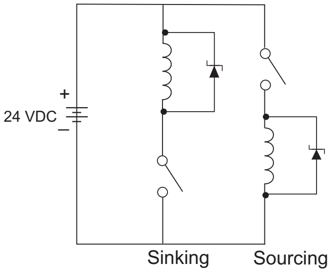
DN-D10DR-A
Diode Terminal Block

Two more common options for DC coils are Metal Oxide Varistors (MOV) or TVS diodes. These devices should be connected across the driver (PLC output) for best protection as shown below. The optimum voltage rating for the suppressor is the lowest rated voltage available that will NOT conduct at the supply voltage, while allowing a safe margin.

AutomationDirect's ZL-TSD8-24 transorb module is a good choice for 24VDC circuits. It is a bank of 8 uni-directional 30V TVS diodes. Since they are uni-directional, be sure to observe the polarity during installation. MOVs or bi-directional TVS diodes would install at the same location, but have no polarity concerns.

ZL-TSD8-24
Transorb Module

DC MOV or TVS Diode Circuit



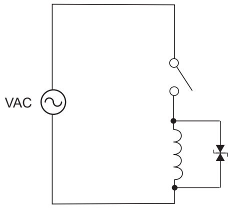
AC Coils:

Two options for AC coils are MOVs or bi-directional TVS diodes. These devices are most effective at protecting the driver from a transient voltage when connected across the driver (PLC output) but are also commonly connected across the coil. The optimum voltage rating for the suppressor is the lowest rated voltage available that will NOT conduct at the supply voltage, while allowing a safe margin.

AutomationDirect's ZL-TSD8-120 transorb module is a good choice for 120VAC circuits. It is a bank of eight bi-directional 180V TVS diodes.

ZL-TSD8-120
Transorb Module

AC MOV or Bi-Directional Diode Circuit



NOTE: *Manufacturers of devices with coils frequently offer MOV or TVS diode suppressors as an add-on option which mount conveniently across the coil. Before using them, carefully check the suppressor ratings. Just because the suppressor is made specifically for that part does not mean it will reduce the transient voltages to an acceptable level.*

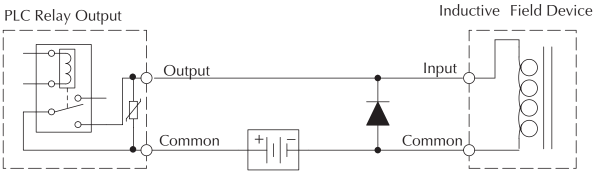
For example, a MOV or TVS diode rated for use on 24–48 VDC coils would need to have a high enough voltage rating to NOT conduct at 48V. That suppressor might typically start conducting at roughly 60VDC. If it were mounted across a 24V coil, transients of roughly 84V (if sinking output) or -60V (if sourcing output) could reach the PLC output. Many semiconductor PLC outputs cannot tolerate such levels.

Prolonging Relay Contact Life

Relay contacts wear according to the amount of relay switching, amount of spark created at the time of open or closure, and presence of airborne contaminants. There are some steps you can take to help prolong the life of relay contacts, such as switching the relay on or off only when it is necessary, and if possible, switching the load on or off at a time when it will draw the least current. Also, take measures to suppress inductive voltage spikes from inductive DC loads such as contactors and solenoids.

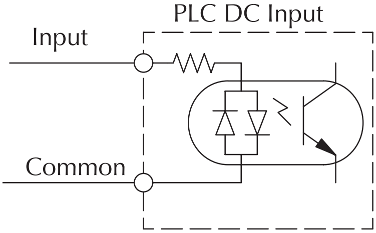
For inductive loads in DC circuits we recommend using a suppression diode as shown in the following diagram (DO NOT use this circuit with an AC power supply). When the load is energized the diode is reverse-biased (high impedance). When the load is turned off, energy stored in its coil is released in the form of a negative-going voltage spike. At this moment the diode is forward-biased (low impedance) and shunts the energy to ground. This protects the relay contacts from the high voltage arc that would occur just as the contacts are opening.

Place the diode as close to the inductive field device as possible. Use a diode with a peak inverse voltage rating (PIV) at least 100 PIV, 3A forward current or larger. Use a fast-recovery type (such as Schottky type). DO NOT use a small-signal diode such as 1N914, 1N941, etc. Be sure the diode is in the circuit correctly before operation. If installed backwards, it short-circuits the supply when the relay energizes.

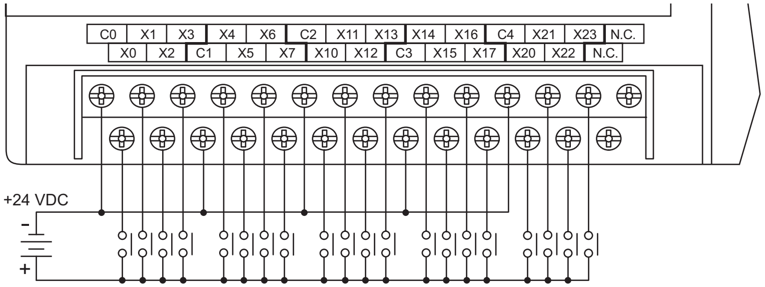


DC Input Wiring Methods

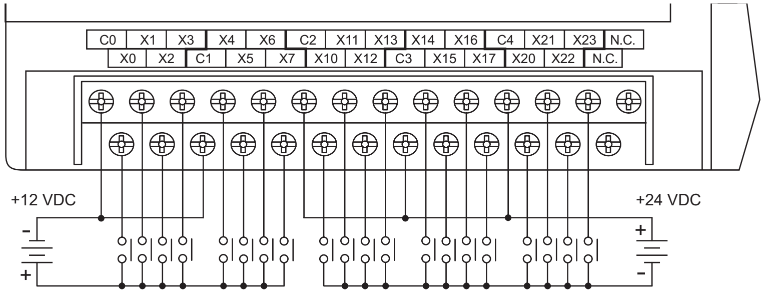
DL06 Micro PLCs with DC inputs are particularly flexible because they can be wired as either sinking or sourcing. The dual diodes (shown to the right) allow 10.8–26.4 VDC. The target applications are +12VDC and +24VDC. You can actually wire each group of inputs associated common group of inputs as DC sinking and the other half as DC sourcing. Inputs grouped by a common must be all sinking or all sourcing.



In the first and simplest example below, all commons are connected together and all inputs are sinking.



In the next example, the first eight inputs are sinking, and the last twelve are sourcing.

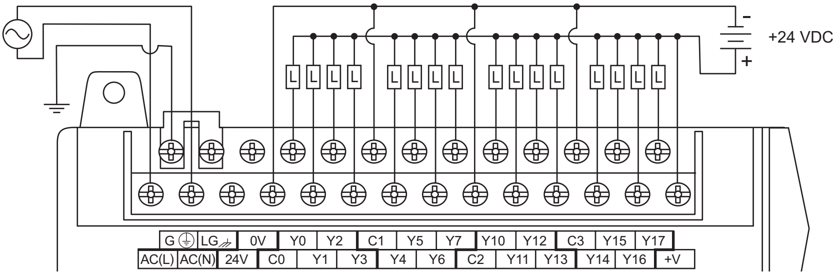


DC Output Wiring Methods

DL06 DC output circuits are high-performance transistor switches with low on-resistance and fast switching times. Please note the following characteristics which are unique to the DC output type:

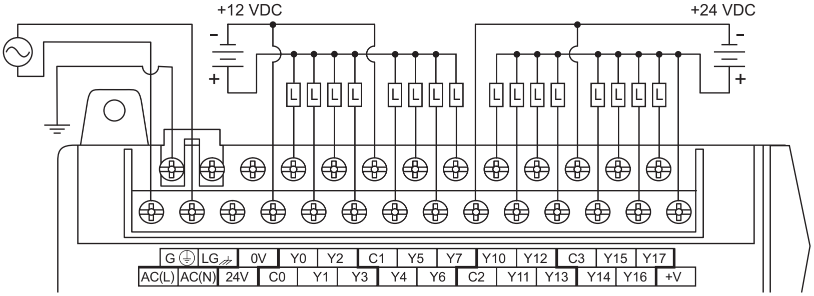
- There is only one electrical common for all sixteen outputs. All sixteen outputs belong to one bank.
- The output switches are current-sinking only or current sourcing only. ***Refer to the detailed specifications in this manual to determine which type output is present on a particular model.***
- The output circuit inside the PLC requires external power. The supply (–) must be connected to a common terminal, and the supply (+) connects the right-most terminal on the upper connector (+V).

In the example below, all sixteen outputs share a common supply.



In the next example below, the outputs have **split** supplies. The first eight outputs are using a +12 VDC supply, and the last eight are using a +24VDC supply. However, you can split the outputs among any number of supplies, as long as:

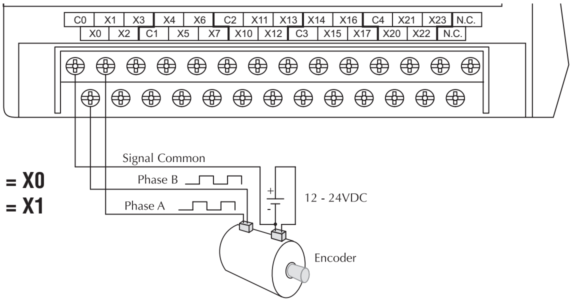
- all supply voltages are within the specified range
- all output points are wired as sinking
- all source (–) terminals are connected together



Warning: The maximum output current from the Auxiliary 24VDC power depends on the I/O configuration. Refer to Chapter 4, page 4-6, to determine how much current can be drawn from the Auxiliary 24VDC power for your particular I/O configuration.

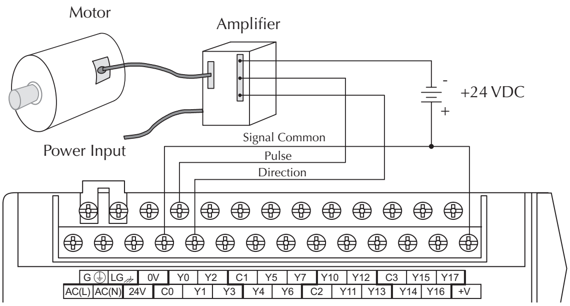
High-Speed I/O Wiring Methods

DL06 versions with DC type input or output points contain a dedicated High-Speed I/O circuit (HSIO). The circuit configuration is programmable, and it processes specific I/O points independently from the CPU scan. Appendix E discusses the programming options for HSIO. While the HSIO circuit has six modes, we show wiring diagrams for two of the most popular modes in this chapter. The high-speed input interfaces to points X0–X3. Properly configured, the DL06 can count quadrature pulses at up to 7kHz from an incremental encoder as shown below.



Phase A = X0
Phase B = X1

NOTE: *Do not use this drawing to wire your device. This is a general example and is not specific to any PLC model, stepper or encoder. Always refer to the device documentation for proper wiring connections.*



DL06 versions with DC type output points can use the High Speed I/O Pulse Output feature. It can generate high-speed pulses at up to 10 kHz for specialized control such as stepper motor / intelligent drive systems. Output Y0 and Y1 can generate pulse and direction signals, or it can generate CCW and CW pulse signals respectively. See Appendix E on high-speed input and pulse output options.

NOTE: *Do not use this drawing to wire your device. This is a general example and is not specific to any PLC model, stepper or encoder. Always refer to the device documentation for proper wiring connections.*

Wiring Diagrams and Specifications

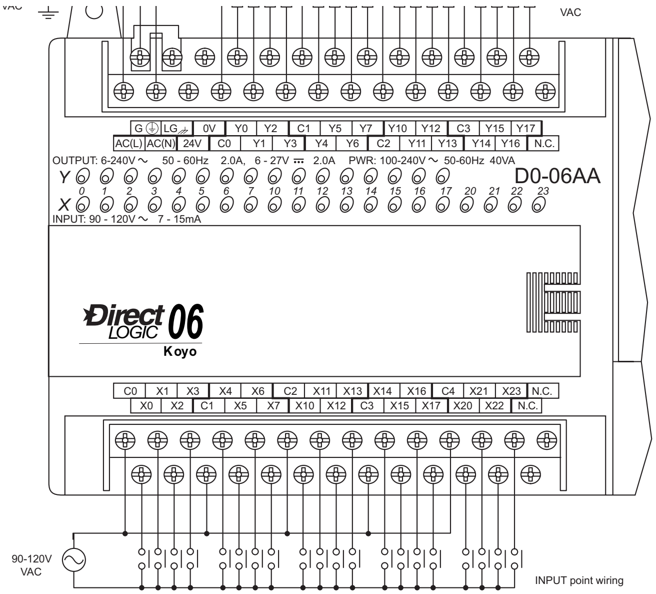
The remainder of this chapter provides detailed technical information for the DL06 PLCs. A basic wiring diagram, equivalent I/O circuits, and specification tables are laid out for each PLC.

D0–06AA I/O Wiring Diagram

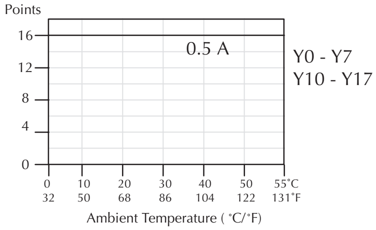
The D0–06AA PLC has twenty AC inputs and sixteen AC outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals as shown.

Inputs are organized into five banks of four. Each bank has an isolated common terminal. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent input circuit shows one channel of a typical bank.

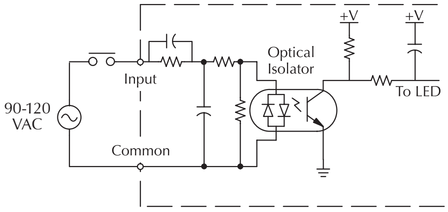
Outputs are organized into four banks of four triac switches. Each bank has a common terminal. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank.



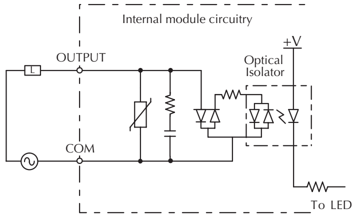
Derating Chart for AC Outputs



Equivalent Input Circuit



Equivalent Output Circuit



D0-06AA General Specifications	
External Power Requirements	100–240 VAC/50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default) 8 data bits, 1 stop bit odd parity	K–Sequence (Slave),DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence / print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

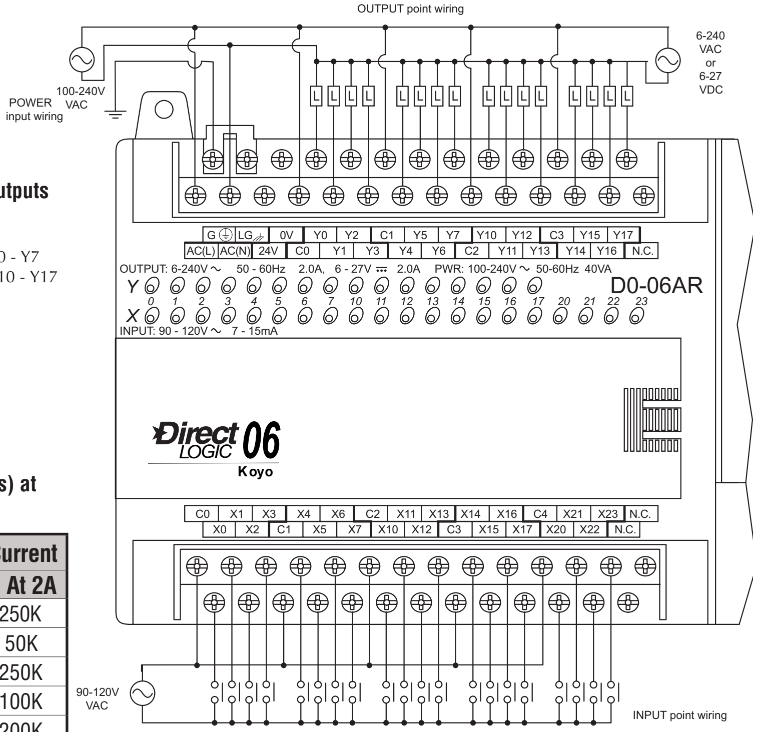
AC Input Specifications	
Input Voltage Range (Min. - Max.)	80–132 VAC, 47–63 Hz
Operating Voltage Range	90–120 VAC, 47–63 Hz
Input Current	8mA @100VAC at 50Hz 10mA @100VAC at 60Hz
Max. Input Current	12mA @132VAC at 50 Hz 15mA @132VAC at 60 Hz
Input Impedance	14KΩ @50 Hz, 12KΩ @60Hz
ON Current/Voltage	> 6mA @ 75VAC
OFF Current/Voltage	< 2mA @ 20VAC
OFF to ON Response	< 40ms
ON to OFF Response	< 40ms
Status Indicators	Logic Side
Commons	4 channels / common x 5 banks (isolated)

AC Output Specifications	
Output Voltage Range (Min. - Max.)	15–264 VAC, 47–63 Hz
Operating Voltage	17–240 VAC, 47–63 Hz
On Voltage Drop	1.5 VAC (>50mA) 4.0 VAC (<50mA)
Max Current	0.5 A / point, 1.5 A / common
Max leakage current	<4mA @ 264VAC
Max inrush current	10A for 10ms
Minimum Load	10mA
OFF to ON Response	1ms
ON to OFF Response	1 ms +1/2 cycle
Status Indicators	Logic Side
Commons	4 channels / common x 4 banks (isolated)
Fuses	None (external recommended)

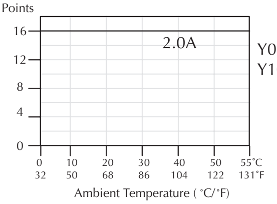
D0–06AR I/O Wiring Diagram

The D0–06AR PLC has twenty AC inputs and sixteen relay contact outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals at the left as shown.

The twenty AC input channels use terminals on the bottom of the connector. Inputs are organized into five banks of four. Each bank has a common terminal. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent input circuit shows one channel of a typical bank.


OUTPUT point wiring
6-240
VAC
or
6-27
VDC
POWER
input wiring
100-240V
VAC
L L L L L L L L L L L L L L L L
G ⏚ LG | 0V | Y0 | Y2 | C1 | Y5 | Y7 | Y10 | Y12 | C3 | Y15 | Y17
AC(L) | AC(N) | 24V | C0 | Y1 | Y3 | Y4 | Y6 | C2 | Y11 | Y13 | Y14 | Y16 | N.C.
OUTPUT: 6-240V ~ 50 - 60Hz 2.0A, 6 - 27V ⎓ 2.0A PWR: 100-240V ~ 50-60Hz 40VA
Y 0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20 21 22 23
X
INPUT: 90 - 120V ~ 7 - 15mA
D0-06AR
Direct LOGIC 06
Koyo
C0 | X1 | X3 | X4 | X6 | C2 | X11 | X13 | X14 | X16 | C4 | X21 | X23 | N.C.
X0 | X2 | C1 | X5 | X7 | X10 | X12 | C3 | X15 | X17 | X20 | X22 | N.C.
90-120V
VAC
INPUT point wiring

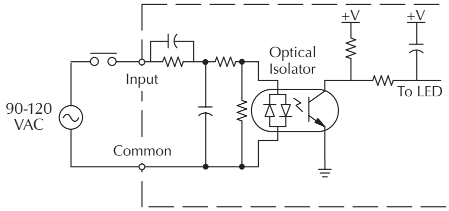
Derating Chart for Relay Outputs



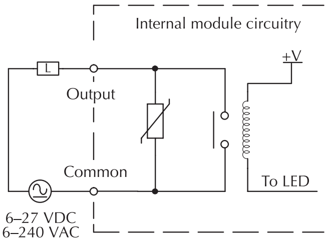
Typical Relay Life (Operations) at Room Temperature

Voltage & Load Type	Load Current	
	At 1A	At 2A
24VDC Resistive	500K	250K
24VDC Inductive	100K	50K
110VAC Resistive	500K	250K
110VAC Inductive	200K	100K
220VAC Resistive	350K	200K
220VAC Inductive	100K	50K

Equivalent Input Circuit



Equivalent Output Circuit



The sixteen relay output channels use terminals on the right side top connector. Outputs are organized into four banks of four normally-open relay contacts. Each bank has a common terminal. The wiring example on the last page shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank. The relay contacts can switch AC or DC voltages.

D0-06AR General Specifications	
External Power Requirements	100– 240 VAC/ 50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence / print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

AC Input Specifications X0-X23	
Input Voltage Range (Min. - Max.)	80–132 VAC, 47–63 Hz
Operating Voltage Range	90–120 VAC, 47–63 Hz
Input Current	8mA @ 100VAC at 50Hz 10mA @ 100VAC at 60Hz
Max. Input Current	12mA @ 132VAC at 50Hz 15mA @ 132VAC at 60Hz
Input Impedance	14KΩ @50Hz, 12KΩ @60Hz
ON Current/Voltage	>6mA @ 75VAC
OFF Current/Voltage	<2mA @ 20VAC
OFF to ON Response	< 40ms
ON to OFF Response	< 40ms
Status Indicators	Logic Side
Commons	4 channels / common x 5 banks (isolated)

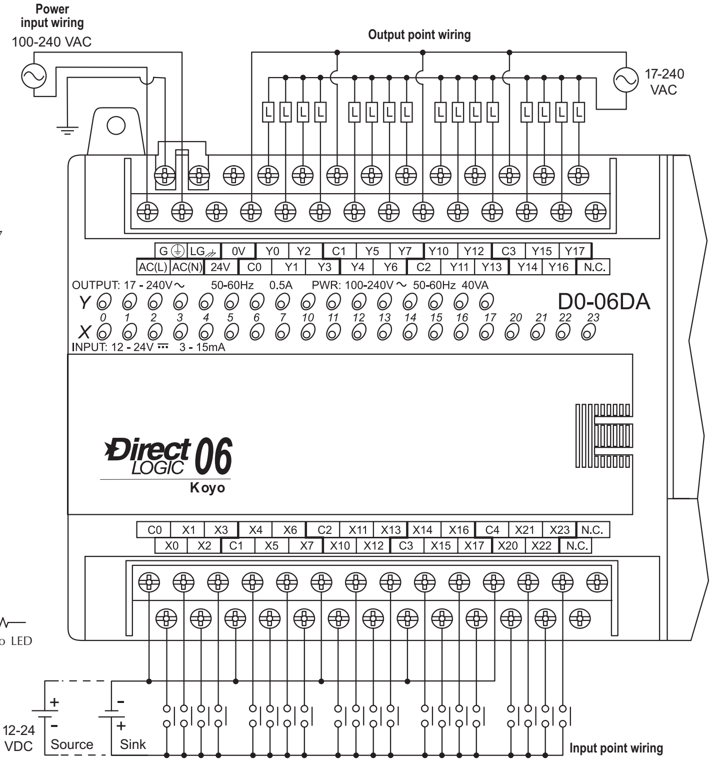
Relay Output Specifications Y0-Y17	
Output Voltage Range	(Min. – Max.) 5–264 VAC (47–63 Hz), 5–30 VDC
Operating Voltage Range	6–240 VAC (47–63 Hz), 6–27 VDC
Output Current	2A / point, 6A / common
Max. leakage current	0.1 mA @ 264VAC
Smallest Recommended Load	5mA @ 5VDC
OFF to ON Response	< 15ms
ON to OFF Response	< 10ms
Status Indicators	Logic Side
Commons	4 channels / common x 4 banks (isolated)
Fuses	None (external recommended)

D0–06DA I/O Wiring Diagram

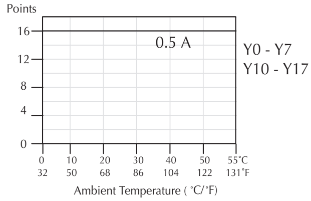
The D0–06DA PLC has twenty DC inputs and sixteen AC outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals as shown.

Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as sinking or sourcing. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent circuit for standard inputs is shown below, and the high-speed input circuit is shown to the left.

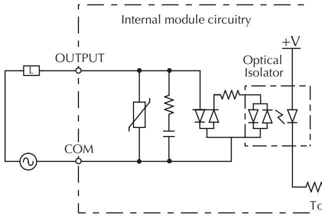
Outputs are organized into four banks of four triac switches. Each bank has a common terminal. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank.


Power input wiring
100-240 VAC
Output point wiring
17-240 VAC
L L L L L L L L L L L L L L L L
G | LG | 0V | Y0 | Y2 | C1 | Y5 | Y7 | Y10 | Y12 | C3 | Y15 | Y17
AC(L) | AC(N) | 24V | C0 | Y1 | Y3 | Y4 | Y6 | C2 | Y11 | Y13 | Y14 | Y16 | N.C.
OUTPUT: 17 - 240V~ 50-60Hz 0.5A PWR: 100-240V~ 50-60Hz 40VA
Y 0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17
X 0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20 21 22 23
D0-06DA
INPUT: 12 - 24V ⎓ 3 - 15mA
Direct LOGIC 06
Koyo
C0 | X1 | X3 | X4 | X6 | C2 | X11 | X13 | X14 | X16 | C4 | X21 | X23 | N.C.
X0 | X2 | C1 | X5 | X7 | X10 | X12 | C3 | X15 | X17 | X20 | X22 | N.C.
+ - 12-24 VDC Source
- + Sink
Input point wiring

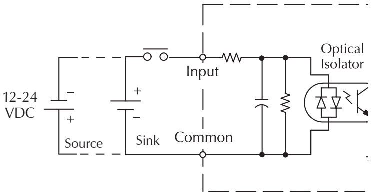
Derating Chart for AC Outputs



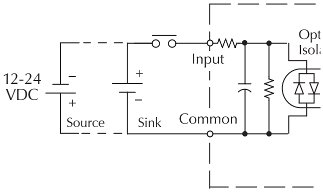
Equivalent Output Circuit



Standard Inputs (X4-X23)



High Speed Inputs (X0-X3)



D0-06DA General Specifications	
External Power Requirements	100–240 VAC/ 50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence/print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	High–Speed Inputs, X0 – X3	Standard DC Inputs X4 – X23
Input Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Maximum Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	70µs	N/A
ON Voltage Level	> 10VDC	> 10VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Input Impedance	1.8 kΩ @ 12–24 VDC	2.8 kΩ @ 12–24 VDC
Minimum ON Current	>5mA	>4mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	<70µs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 bank (isolated)	

AC Output Specifications	
Output Voltage Range (Min. - Max.)	15–264 VAC, 47–63 Hz
Operating Voltage	17–240 VAC, 47–63 Hz
On Voltage Drop	1.5 VAC @> 50mA, 4VAC @< 50mA
Max Current	0.5 A / point, 1.5 A / common
Max leakage current	< 4mA @ 264VAC, 60Hz
Max inrush current	10A for 10ms
Minimum Load	10mA
OFF to ON Response	1ms
ON to OFF Response	1 ms +1/2 cycle
Status Indicators	Logic Side
Commons	4 channels / common x 4 banks (isolated)
Fuses	None (external recommended)

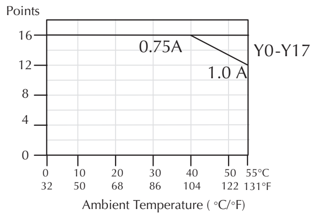
D0–06DD1 I/O Wiring Diagram

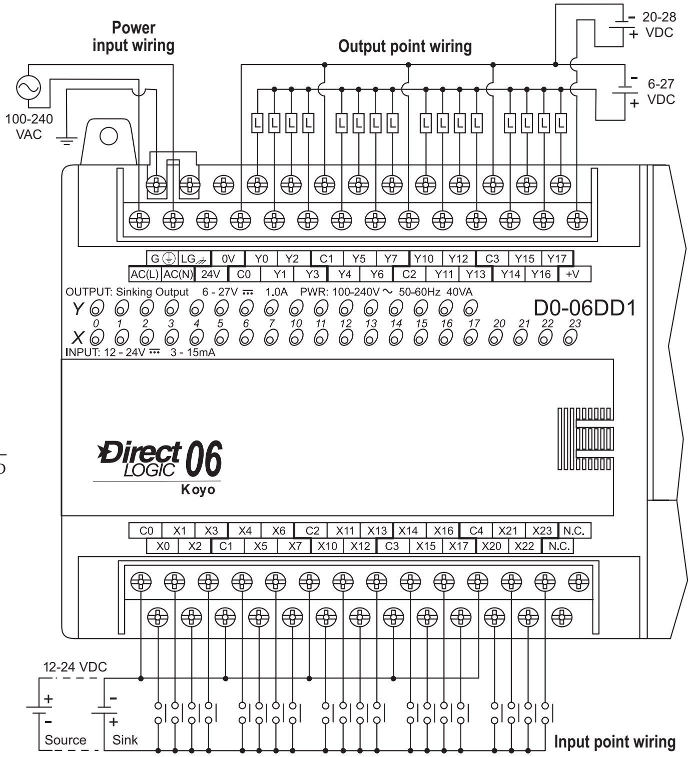
The D0-06DD1 PLC has twenty sinking/sourcing DC inputs and sixteen sinking DC outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals as shown.

Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used.

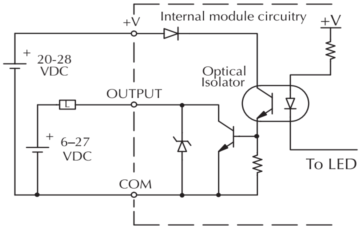
Outputs all share the same common. Note the requirement for external power.

Derating Chart for DC Outputs

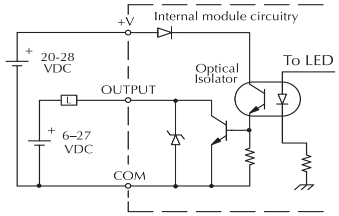




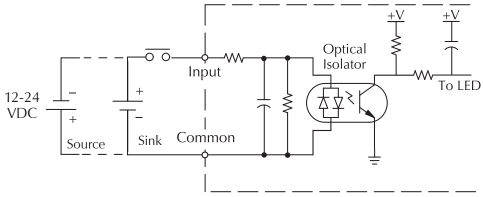
DC Pulse Outputs (Y0-Y1)



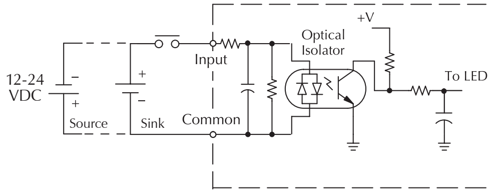
DC Standard Outputs (Y2-Y17)



DC Standard Inputs (X4-X23)



High Speed Inputs (X0-X3)



D0-06DD1 General Specifications	
External Power Requirements	100–240 VAC/ 50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence / print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	High–Speed Inputs, X0 – X3	Standard DC Inputs X4 – X23
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	100µs	N/A
ON Voltage Level	> 10.0 VDC	> 10.0 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Max. Input Current	6mA @12VDC, 13mA @24VDC	4mA @12VDC, 8.5mA @24VDC
Input Impedance	1.8 Ωk @ 12–24 VDC	2.8 Ωk @ 12–24 VDC
Minimum ON Current	>5mA	>4mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	<70µs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 banks isolated	

DC Output Specifications		
Parameter	Pulse Outputs Y0 – Y1	Standard Outputs Y2 – Y17
Min. - Max. Voltage Range	5–30 VDC	5–30 VDC
Operating Voltage	6–27 VDC	6–27 VDC
Peak Voltage	< 50VDC (10kHz max. frequency)	< 50VDC
On Voltage Drop	0.3 VDC @ 1 A	0.3 VDC @ 1A
Max Current (resistive)	0.5 A / pt., 1A / pt. as standard pt.	1.0 A / point
Max leakage current	15µA @ 30VDC	15µA @ 30VDC
Max inrush current	2A for 100ms	2A for 100ms
External DC power required	20–28 VDC Max 150mA	20–28 VDC Max 280mA (Aux. 24VDC powers V+ terminal (sinking outputs)
OFF to ON Response	< 10µs	< 10µs
ON to OFF Response	< 20µs	< 60µs
Status Indicators	Logic Side	Logic Side
Commons	4 channels / common x 4 banks non-isolated	
Fuses	None (external recommended)	

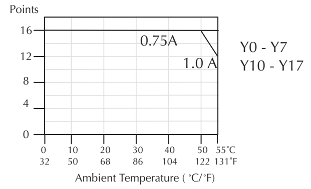
D0–06DD2 I/O Wiring Diagram

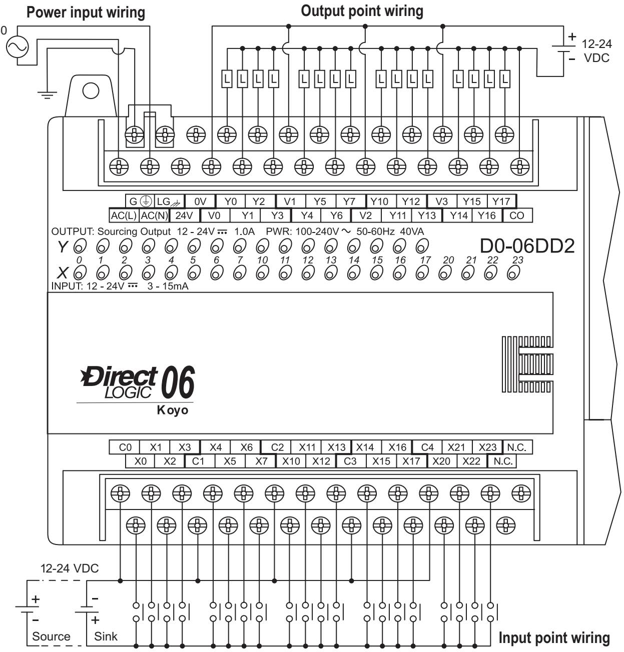
The D0–06DD2 PLC has twenty sinking/sourcing DC inputs and sixteen sourcing DC outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals as shown.

Inputs are organized into four banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used.

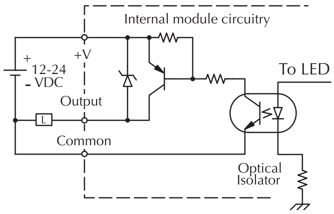
All outputs share the same common. Note the requirement for external power.

Derating Chart for DC Outputs

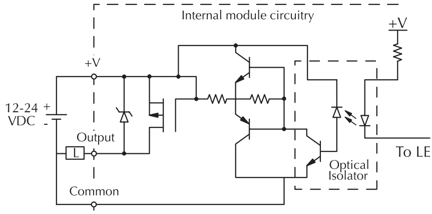



Power input wiring
100-240 VAC
Output point wiring
+ 12-24 VDC -
L
G | LG | 0V | Y0 | Y2 | V1 | Y5 | Y7 | Y10 | Y12 | V3 | Y15 | Y17
AC(L) | AC(N) | 24V | V0 | Y1 | Y3 | Y4 | Y6 | V2 | Y11 | Y13 | Y14 | Y16 | CO
OUTPUT: Sourcing Output 12 - 24V 1.0A PWR: 100-240V ~ 50-60Hz 40VA
Y
0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20 21 22 23
X
INPUT: 12 - 24V 3 - 15mA
D0-06DD2
Direct LOGIC 06
Koyo
C0 | X1 | X3 | X4 | X6 | C2 | X11 | X13 | X14 | X16 | C4 | X21 | X23 | N.C.
X0 | X2 | C1 | X5 | X7 | X10 | X12 | C3 | X15 | X17 | X20 | X22 | N.C.
12-24 VDC
Source
Sink
Input point wiring

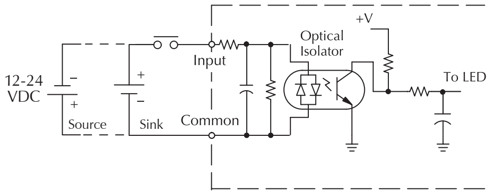
DC Standard Outputs (Y2-Y17)



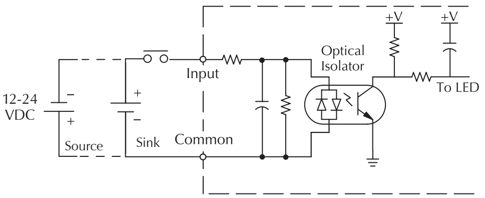
DC Pulse Outputs (Y0-Y1)



High Speed Inputs (X0-X3)



DC Standard Inputs (X4-X23)



D0-06DD2 General Specifications	
External Power Requirements	100–240 VAC/50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence / print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	**High–Speed Inputs, X0 – X3**	**Standard DC Inputs X4 – X23**
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30 VDC (7 kHz maximum frequency)	30VDC
Minimum Pulse Width	70 µs	N/A
ON Voltage Level	> 10.0 VDC	> 10.0 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Max. Input Current	6mA @ 12VDC, 13mA @ 24VDC	4mA @ 12VDC, 8.5 mA @ 24VDC
Input Impedance	1.8 Ωk @ 12–24 VDC	2.8 Ωk @ 12–24 VDC
Minimum ON Current	>5mA	>4mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	<70µs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels/common x 5 banks (isolated)	

DC Output Specifications		
Parameter	**Pulse Outputs Y0 – Y1**	**Standard Outputs Y2 – Y17**
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage	12–24 VDC	12–24 VDC
Peak Voltage	< 50VDC (10kHz max. frequency)	< 50VDC
On Voltage Drop	0.5 VDC @ 1A	1.2 VDC @ 1A
Max Current (resistive)	0.5 A / pt., 1A / pt. as standard pt.	1.0 A / point
Max leakage current	15µA @ 30VDC	15µA @ 30VDC
Max inrush current	2A for 100ms	2A for 100ms
External DC power required	12–24 VDC	12 -24 VDC
OFF to ON Response	< 10µs	< 10µs
ON to OFF Response	< 20µs	< 0.5 µs
Status Indicators	Logic Side	Logic Side
Commons	4 channels / common x 4 banks (non-isolated)	
Fuses	None (external recommended)	

D0–06DR I/O Wiring Diagram

The D0–06DR PLCs feature twenty DC inputs and sixteen relay contact outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals as shown.

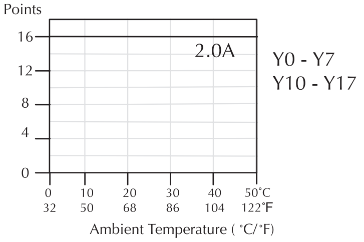
Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent circuit for standard inputs is shown below, and the high-speed input circuit is shown to the left.

Outputs are organized into four banks of four normally-open relay contacts. Each bank has a common terminal. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank. The relay contacts can switch AC or DC voltages.

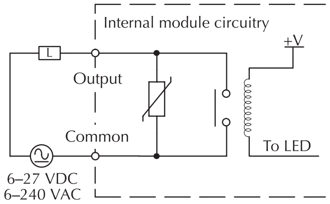
Typical Relay Life (Operations) at Room Temperature

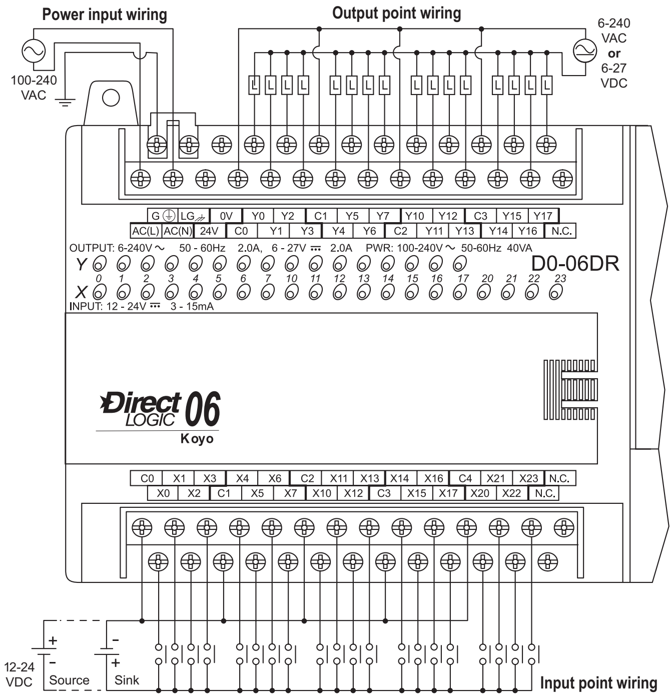
Voltage & Load Type	Load Current	
	At 1A	At 2A
24VDC Resistive	500K	250K
24VDC Inductive	100K	50K
110VAC Resistive	500K	250K
110VAC Inductive	200K	100K
220VAC Resistive	350K	200K
220VAC Inductive	100K	50K

Derating Chart for Relay Outputs

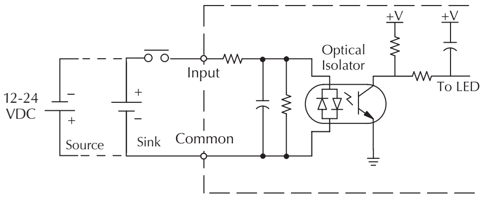


Equivalent Output Circuit

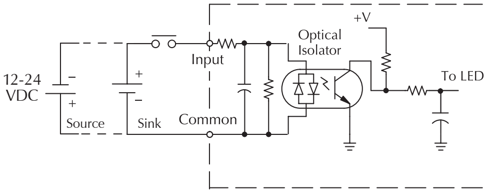




Equivalent Circuit, Standard Inputs (X4-X23)



Equivalent Circuit, High-speed Inputs (X0-X3)



D0-06DR General Specifications	
External Power Requirements	100–240 VAC/ 50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence /print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	**High–Speed Inputs, X0 – X3**	**Standard DC Inputs X4 – X23**
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	70μs	N/A
ON Voltage Level	> 10VDC	> 10VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Input Impedance	1.8 kΩ @ 12–24 VDC	2.8 kΩ @ 12–24 VDC
Max. Input Current	6mA @12VDC 13mA @ 24VDC	4mA @ 12VDC 8.5 mA @ 24VDC
Minimum ON Current	>5mA	>4mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70μs	2–8 ms, 4ms typical
ON to OFF Response	<70μs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 banks (isolated)	

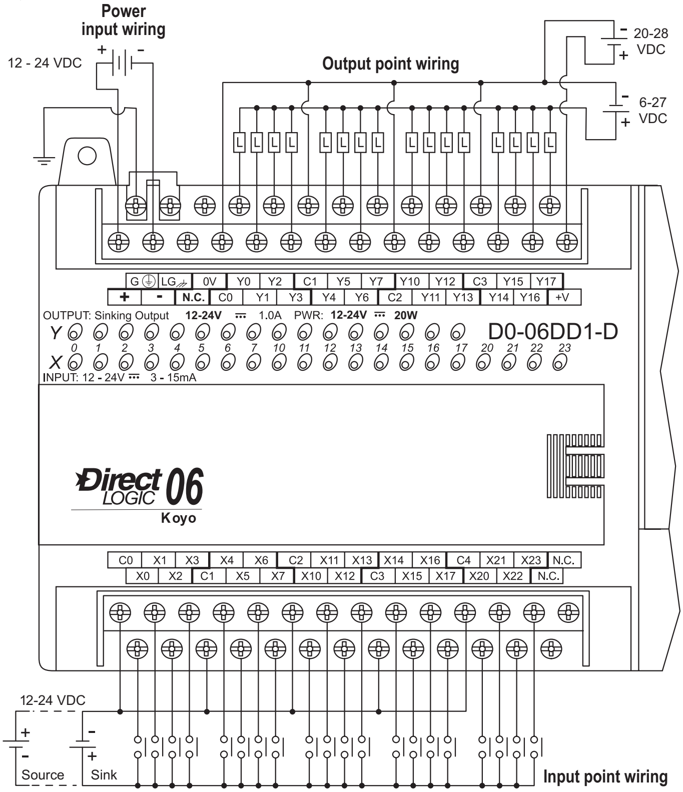
Relay Output Specifications	
Output Voltage Range (Min. - Max.)	5–264 VAC (47–63 Hz), 5–30 VDC
Operating Voltage	6–240 VAC (47–63 Hz), 6–27 VDC
Output Current	2A / point 6A / common
Maximum Voltage	264VAC, 30VDC
Max leakage current	0.1 mA @ 264VAC
Smallest Recommended Load	5mA
OFF to ON Response	< 15ms
ON to OFF Response	< 10ms
Status Indicators	Logic Side
Commons	4 channels / common x 4 banks (isolated)
Fuses	None (external recommended)

D0–06DD1–D I/O Wiring Diagram

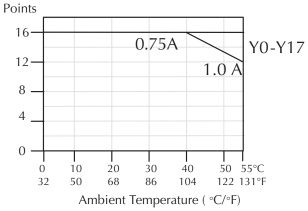
These micro PLCs feature twenty DC inputs and sixteen sinking DC outputs. The following diagram shows a typical field wiring example. The DC external power connection uses four terminals at the left as shown.

Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used.

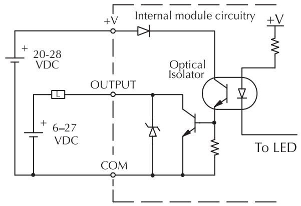
All outputs actually share the same common. Note the requirement for external power.



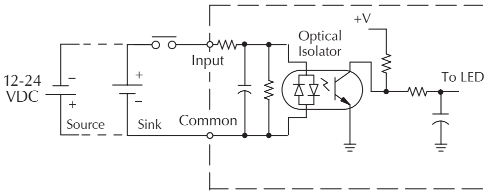
Derating Chart for DC Outputs



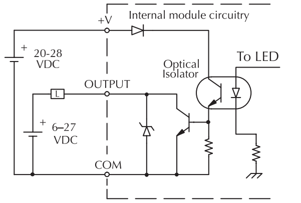
DC Pulse Outputs (Y0-Y1)



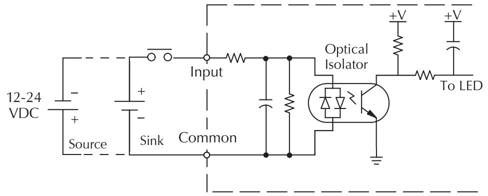
High Speed Inputs (X0-X3)



DC Standard Outputs (Y2-Y17)



Standard Input Circuit (X4-X23)



D0-06DD1-D General Specifications	
External Power Requirements	12–24 VDC, 20W maximum,
Communication Port 1: 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2: 9600 baud (default), 8 data bits, 1 stop bit,odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence/print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	High–Speed Inputs, X0 – X3	Standard DC Inputs X4 – X23
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	70µs	N/A
ON Voltage Level	>10.0 VDC	> 10.0 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Max. Input Current	6mA @12VDC, 13mA @ 24VDC	4mA @12VDC, 8.5 mA @ 24VDC
Input Impedance	1.8 kΩ @ 12–24VDC	2.8 kΩ @ 12–24 VDC
Minimum ON Current	>5mA	>4mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	<70µs	2–8 ms, 4 ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 banks (isolated)	

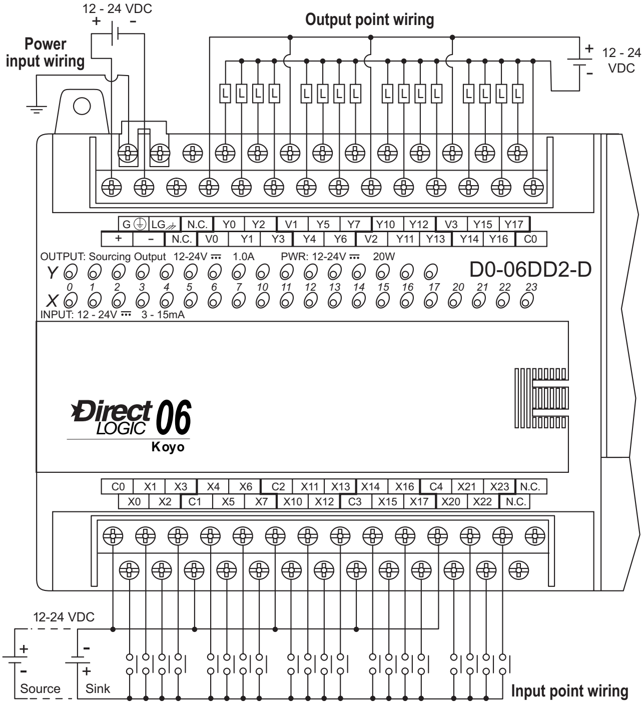
DC Output Specifications		
Parameter	Pulse Outputs, Y0 – Y1	Standard Outputs, Y2 – Y17
Min. - Max. Voltage Range	5–30 VDC	5–30 VDC
Operating Voltage	6–27 VDC	6–27 VDC
Peak Voltage	< 50VDC (10kHz max. frequency)	< 50VDC
On Voltage Drop	0.3 VDC @ 1 A	0.3 VDC @ 1A
Max Current (resistive)	0.5 A / pt., 1A / pt. as standard pt.	1.0 A / point
Max leakage current	15µA @ 30VDC	15µA @ 30VDC
Max inrush current	2 A for 100 ms	2A for 100ms
External DC power required	20–28 VDC Max 150mA	20–28 VDC Max 150mA
OFF to ON Response	< 10µs	< 10µs
ON to OFF Response	< 20µs	< 60µs
Status Indicators	Logic Side	Logic Side
Commons	4 channels / common x 4 banks (non-isolated)	
Fuses	None (external recommended)	

D0–06DD2–D I/O Wiring Diagram

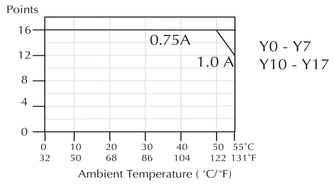
These micro PLCs feature twenty DC inputs and sixteen sourcing DC outputs. The following diagram shows a typical field wiring example. The DC external power connection uses four terminals at the left as shown.

Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used.

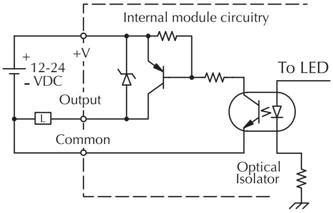
All outputs actually share the same common. Note the requirement for external power.


12 - 24 VDC
+ -
Power input wiring
Output point wiring
+ 12 - 24 VDC -
L
G LG N.C. Y0 Y2 V1 Y5 Y7 Y10 Y12 V3 Y15 Y17
+ - N.C. V0 Y1 Y3 Y4 Y6 V2 Y11 Y13 Y14 Y16 C0
OUTPUT: Sourcing Output 12-24V 1.0A PWR: 12-24V 20W
Y 0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20 21 22 23
X
INPUT: 12 - 24V 3 - 15mA
D0-06DD2-D
Direct LOGIC 06
Koyo
C0 X1 X3 X4 X6 C2 X11 X13 X14 X16 C4 X21 X23 N.C.
X0 X2 C1 X5 X7 X10 X12 C3 X15 X17 X20 X22 N.C.
12-24 VDC
+ - Source
- + Sink
Input point wiring

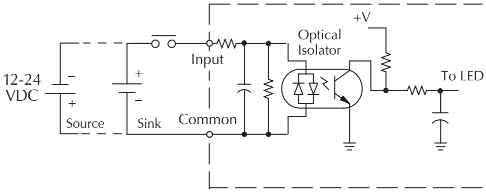
Derating Chart for DC Outputs



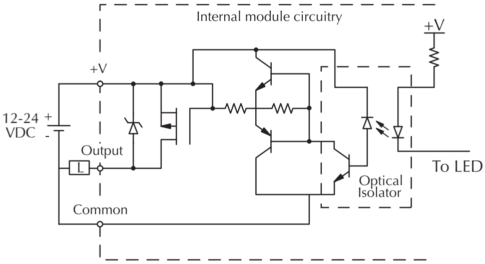
DC Standard Outputs (Y2-Y17)



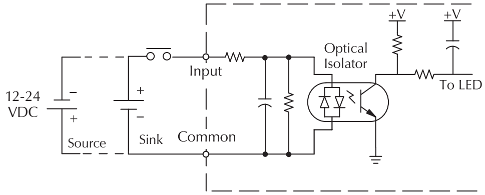
High Speed Inputs (X0-X3)



DC Pulse Outputs (Y0-Y1)



Standard Input Circuit (X4-X23)



D0-06DD2-D General Specifications	
External Power Requirements	12–24 VDC, 20W maximum,
Communication Port 1: 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2: 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence/print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	High–Speed Inputs, X0 – X3	Standard DC Inputs X4 – X23
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	70µs	N/A
ON Voltage Level	>10.0 VDC	> 10.0 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Max. Input Current	15mA @26.4VDC	11mA @ 26.4 VDC
Input Impedance	1.8 kΩ @ 12–24 VDC	2.8 kΩ @ 12–24 VDC
Minimum ON Current	5mA	3mA
Maximum OFF Current	0.5 mA	0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	<70µs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 banks (isolated)	

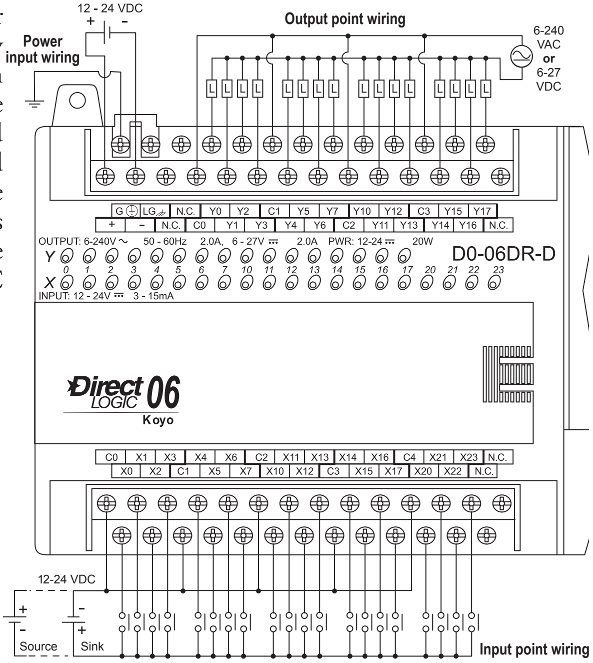
DC Output Specifications		
Parameter	Pulse Outputs, Y0 – Y1	Standard Outputs, Y2 – Y17
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (10kHz max. frequency)	30VDC
On Voltage Drop	0.5 VDC @ 1A	1.2 VDC @ 1 A
Max Current (resistive)	0.5 A / pt., 1A / pt. as standard pt.	1.0 A / point
Max leakage current	15µA @ 30VDC	15µA @ 30VDC
Max inrush current	2A for 100ms	2A for 100ms
External DC power required	N/A	N/A
OFF to ON Response	< 10µs	< 10µs
ON to OFF Response	< 20µs	< 0.5 ms
Status Indicators	Logic Side	Logic Side
Commons	4 channels / common x 4 banks (non-isolated)	
Fuses	None (external recommended)	

D0–06DR–D I/O Wiring Diagram

The D0–06DR–D PLC has twenty DC inputs and sixteen relay contact outputs. The following diagram shows a typical field wiring example. The DC external power connection uses three terminals as shown.

Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example above shows all commons connected together, but separate supplies and common circuits may be used.

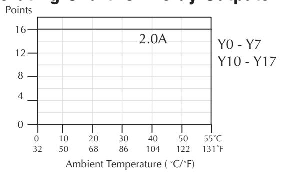
Outputs are organized into four banks of four normally-open relay contacts. Each bank has a common terminal. The wiring example above shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank. The relay contacts can switch AC or DC voltages.


12 - 24 VDC
+
-
Power input wiring
Output point wiring
6-240 VAC or 6-27 VDC
L
G LG N.C. Y0 Y2 C1 Y5 Y7 Y10 Y12 C3 Y15 Y17
+ - N.C. C0 Y1 Y3 Y4 Y6 C2 Y11 Y13 Y14 Y16 N.C.
OUTPUT: 6-240V ~ 50 - 60Hz 2.0A, 6 - 27V ⎓ 2.0A PWR: 12-24⎓ 20W
D0-06DR-D
Y
0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20 21 22 23
X
INPUT: 12 - 24V ⎓ 3 - 15mA
Direct LOGIC 06
Koyo
C0 X1 X3 X4 X6 C2 X11 X13 X14 X16 C4 X21 X23 N.C.
X0 X2 C1 X5 X7 X10 X12 C3 X15 X17 X20 X22 N.C.
12-24 VDC
+
-
Source
-
+
Sink
Input point wiring

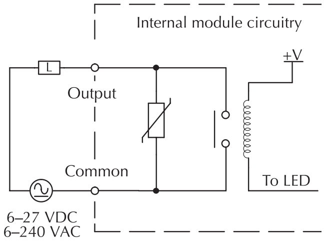
Typical Relay Life (Operations) at Room Temperature

Voltage & Load Type	Load Current	
	At 1A	At 2A
24VDC Resistive	500K	250K
24VDC Inductive	100K	50K
110VAC Resistive	500K	250K
110VAC Inductive	200K	100K
220VAC Resistive	350K	200K
220VAC Inductive	100K	50K

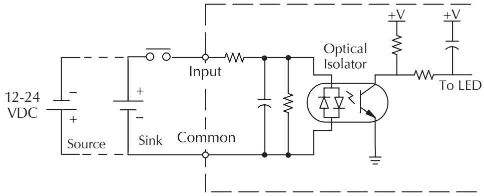
Derating Chart for Relay Outputs



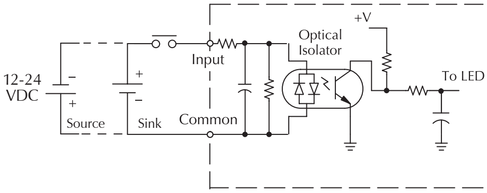
Standard Output Circuit



Standard Input Circuit (X4-X23)



High-speed Input Circuit (X0-X3)



D0-06DR-D General Specifications	
External Power Requirements	12–24 VDC, 20W maximum,
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave),Non-sequence/print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18AWG, 24AWG minimum

DC Input Specifications		
Parameter	**High–Speed Inputs, X0 – X3**	**Standard DC Inputs X4 – X23**
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	70µs	N/A
ON Voltage Level	> 10VDC	> 10VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Input Impedance	1.8 kΩ @ 12–24 VDC	2.8 kΩ @ 12–24 VDC
Max. Input Current	6mA @ 12VDC 13mA @ 24VDC	4mA @12VDC 8.5mA @ 24VDC
Minimum ON Current	>5mA	>4 mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	< 70µs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 banks (isolated)	

Relay Output Specifications	
Output Voltage Range (Min. - Max.)	5–264 VAC (47–63 Hz), 5–30 VDC
Operating Voltage	6–240 VAC (47–63 Hz), 6–27 VDC
Output Current	2A / point 6A / common
Maximum Voltage	264VAC, 30VDC
Max leakage current	0.1 mA @ 264VAC
Smallest Recommended Load	5mA
OFF to ON Response	< 15ms
ON to OFF Response	< 10ms
Status Indicators	Logic Side
Commons	4 channels / common x 4 banks isolated commons
Fuses	None (external recommended)

Glossary of Specification Terms

Discrete Input

One of twenty input connections to the PLC which converts an electrical signal from a field device to a binary status (off or on), which is read by the internal CPU each PLC scan.

Discrete Output

One of sixteen output connections from the PLC which converts an internal ladder program result (0 or 1) to turn On or Off an output switching device. This enables the program to turn on and off large field loads.

I/O Common

A connection in the input or output terminals which is shared by multiple I/O circuits. It usually is in the return path to the power supply of the I/O circuit.

Input Voltage Range

The operating voltage range of the input circuit.

Maximum Voltage

Maximum voltage allowed for the input circuit.

ON Voltage Level

The minimum voltage level at which the input point will turn ON.

OFF Voltage Level

The maximum voltage level at which the input point will turn OFF

Input Impedance

Input impedance can be used to calculate input current for a particular operating voltage.

Input Current

Typical operating current for an active (ON) input.

Minimum ON Current

The minimum current for the input circuit to operate reliably in the ON state.

Maximum OFF Current

The maximum current for the input circuit to operate reliably in the OFF state.

OFF to ON Response

The time the module requires to process an OFF to ON state transition.

ON to OFF Response

The time the module requires to process an ON to OFF state transition.

Status Indicators

The LEDs that indicate the ON/OFF status of an input or output point. All LEDs on DL06 Micro PLCs are electrically located on the logic side of the input or output circuit.

CPU Specifications and Operation

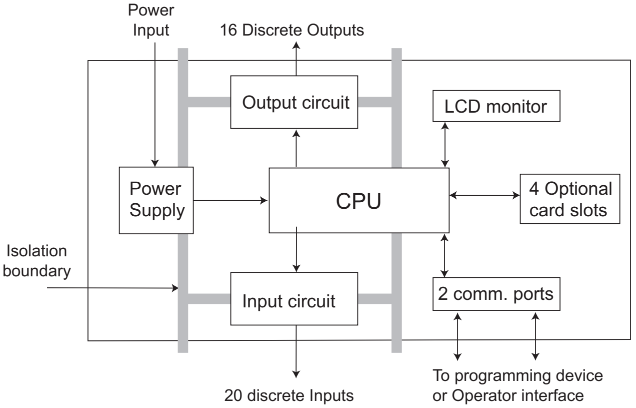
Chapter 3

In This Chapter

Overview

The Central Processing Unit (CPU) is the heart of the Micro PLC. Almost all PLC operations are controlled by the CPU, so it is important that it is set up correctly. This chapter provides the information needed to understand:

- Steps required to set up the CPU
- Operation of ladder programs
- Organization of Variable Memory



NOTE: *The High-Speed I/O function (HSIO) consists of dedicated but configurable hardware in the DL06. It is not considered part of the CPU because it does not execute the ladder program. For more on HSIO operation, see Appendix E.*

DL06 CPU Features

The DL06 Micro PLC has 14.8K words of memory comprised of 7.6K of ladder memory and 7.6K words of V-memory (data registers). Program storage is in the FLASH memory which is a part of the CPU board in the PLC. In addition, there is RAM with the CPU which will store system parameters, V-memory, and other data not in the application program. The RAM is backed up by a super-capacitor, storing the data for several hours in the event of a power outage. The capacitor automatically charges during powered operation of the PLC.

The DL06 supports fixed I/O which includes twenty discrete input points and sixteen output points.

Over 220 different instructions are available for program development as well as extensive internal diagnostics that can be monitored from the application program or from an operator interface. Chapters 5, 6, and 7 provide detailed descriptions of the instructions.

The DL06 provides two built-in communication ports, so you can easily connect a handheld programmer, operator interface, or a personal computer without needing any additional hardware.

CPU Specifications

Specifications	
Feature	**DL06**
Total Program memory (words)	14.8K
Ladder memory (words)	7680
Total V-memory (words)	7616
User V-memory (words)	7488
Non-volatile V Memory (words)	128
Contact execution (boolean)	<0.6us
Typical scan (1k boolean)	1-2ms
RLL Ladder style Programming	Yes
RLL and RLLPLUS Programming	Yes
Run Time Edits	Yes
Supports Overrides	Yes
Scan	Variable / fixed
Handheld programmer	Yes
*Direct*SOFT **programming for Windows**	Yes
Built-in communication ports (RS232C)	Yes
FLASH Memory	Standard on CPU
Local Discrete I/O points available	36
Local Analog input / output channels maximum	None
High-Speed I/O (quad., pulse out, interrupt, pulse catch, etc.)	Yes, 2
I/O Point Density	20 inputs, 16 outputs
Number of instructions available (see Chapter 5 for details)	229
Control relays	1024
Special relays (system defined)	512
Stages in RLL^{PLUS}	1024
Timers	256
Counters	128
Immediate I/O	Yes
Interrupt input (external / timed)	Yes
Subroutines	Yes
For/Next Loops	Yes
Math (Integer and floating point)	Yes
Drum Sequencer Instruction	Yes
Time of Day Clock/Calendar	Yes
Internal diagnostics	Yes
Password security	Yes
System error log	Yes
User error log	Yes
Battery backup	Optional D2-BAT-1 available (not included with unit)

CPU Hardware Setup

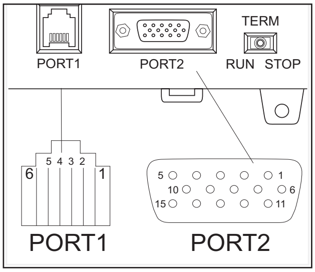
Communication Port Pinout Diagrams

Cables are available that allow you to quickly and easily connect a Handheld Programmer or a personal computer to the DL06 PLCs. However, if you need to build your cable(s), use the pinout descriptions shown below, or use the Tech Support/Cable Wiring Diagrams located on our website. The DL06 PLCs require an RJ-12 phone plug for port 1 (D2-DSCBL) and a 15-pin SVGA DSub for port 2 (D2-DSCBL-1).

The DL06 PLC has two built-in serial communication ports. Port 1 (RS232C only) is generally used for connecting to a D2-HPP, *Direct*SOFT, operator interface, MODBUS slave only, or a *Direct*NET slave only. The baud rate is fixed at 9600 baud for port 1. Port 2 (RS232C/RS422/RS485) can be used to connect to a D2-HPP, *Direct*SOFT, operator interface, MODBUS master/slave, *Direct*NET master/slave or ASCII in/out. Port 2 has a range of speeds from 300 baud to 38.4K baud.

NOTE: *The 5v pins are rated at 220mA maximum, primarily for use with some operator interface units.*

Port 1 Pin Descriptions		
1	0V	Power (-) connection (GND)
2	5V	Power (-) 220 mA max
3	RXD	Receive data (RS-232C)
4	TXD	Transmit data (RS-232C)
5	5V	Power (+) connection
6	0V	Power (-) connection (GND)



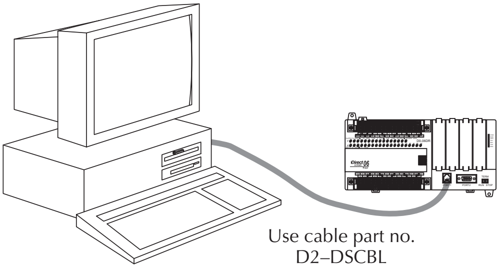
Communications Port 1	
Com 1	Connects to HPP, ***Direct***SOFT, operator interfaces, etc.
	6-pin, RS232C
	Communication speed (baud): 9600 (fixed)
	Parity: odd (fixed)
	Station Address: 1 (fixed)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocol (auto-select): K-sequence (slave only), ***Direct***NET (slave only), MODBUS (slave only)

Port 2 Pin Descriptions		
1	5V	Power (+) connection
2	TXD	Transmit data (RS-232C)
3	RXD	Receive data (RS-232C)
4	RTS	Ready to send
5	CTS	Clear to send
6	RXD-	Receive data (-) (RS-422/485)
7	0V	Power (-) connection (GND)
8	0V	Power (-) connection (GND)
9	TXD+	Transmit data (+) (RS-422/485)
10	TXD-	Transmit data (-) (RS-422/485)
11	RTS+	Ready to send (+) (RS-422/485)
12	RTS-	Ready to send (-) (RS-422/485)
13	RXD+	Receive data (+) (RS-422/485)
14	CTS+	Clear to send (+) (RS-422/485)
15	CTS-	Clear to send (-) (RS-422/485)

Communications Port 2	
Com 2	Connects to HPP, ***Direct***SOFT, operator interfaces, etc.
	15-pin, multifunction port, RS232C, RS422, RS485 (RS485 with 2-wire is only available for MODBUS and Non-sequence.)
	Communication speed (baud): 300, 600, 1200, 2400, 4800, 9600, 19200, 38400
	Parity: odd (default), even, none
	Station Address: 1 (default)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocol (auto-select): K-sequence (slave only), ***Direct***NET (master/slave), MODBUS (master/slave), non-sequence/print/ASCII in/out

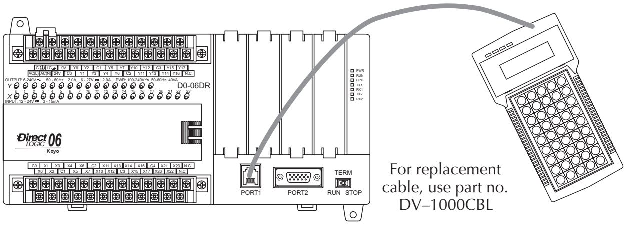
Connecting the Programming Devices

If you're using a Personal Computer with the ***Direct***SOFT programming package, you can connect the computer to either of the DL06's serial ports. For an engineering office environment (typical during program development), this is the preferred method of programming.



Use cable part no.
D2–DSCBL

The Handheld programmer D2-HPP is connected to the CPU with a handheld programmer cable. This device is ideal for maintaining existing installations or making small program changes. The handheld programmer is shipped with a cable, which is approximately 6.5 feet (200 cm) long.



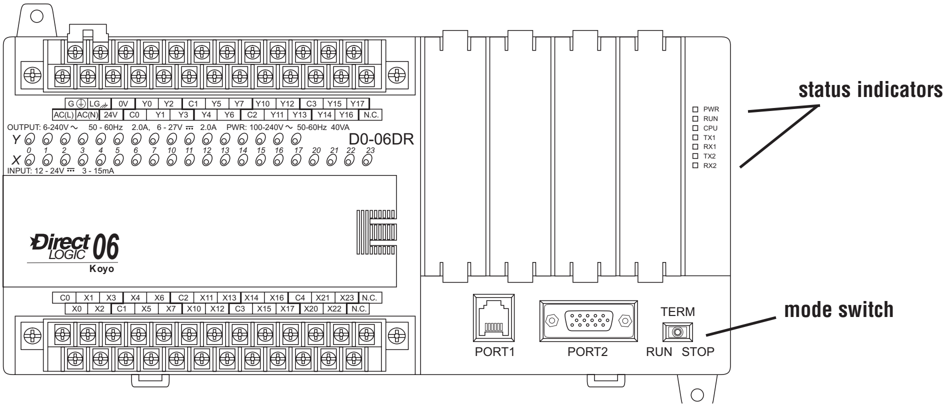
For replacement
cable, use part no.
DV–1000CBL

CPU Setup Information

Even if you have years of experience using PLCs, there are a few things you need to do before you can start entering programs. This section includes some basic things, such as changing the CPU mode, but it also includes some things that you may never have to use. Here's a brief list of the items that are discussed:

- Using Auxiliary Functions
- Clearing the program (and other memory areas)
- How to initialize system memory
- Setting retentive memory ranges

The following paragraphs provide the setup information necessary to get the CPU ready for programming. They include setup instructions for either type of programming device you are using. The D2–HPP Handheld Programmer Manual provides the Handheld keystrokes required to perform all of these operations. The ***Direct***SOFT Manual provides a description of the menus and keystrokes required to perform the setup procedures via ***Direct***SOFT.



Status Indicators

The status indicator LEDs on the CPU front panels have specific functions which can help in programming and troubleshooting.

Mode Switch Functions

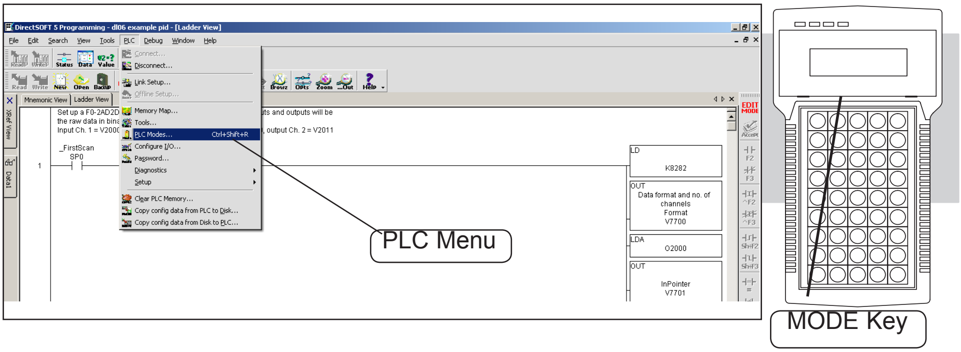
The mode switch on the DL06 PLC provides positions for enabling and disabling program changes in the CPU. Unless the mode switch is in the TERM position, RUN and STOP mode changes will not be allowed by any interface device, (handheld programmer, *Direct*SOFT programming package or operator interface). Programs may be viewed or monitored but no changes may be made. If the switch is in the TERM position and no program password is in effect, all operating modes as well as program access will be allowed through the connected programming or monitoring device.

Indicator	Status	Meaning
PWR	ON	Power good
	OFF	Power failure
RUN	ON	CPU is in Run Mode
	OFF	CPU is in Stop or Program Mode
	Blinking	CPU is in firmware upgrade mode
CPU	ON	CPU self diagnostics error
	OFF	CPU self diagnostics good
	Blinking	The CPU indicator will blink if the battery is less than 2.5 VDC
TX1	ON	Data is being transmitted by the CPU - Port 1
	OFF	No data is being transmitted by the CPU - Port 1
RX1	ON	Data is being received by the CPU - Port 1
	OFF	No data is being received by the CPU - Port 1
TX2	ON	Data is being transmitted by the CPU - Port 2
	OFF	No data is being transmitted by the CPU - Port 2
RX2	ON	Data is being received by the CPU - Port 2
	OFF	No data is being received by the CPU - Port 2

Changing Modes in the DL06 PLC

Mode Switch Position	CPU Action
RUN (Run Program)	CPU is forced into the RUN mode if no errors are encountered. No changes are allowed by the attached programming/ monitoring device.
TERM (Terminal) RUN	PROGRAM and the TEST modes are available. Mode and program changes are allowed by the programming/monitoring device.
STOP	CPU is forced into the STOP mode. No changes are allowed by the programming/monitoring device.

There are two ways to change the CPU mode. You can use the CPU mode switch to select the operating mode, or you can place the mode switch in the TERM position and use a programming device to change operating modes. With the switch in this position, the CPU can be changed between Run and Program modes. You can use either *Direct*SOFT or the Handheld Programmer to change the CPU mode of operation. With *Direct*SOFT use the PLC menu option **PLC > Mode** or use the **Mode** button located on the Online toolbar. With the Handheld Programmer, you use the MODE key.


DirectSOFT 5 Programming - dl06 example pid - [Ladder View]
File Edit Search View Tools PLC Debug Window Help
Connect...
Disconnect...
Link Setup...
Offline Setup...
Memory Map...
Tools...
PLC Modes... Ctrl+Shift+R
Configure I/O...
Password...
Diagnostics
Setup
Clear PLC Memory...
Copy config data from PLC to Disk...
Copy config data from Disk to PLC...
Mnemonic View
Ladder View
_FirstScan
SP0
LD
K8282
OUT
Data format and no. of channels
Format
V7700
LDA
O2000
OUT
InPointer
V7701
PLC Menu
MODE Key

Mode of Operation at Power-up

The DL06 CPU will normally power-up in the mode that it was in just prior to the power interruption. For example, if the CPU was in Program Mode when the power was disconnected, the CPU will power-up in Program Mode (see warning note below).

WARNING: Once the super capacitor has discharged, the system memory may not retain the previous mode of operation. When this occurs, the PLC can power-up in either Run or Program Mode if the mode switch is in the term position. There is no way to determine which mode will be entered as the startup mode. Failure to adhere to this warning greatly increases the risk of unexpected equipment startup.

The mode which the CPU will power-up in is also determined by the state of B7633.13. If the bit is set and the Mode Switch is in the TERM position, the CPU will power-up in RUN mode. If B7633.13 is not set with the Mode Switch in TERM position, then the CPU will power-up in the state it was in when it was powered-down.

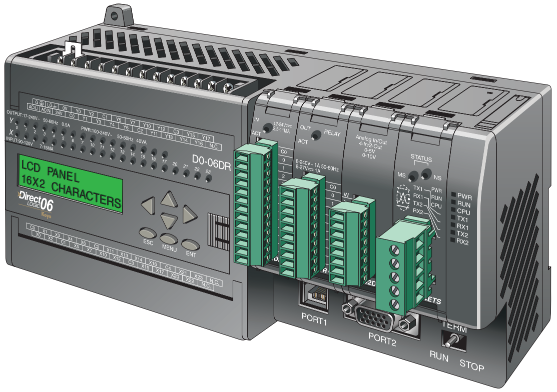
Using Battery Backup

An optional lithium battery is available to maintain the system RAM retentive memory when the DL06 system is without external power. Typical CPU battery life is five years, which includes PLC runtime and normal shutdown periods. However, consider installing a fresh battery if your battery has not been changed recently and the system will be shut down for a period of more than ten days.

***NOTE**: Before installing or replacing your CPU battery, back-up your V-memory and system parameters. You can do this by using **Direct**SOFT to save the program, V-memory, and system parameters to hard/floppy disk on a personal computer.*

To install the D2–BAT–1 CPU battery in the DL06 CPU:

1. Press the retaining clip on the battery door down and swing the battery door open.
2. Place the battery into the coin–type slot with the +, or larger, side out.
3. Close the battery door making sure that it locks securely in place.
4. Make a note of the date the battery was installed



Battery door

WARNING: Do not attempt to recharge the battery or dispose of an old battery by fire. The battery may explode or release hazardous materials.

Battery Backup

The battery backup is available immediately after the battery has been installed. ***The CPU indicator will blink if the battery is low*** (refer to the table on page 3-6). Special Relay 43 (SP43) will also be set when the battery is low. The low battery indication is enabled by setting bit 12 of V7633 (B7633.12). If the low battery feature is not desired, do not set bit V7633.12. The super capacitor will retain memory IF it is configured as retentive regardless of the state of B7633.12. The battery will do the same, but for a much longer time.

Auxiliary Functions

Many CPU setup tasks involve the use of Auxiliary (AUX) Functions. The AUX Functions perform many different operations, ranging from clearing ladder memory, displaying the scan time, copying programs to EEPROM in the handheld programmer, etc. They are divided into categories that affect different system parameters. Appendix A provides a description of the AUX functions.

You can access the AUX Functions from *Direct*SOFT or from the D2–HPP Handheld Programmer. The manuals for those products provide step-by-step procedures for accessing the AUX Functions. Some of these AUX Functions are designed specifically for the Handheld Programmer setup, so they will not be needed (or available) with the *Direct*SOFT package. The following table shows a list of the Auxiliary functions for the Handheld Programmer.

Auxiliary Functions	
AUX 2* — RLL Operations	
21	Check Program
22	Change Reference
23	Clear Ladder Range
24	Clear All Ladders
AUX 3* — V-Memory Operations	
31	Clear V-memory
AUX 4* — I/O Configuration	
41	Show I/O Configuration
42	I/O Diagnostics
44	Power Up I/O Configuration check
45	Select Configuration
46	Configure I/O
AUX 5* — CPU Configuration	
51	Modify Program Name
52	Display/Change Calendar
53	Display Scan Time
54	Initialize Scratchpad
55	Set Watchdog Timer
56	Set Communication Port 2

Auxiliary Functions (cont'd)	
57	Set Retentive Ranges
58	Test Operations
59	Override Setup
5B	HSIO Configuration
5C	Display Error History
5D	Scan Control Setup
AUX 6* — Handheld Programmer Configuration	
61	Show Revision Numbers
62	Beeper On / Off
65	Run Self Diagnostics
AUX 7* — EEPROM Operations	
71	Copy CPU memory to HPP EEPROM
72	Write HPP EEPROM to CPU
73	Compare CPU to HPP EEPROM
74	Blank Check (HPP EEPROM)
75	Erase HPP EEPROM
76	Show EEPROM Type (CPU and HPP)
AUX 8* — Password Operations	
81	Modify Password
82	Unlock CPU
83	Lock CPU

Clearing an Existing Program

Before you enter a new program, be sure to always clear ladder memory. You can use AUX Function 24 to clear the complete program. You can also use other AUX functions to clear other memory areas.

• AUX 23 — Clear Ladder Range • AUX 24 — Clear all Ladders • AUX 31 — Clear V-memory

Initializing System Memory

The DL06 Micro PLC maintains system parameters in a memory area often referred to as the **scratchpad**. In some cases, you may make changes to the system setup that will be stored in system memory. For example, if you specify a range of Control Relays (CRs) as retentive, these changes are stored in system memory. AUX 54 resets the system memory to the default values.

⚠ **WARNING: You may never have to use this feature unless you want to clear any setup information that is stored in system memory. Usually, you'll only need to initialize the system memory if you are changing programs and the old program required a special system setup. You can usually load in new programs without ever initializing system memory.**

Remember, this AUX function will reset all system memory. If you have set special parameters such as retentive ranges, for example, they will be erased when AUX 54 is used. Make sure that you have considered all ramifications of this operation before you select it. See Appendix F for additional information in reference to PLC memory.

Setting Retentive Memory Ranges

The DL06 PLCs provide certain ranges of retentive memory by default. The default ranges are suitable for many applications, but you can change them if your application requires additional retentive ranges or no retentive ranges at all. The default settings are:

Memory Area	DL06	
	Default Range	Available Range
Control Relays	C1000 – C1777	C0 – C1777
V-Memory	V400 – V37777	V0 – V37777
Timers	None by default	T0 – T377
Counters	CT0 – CT177	CT0 – CT177
Stages	None by default	S0 – S1777

You can use AUX 57 to set the retentive ranges. You can also use *Direct*SOFT menus to select the retentive ranges. Appendix A contains detailed information about auxiliary functions.

⚠ **WARNING: The DL06 CPUs do not come with a battery. The super capacitor will retain the values in the event of a power loss, but only for a short period of time, depending on conditions (typically 4 to 7 days). If the retentive ranges are important for your application, make sure you obtain the optional battery.**

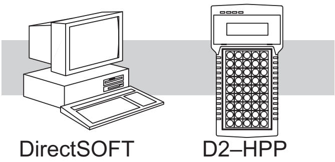
Using a Password

The DL06 PLCs allow you to use a password to help minimize the risk of unauthorized program and/or data changes. Once you enter a password you can lock the PLC against access. Once the CPU is locked you must enter the password before you can use a programming device to change any system parameters.

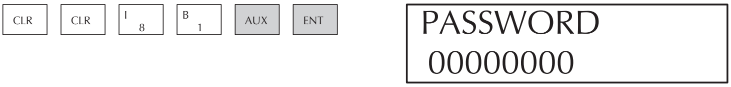
You can select an 8-digit numeric password. The Micro PLCs are shipped from the factory with a password of 00000000. All zeros removes the password protection. If a password has been entered into the CPU you cannot just enter all zeros to remove it. Once you enter the correct password, you can change the password to all zeros to remove the password protection.

WARNING: Make sure you remember your password. If you forget your password you will not be able to access the CPU. The Micro PLC must be returned to the factory to have the password (along with the ladder project) removed. It is the policy of Automationdirect to require the memory of the PLC to be cleared along with the password.

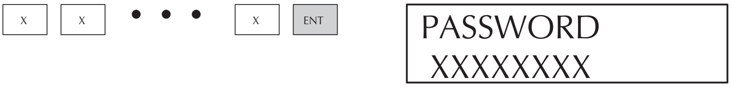
You can use the D2–HPP Handheld Programmer or ***Direct*SOFT**. to enter a password. The following diagram shows how you can enter a password with the Handheld Programmer.



Select AUX 81



Enter the new 8-digit password



Press CLR to clear the display

There are three ways to lock the CPU once the password has been entered.

1. If the CPU power is disconnected, the CPU will be automatically locked against access.
2. If you enter the password with ***Direct*SOFT**, the CPU will be automatically locked against access when you exit ***Direct*SOFT**.
3. Use AUX 83 to lock the CPU.

When you use ***Direct*SOFT**, you will be prompted for a password if the CPU has been locked. If you use the Handheld Programmer, you have to use AUX 82 to unlock the CPU. Once you enter AUX 82, you will be prompted to enter the password.

***NOTE**: The DL06 CPUs support multi-level password protection of the ladder program. This allows password protection while not locking the communication port to an operator interface. The multi-level password can be invoked by creating a password with an upper case **A** followed by seven numeric characters (e.g. A1234567).*

CPU Operation

Achieving the proper control for your equipment or process requires a good understanding of how DL06 CPUs control all aspects of system operation. There are four main areas to understand before you create your application program:

- CPU Operating System — the CPU manages all aspects of system control. A quick overview of all the steps is provided in the next section.
- CPU Operating Modes — The two primary modes of operation are Program Mode and Run Mode.
- CPU Timing — The two important areas we discuss are the I/O response time and the CPU scan time.
- CPU Memory Map — DL06 CPUs offer a wide variety of resources, such as timers, counters, inputs, etc. The memory map section shows the organization and availability of these data types.

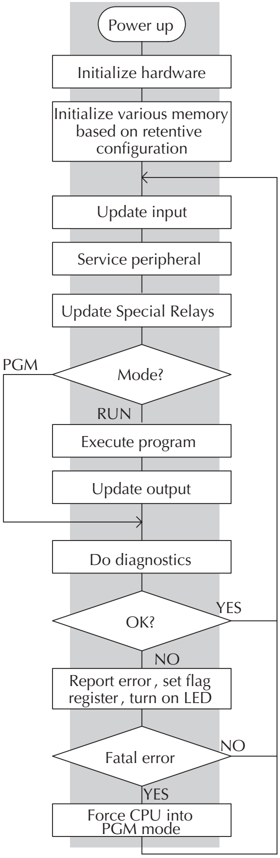
CPU Operating System

At powerup, the CPU initializes the internal electronic hardware. Memory initialization starts with examining the retentive memory settings. In general, the contents of retentive memory is preserved, and non-retentive memory is initialized to zero (unless otherwise specified).

After the one-time powerup tasks, the CPU begins the cyclical scan activity. The flowchart to the right shows how the tasks differ, based on the CPU mode and the existence of any errors. The scan time is defined as the average time around the task loop. Note that the CPU is always reading the inputs, even during program mode. This allows programming tools to monitor input status at any time.

The outputs are only updated in Run mode. In program mode, they are in the off state.

Error detection has two levels. Non-fatal errors are reported, but the CPU remains in its current mode. If a fatal error occurs, the CPU is forced into program mode and the outputs go off.



Program Mode

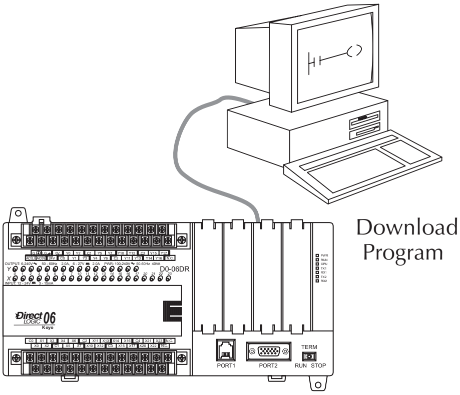
In Program Mode, the CPU does not execute the application program or update the output points. The primary use for Program Mode is to enter or change an application program. You also use program mode to set up the CPU parameters, such as HSIO features, retentive memory areas, etc.

You can use a programming device, such as *Direct*SOFT, the D2–HPP (Handheld Programmer) or the CPU mode switch to place the CPU in Program Mode.

Run Mode

In Run Mode, the CPU executes the application program and updates the I/O system. You can perform many operations during Run Mode. Some of these include:

- Monitor and change I/O point status
- Change timer/counter preset values
- Change variable memory locations

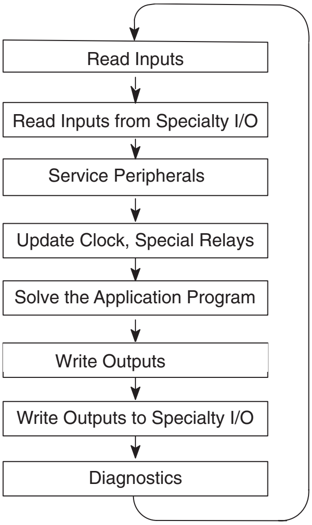


Run Mode operation can be divided into several key areas. For the vast majority of applications, some of these execution segments are more important than others. For example, you need to understand how the CPU updates the I/O points, handles forcing operations, and solves the application program. The remaining segments are not that important for most applications.

You can use *Direct*SOFT, the D2–HPP (Handheld Programmer) or the CPU mode switch to place the CPU in Run Mode.

You can also edit the program during Run Mode. The Run Mode Edits are not bumpless to the outputs. Instead, the CPU ignores the inputs and maintains the outputs in their last state while it accepts the new program information. If an error is found in the new program, then the CPU will turn all the outputs off and enter the Program Mode. This feature is discussed in more detail in Chapter 9.

Normal Run mode scan


Read Inputs
Read Inputs from Specialty I/O
Service Peripherals
Update Clock, Special Relays
Solve the Application Program
Write Outputs
Write Outputs to Specialty I/O
Diagnostics

WARNING: Only authorized personnel fully familiar with all aspects of the application should make changes to the program. Changes during Run Mode become effective immediately. Make sure you thoroughly consider the impact of any changes to minimize the risk of personal injury or damage to equipment.

Read Inputs

The CPU reads the status of all inputs, then stores it in the image register. Input image register locations are designated with an X followed by a memory location. Image register data is used by the CPU when it solves the application program.

Of course, an input may change after the CPU has just read the inputs. Generally, the CPU scan time is measured in milliseconds. If you have an application that cannot wait until the next I/O update, you can use Immediate Instructions. These do not use the status of the input image register to solve the application program. The Immediate instructions immediately read the input status directly from the I/O modules. However, this lengthens the program scan since the CPU has to read the I/O point status again. A complete list of the Immediate instructions is included in Chapter 5.

Service Peripherals and Force I/O

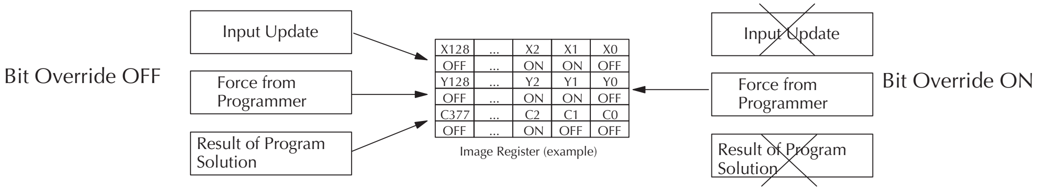
After the CPU reads the inputs from the input modules, it reads any attached peripheral devices. This is primarily a communications service for any attached devices. For example, it would read a programming device to see if any input, output, or other memory type status needs to be modified. There are two basic types of forcing available with the DL06 CPUs:

- Forcing from a peripheral – not a permanent force, good only for one scan
- Bit Override – holds the I/O point (or other bit) in the current state. Valid bits are X, Y, C, T, CT, and S. (These memory types are discussed in more detail later in this chapter).

Regular Forcing — This type of forcing can temporarily change the status of a discrete bit. For example, you may want to force an input on, even though it is really off. This allows you to change the point status that was stored in the image register. This value will be valid until the image register location is written to during the next scan. This is primarily useful during testing situations when you need to force a bit on to trigger another event.

Bit Override — Bit override can be enabled on a point-by-point basis by using AUX 59 from the Handheld Programmer or, by a menu option from within ***Direct***SOFT. Bit override basically disables any changes to the discrete point by the CPU. For example, if you enable bit override for X1, and X1 is off at the time, then the CPU will not change the state of X1. This means that even if X1 comes on, the CPU will not acknowledge the change. So, if you used X1 in the program, it would always be evaluated as **Off** in this case. Of course, if X1 was on when the bit override was enabled, then X1 would always be evaluated as **On**.

There is an advantage available when you use the bit override feature. The regular forcing is not disabled because the bit override is enabled. For example, if you enabled the Bit Override for Y0 and it was off at the time, then the CPU would not change the state of Y0. However, you can still use a programming device to change the status. Now, if you use the programming device to force Y0 on, it will remain on and the CPU will not change the state of Y0. If you then force Y0 off, the CPU will maintain Y0 as off. The CPU will never update the point with the results from the application program or from the I/O update until the bit override is removed. The following diagram shows a brief overview of the bit override feature. Notice the CPU does not update the Image Register when bit override is enabled.



X128	...	X2	X1	X0
OFF	...	ON	ON	OFF
Y128	...	Y2	Y1	Y0
OFF	...	ON	ON	OFF
C377	...	C2	C1	C0
OFF	...	ON	OFF	OFF

WARNING: Only authorized personnel fully familiar with all aspects of the application should make changes to the program. Make sure you thoroughly consider the impact of any changes to minimize the risk of personal injury or damage to equipment.

CPU Bus Communication

It is possible to transfer data to and from the CPU over the CPU bus on the backplane. This data is more than standard I/O point status. This type of communications can only occur on the CPU (local) base. There is a portion of the execution cycle used to communicate with these modules. The CPU performs both read and write requests during this segment.

Update Clock, Special Relays and Special Registers

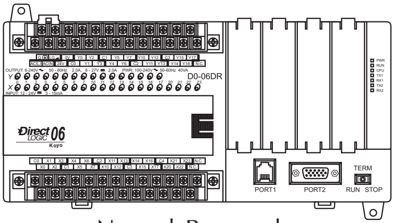
The DL06 CPUs have an internal real-time clock and calendar timer which is accessible to the application program. Special V-memory locations hold this information. This portion of the execution cycle makes sure these locations get updated on every scan. Also, there are several different Special Relays, such as diagnostic relays, for example, that are also updated during this segment.

Solve Application Program

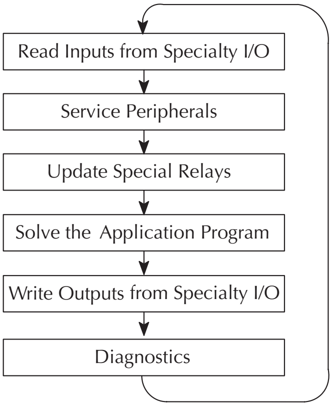
The CPU evaluates each instruction in the application program during this segment of the scan cycle. The instructions define the relationship between the input conditions and the desired output response. The CPU uses the output image register area to store the status of the desired action for the outputs. Output image register locations are designated with a Y followed by a memory location. The actual outputs are updated during the write outputs segment of the scan cycle. There are immediate output instructions available that will update the output points immediately instead of waiting until the write output segment. A complete list of the Immediate instructions is provided in Chapter 5.

The internal control relays (C), the stages (S), and the variable memory (V) are also updated in this segment.

You may recall that you can force various types of points in the system, discussed earlier in this chapter. If any I/O points or memory data have been forced, the output image register also contains this information.



Normal Run mode scan


Read Inputs from Specialty I/O
Service Peripherals
Update Special Relays
Solve the Application Program
Write Outputs from Specialty I/O
Diagnostics

Solve PID Loop Equations

The DL06 CPU can process up to 8 PID loops. The loop calculations are run as a separate task from the ladder program execution, immediately following it. Only loops which have been configured are calculated, and then only according to a built-in loop scheduler. The sample time (calculation interval) of each loop is programmable. Please refer to Chapter 8, PID Loop Operation, for more on the effects of PID loop calculation on the overall CPU scan time.

Write Outputs

Once the application program has solved the instruction logic and constructed the output image register, the CPU writes the contents of the output image register to the corresponding output points. Remember, the CPU also made sure that any forcing operation changes were stored in the output image register, so the forced points get updated with the status specified earlier.

Write Outputs to Specialty I/O

After the CPU updates the outputs in the local and expansion bases, it sends the output point information that is required by any Specialty modules which are installed. Specialty modules have built-in microprocessors which communicate to the CPU via the backplane. Some of these modules can process data. Refer to the specific Specialty module user manual for detailed information.

Diagnostics

During this part of the scan, the CPU performs all system diagnostics and other tasks such as calculating the **scan time** and resetting the **watchdog time**r. There are many different error conditions that are automatically detected and reported by the DL06 PLCs. Appendix B contains a listing of the various error codes.

Probably one of the more important things that occurs during this segment is the **scan tim**e **calculation and watchdog timer control**. The DL06 CPU has a **watchdog timer** that stores the maximum time allowed for the CPU to complete the solve application segment of the scan cycle. If this time is exceeded, the CPU will enter the Program Mode and turn off all outputs. The default value set from the factory is 200 ms. An error is automatically reported. For example, the Handheld Programmer would display the following message "**E003 S/W TIMEOUT**" when the scan overrun occurs.

You can use AUX 53 to view the minimum, maximum, and current scan time. Use AUX 55 to increase or decrease the watchdog timer value.

I/O Response Time

Is Timing Important for Your Application?

I/O response time is the amount of time required for the control system to sense a change in an input point and update a corresponding output point. In the majority of applications, the CPU performs this task in such a short period of time that you may never have to concern yourself with the aspects of system timing. However, some applications do require extremely fast update times. In these cases, you may need to know how to determine the amount of time spent during the various segments of operation.

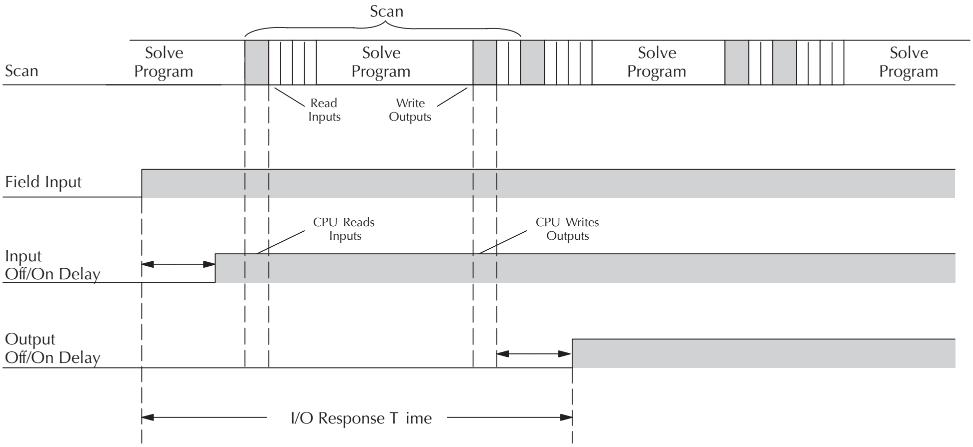
There are four things that can affect the I/O response time.

- The point in the scan cycle when the field input changes states
- Input Off to On delay time
- CPU scan time
- Output Off to On delay time

The next paragraphs show how these items interact to affect the response time.

Normal Minimum I/O Response

The I/O response time is shortest when the input changes just before the Read Inputs portion of the execution cycle. In this case the input status is read, the application program is solved, and the output point gets updated. The following diagram shows an example of the timing for this situation.

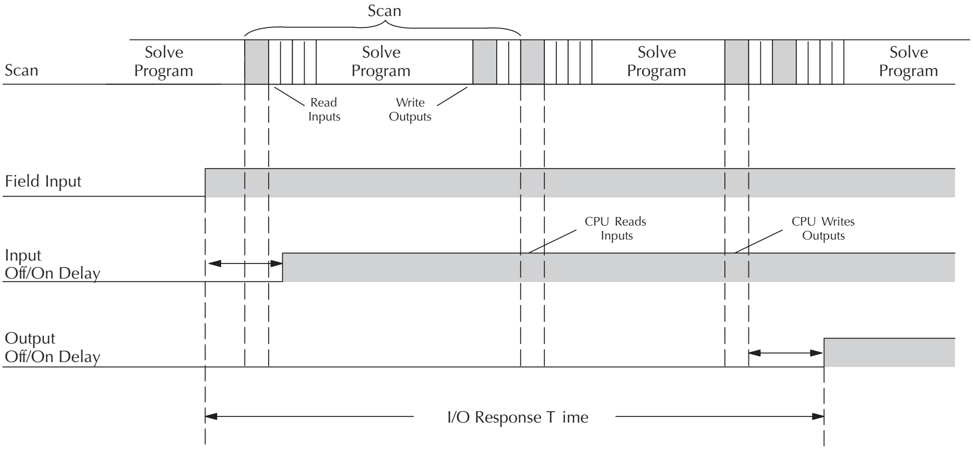


In this case, you can calculate the response time by simply adding the following items:

Input Delay + Scan Time + Output Delay = Response Time

Normal Maximum I/O Response

The I/O response time is longest when the input changes just after the Read Inputs portion of the execution cycle. In this case the new input status is not read until the following scan. The following diagram shows an example of the timing for this situation.



In this case, you can calculate the response time by simply adding the following items:

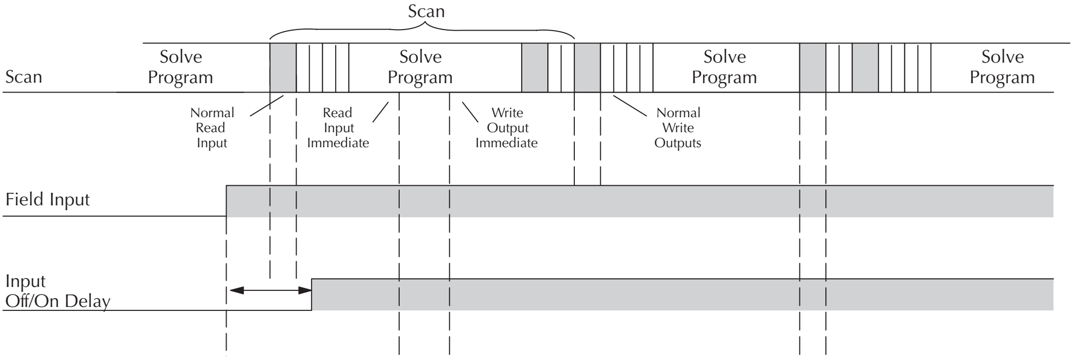
Input Delay +(2 x Scan Time) + Output Delay = Response Time

Improving Response Time

There are a few things you can do to help improve throughput.

- You can choose instructions with faster execution times
- You can use immediate I/O instructions (which update the I/O points during the program execution)
- You can use the HSIO Mode 50 Pulse Catch features designed to operate in high-speed environments. See Appendix E for details on using this feature.
- You can change Mode 60 filter to 0 msec for X0, X1, X2, and X3.

Of these three things the Immediate I/O instructions are probably the most important and most useful. The following example shows how an immediate input instruction and immediate output instruction would affect the response time.



In this case, you can calculate the response time by simply adding the following items.

Input Delay + Instruction Execution Time + Output Delay = Response Time

The instruction execution time would be calculated by adding the time for the immediate input instruction, the immediate output instruction, and any other instructions in between the two.

***NOTE**: Even though the immediate instruction reads the most current status from I/O, it only uses the results to solve that one instruction. It does not use the new status to update the image register. Therefore, any regular instructions that follow will still use the image register values. Any immediate instructions that follow will access the I/O again to update the status.*

CPU Scan Time Considerations

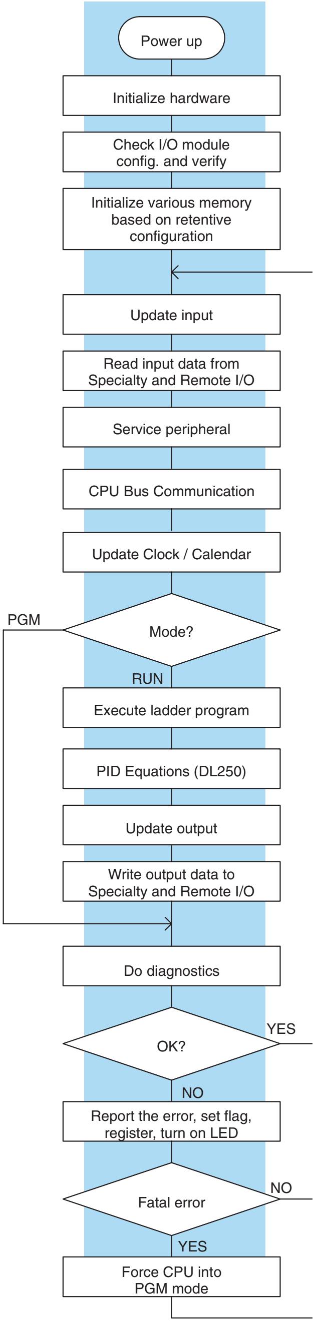
The scan time covers all the cyclical tasks that are performed by the operating system. You can use *Direct*SOFT or the Handheld Programmer to display the minimum, maximum, and current scan times that have occurred since the previous Program Mode to Run Mode transition. This information can be very important when evaluating the performance of a system. As we've shown previously there are several segments that make up the scan cycle. Each of these segments requires a certain amount of time to complete. Of all the segments, the following are the most important:

- Input Update
- Peripheral Service
- Program Execution
- Output Update
- Timed Interrupt Execution

The one you have the most control over is the amount of time it takes to execute the application program. This is because different instructions take different amounts of time to execute. So, if you think you need a faster scan, then you can try to choose faster instructions.

Your choice of I/O type and peripheral devices can also affect the scan time. However, these things are usually dictated by the application.

The following paragraphs provide some general information on how much time some of the segments can require.


Power up
Initialize hardware
Check I/O module config. and verify
Initialize various memory based on retentive configuration
Update input
Read input data from Specialty and Remote I/O
Service peripheral
CPU Bus Communication
Update Clock / Calendar
PGM
Mode?
RUN
Execute ladder program
PID Equations (DL250)
Update output
Write output data to Specialty and Remote I/O
Do diagnostics
YES
OK?
NO
Report the error, set flag, register, turn on LED
NO
Fatal error
YES
Force CPU into PGM mode

Reading Inputs

The time required during each scan to read the input status of built-in inputs is 52.6 µs. Don't confuse this with the I/O response time that was discussed earlier.

Writing Outputs

The time required to write the output status of built-in outputs is 41.1 µS. Don't confuse this with the I/O response time that was discussed earlier.

Service Peripherals

Communication requests can occur at any time during the scan, but the CPU only logs the requests for service until the Service Peripherals portion of the scan. The CPU does not spend any time on this if there are no peripherals connected.

To Log Request (anytime)		DL06
Nothing Connected	Min. & Max	0µs
Port 1	Send Min. / Max.	5.8/11.8 µs
	Rec. Min. / Max.	12.5/25.2 µs
Port 2	Send Min. / Max.	6.2/14.3 µs
	Rec. Min. / Max.	14.2/31.9 µs
LCD	Min. / Max.	4.8/49.2 µs

During the Service Peripherals portion of the scan, the CPU analyzes the communications request and responds as appropriate. The amount of time required to service the peripherals depends on the content of the request.

To Service Request DL06	DL06
Minimum	9 µs
Run Mode Max.	412 µs
Program Mode Max.	2.5 second

CPU Bus Communication

Some specialty modules can also communicate directly with the CPU via the CPU bus. During this portion of the cycle the CPU completes any CPU bus communications. The actual time required depends on the type of modules installed and the type of request being processed.

Update Clock/Calendar, Special Relays, Special Registers

The clock, calendar, and special relays are updated and loaded into special V-memory locations during this time. This update is performed during both Run and Program Modes.

Modes		DL06
Program Mode	Minimum	12.0 µs
	Maximum	12.0 µs
Run Mode	Minimum	20.0 µs
	Maximum	27.0 µs

NOTE: *The Clock/Calendar is updated while there is energy on the super-capacitor. If the super-capacitor is discharged, the real time and date is lost.*

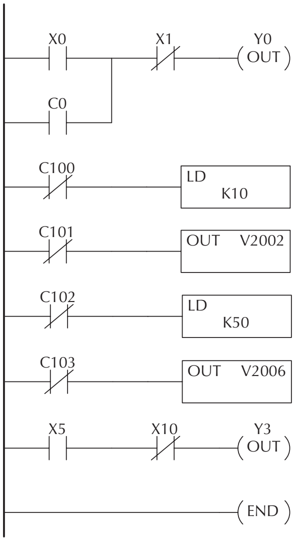
Application Program Execution

The CPU processes the program from address 0 to the END instruction. The CPU executes the program left to right and top to bottom. As each rung is evaluated the appropriate image register or memory location is updated. The time required to solve the application program depends on the type and number of instructions used, and the amount of execution overhead.

Just add the execution times for all the instructions in your program to determine to total execution time. Appendix C provides a complete list of the instruction execution times for the DL06 Micro PLC. For example, the execution time for running the program shown below is calculated as follows:

Instruction	Time
STR X0	.67 µs
OR C0	.51 µs
ANDN X1	.51 µs
OUT Y0	1.82 µs
STRN C100	.67 µs
LD K10	9.00 µs
STRN C101	.67 µs
OUT V2002	9.3 µs
STRN C102	.67 µs
LD K50	9.00 µs
STRN C103	.67 µs
OUT V2006	1.82 µs
STR X5	.67 µs
ANDN X10	.51 µs
OUT Y3	1.82 µs
END	12.80 µs
SUBTOTAL	51.11 µs

Overhead	DL06
Minimum	746.2 µs
Maximum	4352.4 µs

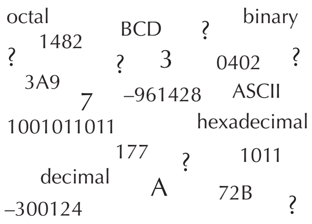


TOTAL TIME = (Program execution time + Overhead) x 1.18

The program above takes only 51.11 µs to execute during each scan. The DL06 spends 0.18ms on internal timed interrupt management, for every 1ms of instruction time. The total scan time is calculated by adding the program execution time to the overhead (shown above) and multiplying the result (ms) by 1.18. **Overhead** includes all other housekeeping and diagnostic tasks. The scan time will vary slightly from one scan to the next, because of fluctuation in overhead tasks.

Program Control Instructions — the DL06 CPUs offer additional instructions that can change the way the program executes. These instructions include FOR/NEXT loops, Subroutines, and Interrupt Routines. These instructions can interrupt the normal program flow and affect the program execution time. Chapter 5 provides detailed information on how these different types of instructions operate.

PLC Numbering Systems



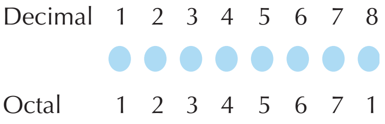
If you are a new PLC user or are using *AutomationDirect* PLCs for the first time, please take a moment to study how our PLCs use numbers. You'll find that each PLC manufacturer has their own conventions on the use of numbers in their PLCs. We want to take just a moment to familiarize you with how numbers are used in *AutomationDirect* PLCs. The information you learn here applies to all of our PLCs.

As any good computer does, PLCs store and manipulate numbers in binary form - just ones and zeros. So, why do we have to deal with numbers in so many different forms? Numbers have meaning, and some representations are more convenient than others for particular purposes. Sometimes we use numbers to represent a size or amount of something. Other numbers refer to locations or addresses, or to time. In science we attach engineering units to numbers to give a particular meaning (see Appendix I for numbering system details).

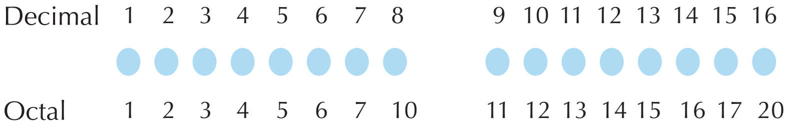
PLC Resources

PLCs offer a fixed amount of resources, depending on the model and configuration. We use the word **resources** to include variable memory (V-memory), I/O points, timers, counters, etc. Most modular PLCs allow you to add I/O points in groups of eight. In fact, all the resources of our PLCs are counted in octal. It's easier for computers to count in groups of eight than ten, because eight is an even power of 2 (see Appendix I for more details).

Octal means simply counting in groups of eight things at a time. In the figure to the right, there are eight circles. The quantity in decimal is 8, but in octal it is 10 (8 and 9 are not valid in octal). In octal, 10 means 1 group of 8 plus 0 (no individuals)

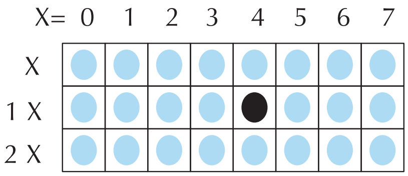


In the figure below, we have two groups of eight circles. Counting in octal we have 20 items, meaning 2 groups of eight, plus 0 individuals Don't say "twenty", say "two–zero octal". This makes a clear distinction between number systems.


Decimal 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Octal 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20

After counting PLC resources, it's time to access PLC resources (there's a difference). The CPU instruction set accesses resources of the PLC using octal addresses. Octal addresses are the same as octal quantities, except they start counting at zero. The number zero is significant to a computer, so we don't skip it.

Our circles are in an array of square containers to the right. To access a resource, our PLC instruction will address its location using the octal references shown. If these were counters, **CT14** would access the black circle location.



V–Memory

Variable memory (V-memory) stores data for the ladder program and for configuration settings. V-memory locations and V-memory addresses are the same thing, and are numbered in octal. For example, V2073 is a valid location, while V1983 is not valid (9 and 8 are not valid octal digits).

Each V-memory location is one data word wide, meaning 16 bits. For configuration registers, our manuals will show each bit of a V-memory word. The least significant bit (LSB) will be on the right, and the most significant bit (MSB) on the left. We use the word "significant", referring to the relative binary weighting of the bits.

V-memory address (octal)	V-memory data (binary)															
	MSB															LSB
V2017	0	1	0	0	1	1	1	0	0	0	1	0	1	0	0	1

V-memory data is 16-bit binary, but we rarely program the data registers one bit at a time. We use instructions or viewing tools that let us work with decimal, octal, and hexadecimal numbers. All these are converted and stored as binary for us.

A frequently-asked question is "How do I tell if a number is octal, BCD, or hex?" The answer is that we usually cannot tell just by looking at the data ... but it does not really matter. What matters is, the source or mechanism which writes data into a V-memory location and the thing which later reads it must both use the same data type (i.e., octal, hex, binary, or whatever). The V-memory location is just a storage box ... that's all. It does not convert or move the data on its own.

Binary-Coded Decimal Numbers

BCD number	4	9	3	6
V-memory storage	0 1 0 0	1 0 0 1	0 0 1 1	0 1 1 0

Since humans naturally count in decimal (10 fingers, 10 toes), we prefer to enter and view PLC data in decimal as well. However, computers are more efficient in using pure binary numbers. A compromise solution between the two is Binary-Coded Decimal (BCD) representation. A BCD digit ranges from 0 to 9, and is stored as four binary bits (a nibble). This permits each V-memory location to store four BCD digits, with a range of decimal numbers from 0000 to 9999.

In a pure binary sense, a 16-bit word can represent numbers from 0 to 65535. In storing BCD numbers, the range is reduced to only 0 to 9999. Many math instructions use Binary-Coded Decimal (BCD) data, and ***Direct***SOFT and the handheld programmer allow us to enter and view data in BCD.

Hexadecimal Numbers

Hexadecimal numbers are similar to BCD numbers, except they utilize all possible binary values in each 4-bit digit. They are base-16 numbers so we need 16 different digits. To extend our decimal digits 0 through 9, we use A through F as shown.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

A 4-digit hexadecimal number can represent all 65536 values in a V-memory word. The range is from 0000 to FFFF (hex). PLCs often need this full range for sensor data, etc. Hexadecimal is just a convenient way for humans to view full binary data.

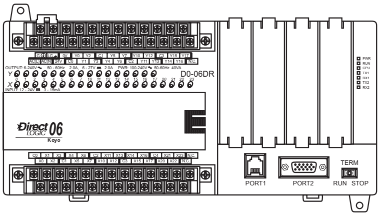
Hexadecimal number	A	7	F	4
V-memory storage	1 0 1 0	0 1 1 1	1 1 1 1	0 1 0 0

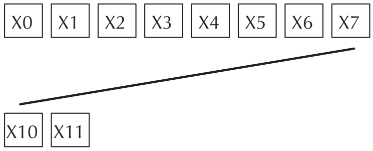
Memory Map

With any PLC system, you generally have many different types of information to process. This includes input device status, output device status, various timing elements, parts counts, etc. It is important to understand how the system represents and stores the various types of data. For example, you need to know how the system identifies input points, output points, data words, etc. The following paragraphs discuss the various memory types used in DL06 Micro PLCs. A memory map overview for the CPU follows the memory descriptions.

Octal Numbering System

All memory locations and resources are numbered in Octal (base 8). For example, the diagram shows how the octal numbering system works for the discrete input points. Notice the octal system does not contain any numbers with the digits 8 or 9.

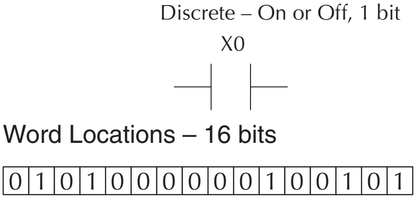




Discrete and Word Locations

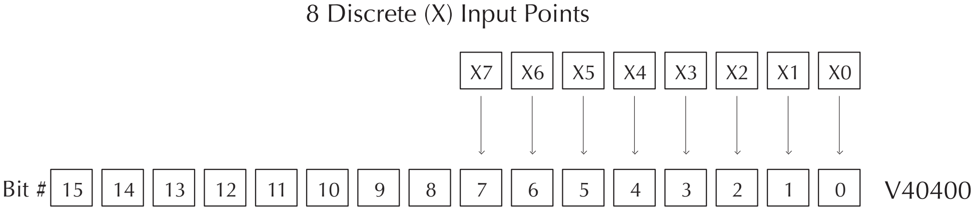
As you examine the different memory types, you'll notice two types of memory in the DL06, discrete and word memory. Discrete memory is one bit that can be either a 1 or a 0. Word memory is referred to as V-memory (variable) and is a 16-bit location normally used to manipulate data/ numbers, store data/numbers, etc.

Some information is automatically stored in V-memory. For example, the timer current values are stored in V-memory.



V-memory Locations for Discrete Memory Areas

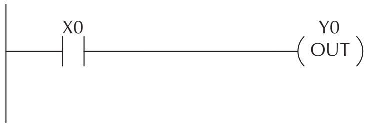
The discrete memory area is for inputs, outputs, control relays, special relays, stages, timer status bits and counter status bits. However, you can also access the bit data types as a V-memory word. Each V-memory location contains 16 consecutive discrete locations. For example, the following diagram shows how the X input points are mapped into V-memory locations.



These discrete memory areas and their corresponding V-memory ranges are listed in the memory area table for DL06 Micro PLCs on the following pages.

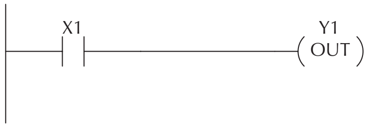
Input Points (X Data Type)

The discrete input points are noted by an X data type. There are 20 discrete input points and 256 discrete input addresses available with DL06 CPUs. In this example, the output point Y0 will be turned on when input X0 energizes.



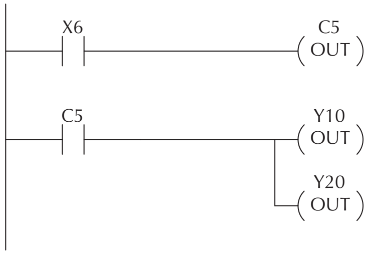
Output Points (Y Data Type)

The discrete output points are noted by a Y data type. There are 16 discrete outputs and 256 discrete output addresses available with DL06 CPUs. In this example, output point Y1 will be turned on when input X1 energizes.



Control Relays (C Data Type)

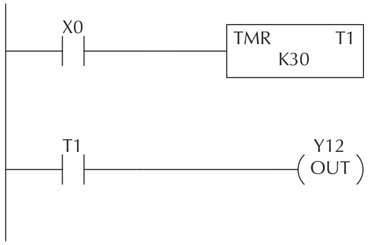
Control relays are discrete bits normally used to control the user program. The control relays do not represent a real world device, that is, they cannot be physically tied to switches, output coils, etc. There are 1024 control relays internal to the CPU. Because of this, control relays can be programmed as discrete inputs or discrete outputs. These locations are used in programming the discrete memory locations (C) or the corresponding word location which contains 16 consecutive discrete locations.



In this example, memory location C5 will energize when input X6 turns on. The second rung shows a simple example of how to use a control relay as an input.

Timers and Timer Status Bits (T Data Type)

There are 256 timers available in the CPU. Timer status bits reflect the relationship between the current value and the preset value of a specified timer. The timer status bit will be on when the current value is equal or greater than the preset value of a corresponding timer.

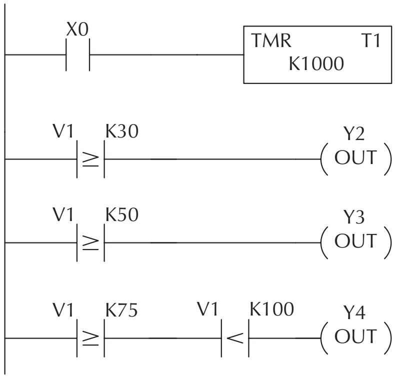


When input X0 turns on, timer T1 will start. When the timer reaches the preset of 3 seconds (K of 30) timer status contact T1 turns on. When T1 turns on, output Y12 turns on. Turning off X0 resets the timer.

Timer Current Values (V Data Type)

As mentioned earlier, some information is automatically stored in V-memory. This is true for the current values associated with timers. For example: V0 holds the current value for Timer 0; V1 holds the current value for Timer 1; and so on. These can also be designated as TA0 (Timer Accumulated) for Timer 0, and TA1 for Timer 1.

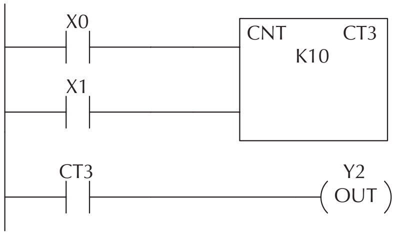
The primary reason for this is programming flexibility. The example shows how you can use relational contacts to monitor several time intervals from a single timer.



Counters and Counter Status Bits (CT Data type)

There are 128 counters available in the CPU. Counter status bits that reflect the relationship between the current value and the preset value of a specified counter. The counter status bit will be on when the current value is equal to or greater than the preset value of a corresponding counter.

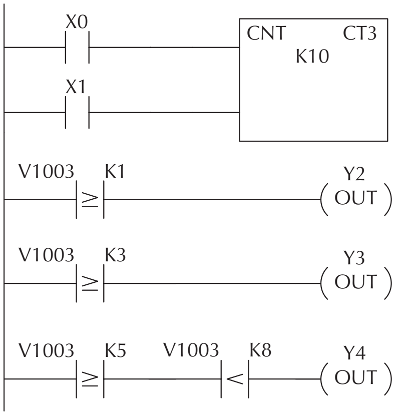
Each time contact X0 transitions from off to on, the counter increments by one. (If X1 comes on, the counter is reset to zero.) When the counter reaches the preset of 10 counts (K of 10) counter status contact CT3 turns on. When CT3 turns on, output Y2 turns on.



Counter Current Values (V Data Type)

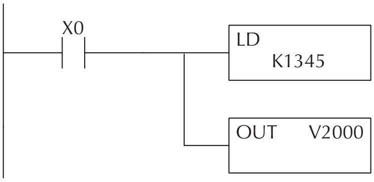
Just like the timers, the counter current values are also automatically stored in V-memory. For example, V1000 holds the current value for Counter CT0, V1001 holds the current value for Counter CT1, etc. These can also be designated as CTA0 (Counter Accumulated) for Counter 0 and CTA01 for Counter 1.

The primary reason for this is programming flexibility. The example shows how you can use relational contacts to monitor the counter values.

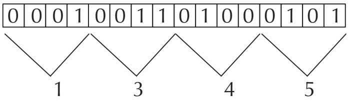


Word Memory (V Data Type)

Word memory is referred to as V-memory (variable) and is a 16-bit location normally used to manipulate data/numbers, store data/numbers, etc. Some information is automatically stored in V-memory. For example, the timer current values are stored in V-memory. The example shows how a four-digit BCD constant is loaded into the accumulator and then stored in a V-memory location.



Word Locations – 16 bits

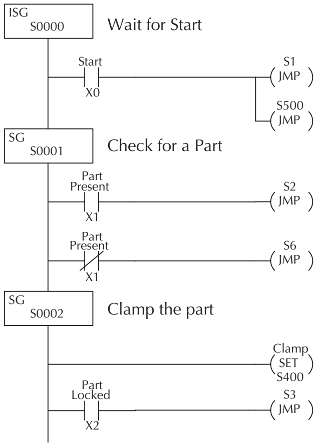


Stages (S Data type)

Stages are used in RLL^{PLUS} programs to create a structured program, similar to a flowchart. Each program Stage denotes a program segment. When the program segment, or Stage, is active, the logic within that segment is executed. If the Stage is off, or inactive, the logic is not executed and the CPU skips to the next active Stage. (See Chapter 7 for a more detailed description of RLL^{PLUS} programming.)

Each Stage also has a discrete status bit that can be used as an input to indicate whether the Stage is active or inactive. If the Stage is active, then the status bit is on. If the Stage is inactive, then the status bit is off. This status bit can also be turned on or off by other instructions, such as the SET or RESET instructions. This allows you to easily control stages throughout the program.

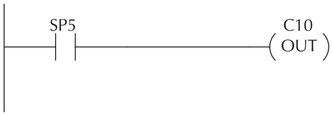
Ladder Representation



Special Relays (SP Data Type)

Special relays are discrete memory locations with pre-defined functionality. There are many different types of special relays. For example, some aid in program development, others provide system operating status information, etc. Appendix D provides a complete listing of the special relays.

In this example, control relay C10 will energize for 50 ms and de-energize for 50 ms because SP5 is a pre–defined relay that will be on for 50 ms and off for 50 ms.



SP4: 1 second clock
SP5: 100 ms clock
SP6: 50 ms clock

DL06 System V-memory

System Parameters and Default Data Locations (V Data Type)

The DL06 PLCs reserve several V-memory locations for storing system parameters or certain types of system data. These memory locations store things like the error codes, High-Speed I/O data, and other types of system setup information.

System V-memory	Description of Contents	Default Values / Ranges	Read Only R/W
V700-V707	Sets the V-memory location for option card in slot 1	N/A	R/W
V710-V717	Sets the V-memory location for option card in slot 2	N/A	R/W
V720-V727	Sets the V-memory location for option card in slot 3	N/A	R/W
V730-V737	Sets the V-memory location for option card in slot 4	N/A	R/W
V3630–V3707	The default location for multiple preset values for UP/DWN and UP Counter 1 or pulse catch function	N/A	R/W
V3710-V3767	The default location for multiple preset values for UP/DWN and UP Counter 2	N/A	R/W
V7620	DV-1000 Sets the V-memory location that contains the value	V0 – V3760	R/W
V7621	DV-1000 Sets the V-memory location that contains the message	V0 – V3760	R/W
V7622	DV-1000 Sets the total number (1 – 32) of V-memory locations to be displayed	1 - 32	R/W
V7623	DV-1000 Sets the V-memory location containing the numbers to be displayed	V0 – V3760	R/W
V7624	DV-1000 Sets the V-memory location that contains the character code to be displayed	V0 – V3760	R/W
V7625	DV-1000 Contains the function number that can be assigned to each key	V-memory for X, Y, or C	R/W
V7626	DV-1000 Powerup operational mode	0,1, 2, 3, 12	R/W
V7627	Change preset value	0000 to 9999	R/W
V7630	Starting location for the multi–step presets for channel 1. The default value is 3630, which indicates the first value should be obtained from V3630. Since there are 24 presets available, the default range is V3630 – V3707. You can change the starting point if necessary.	Default: V3630 Range: V0- V3710	R/W
V7631	Starting location for the multi–step presets for channel 2. The default value is 3710, which indicates the first value should be obtained from V3710. Since there are 24 presets available, the default range is V3710 – V3767. You can change the starting point if necessary.	Default: V3710 Range: V0- V3710	R/W
V7632	Setup Register for Pulse Output	N/A	R/W
V7633	Sets the desired function code for the high speed counter, interrupt, pulse catch, pulse train, and input filter. This location can also be used to set the power-up in Run Mode option.	Default: 0060 Lower Byte Range: Range: 10 – Counter 20 – Quadrature 30 – Pulse Out 40 – Interrupt 50 – Pulse Catch 60 – Filtered discrete In. Upper Byte Range: Bits 8–11, 14, 15: Unused, Bit 13: Power–up in RUN, only if Mode Switch is in TERM position. Bit 12 is used to enable the low battery indications.	R/W
V7634	X0 Setup Register for High-Speed I/O functions for input X0	Default: 1006	R/W
V7635	X1 Setup Register for High-Speed I/O functions for input X1	Default: 1006	R/W
V7636	X2 Setup Register for High-Speed I/O functions for input X2	Default: 1006	R/W
V7637	X3 Setup Register for High-Speed I/O functions for input X3	Default: 1006	R/W
V7640	PID Loop table beginning address	V1200 - V7377 V10000-V17777	R/W
V7641	Number of PID loops enabled	1-8	R/W
V7642	Error Code - PID Loop Table		R
V7643-V7646	DirectSoft I-Box instructions work area		R
V7647	Timed Interrupt		R/W
V7653	Port 2: Terminate code setting Non-procedure		R/W
V7655	Port 2: Setup for the protocol, time-out, and the response delay time		R/W

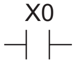
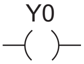
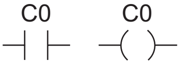
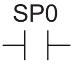
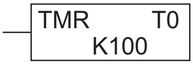
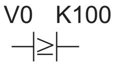
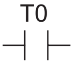
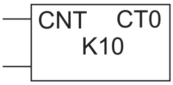
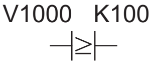
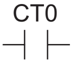
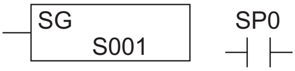
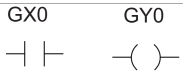
System V-memory	Description of Contents	Default Values / Ranges	Read Only R/W
V7656	Port 2: Setup for the station number, baud rate, STOP bit, and parity		R/W
V7657	Port 2: Setup completion code used to notify the completion of the parameter setup	0400h reset port 2	R/W
V7660	Scan control setup: Keeps the scan control mode		R/W
V7661	Setup timer over counter		R
V7662–V7710	Reserved		R/W
V7711-V7717	DirectSOFT I-Box instructions work area		R
V7720–V7722	Locations for DV–1000 operator interface parameters		R/W
V7720	location for DV-1000 operation interface Titled Timer preset value pointer		R/W
V7721	DV-1000: Title Counter preset value pointer		R/W
V7722	DV-1000: Hibyte-Titled, Lobyte-Timer preset block size		R/W
V7723–V7725	DirectSOFT I-Box instructions work area		R
V7726	Reserved		R/W
V7727	Version No		R
V7730	D0-DCM Module Slot1 Auto Reset Timeout		R/W
V7731	D0-DCM Module Slot2 Auto Reset Timeout		R/W
V7732	D0-DCM Module Slot3 Auto Reset Timeout		R/W
V7733	D0-DCM Module Slot4 Auto Reset Timeout		R/W
V7734-V7737	Reserved		R/W
V7740	Port 2: Communication Auto Reset Timer Setup	Default: 3030	R/W
V7741	Reserved		R/W
V7742	LCD Various LCD setting flags		R/W
V7743	V Memory address in which the default display message is stored as set		R/W
V7744-V7746	Reserved		R/W
V7747	Location contains a 10 ms counter (0-99). This location increments once every 10 ms		R
V7750	Reserved		R/W
V7751	Fault Message Error Code		R
V7752	I/O Configuration Error: stores the module ID code for the module that does not the current configuration		R
V7753	I/O Configuration Error: stores the module ID code		R
V7754	I/O Configuration Error: identifies the base and slot number		R
V7755	Error code — stores the fatal error code		R
V7756	Error code — stores the major error code		R
V7757	Error code — stores the minor error code		R
V7760–V7762	Reserved		R/W
V7763	Program address where syntax error exists		R
V7764	Syntax error code		R
V7765	Scan counter — stores the total number of scan cycles that have occurred since the last Program Mode to Run Mode transition (in decimal)		R
V7766	Contains the number of seconds on the clock (00-59)		R
V7767	Contains the number of minutes on the clock (00-59)		R
V7770	Contains the number of hours on the clock (00-23)		R
V7771	Contains the day of the week (Mon., Tues., Wed., etc.)		R
V7772	Contains the day of the month (01, 02, etc.)		R
V7773	Contains the month (01 to 12)		R
V7774	Contains the year (00 to 99)		R
V7775	Scan — stores the current scan time (milliseconds)		R
V7776	Scan — stores the minimum scan time that has occurred since the last Program Mode to Run Mode transition (milliseconds)		R
V7777	Scan — stores the maximum scan rate since the last power cycle (milliseconds)		R

DL06 Aliases

An alias is an alternate way of referring to certain memory types, such as timer/counter current values, V-memory locations for I/O points, etc., which simplifies understanding the memory address. The use of the alias is optional, but some users may find the alias to be helpful when developing a program. The table below shows how the aliases can be used.

DL06 Aliases		
Address Start	**Alias Start**	**Example**
V0	TA0	V0 is the timer accumulator value for timer 0, therefore, its alias is TA0. TA1 is the alias for V1, etc..
V1000	CTA0	V1000 is the counter accumulator value for counter 0, therefore, its alias is CTA0. CTA1 is the alias for V1001, etc.
V40000	VGX	V40000 is the word memory reference for discrete bits GX0 through GX17, therefore, its alias is VGX0. V40001 is the word memory reference for discrete bits GX20 through GX 37, therefore, its alias is VGX20.
V40200	VGY	V40200 is the word memory reference for discrete bits GY0 through GY17, therefore, its alias is VGY0. V40201 is the word memory reference for discrete bits GY20 through GY 37, therefore, its alias is VGY20.
V40400	VX0	V40400 is the word memory reference for discrete bits X0 through X17, therefore, its alias is VX0. V40401 is the word memory reference for discrete bits X20 through X37, therefore, its alias is VX20.
V40500	VY0	V40500 is the word memory reference for discrete bits Y0 through Y17, therefore, its alias is VY0. V40501 is the word memory reference for discrete bits Y20 through Y37, therefore, its alias is VY20.
V40600	VC0	V40600 is the word memory reference for discrete bits C0 through C17, therefore, its alias is VC0. V40601 is the word memory reference for discrete bits C20 through C37, therefore, its alias is VC20.
V41000	VS0	V41000 is the word memory reference for discrete bits S0 through S17, therefore, its alias is VS0. V41001 is the word memory reference for discrete bits S20 through S37, therefore, its alias is VS20.
V41100	VT0	V41100 is the word memory reference for discrete bits T0 through T17, therefore, its alias is VT0. V41101 is the word memory reference for discrete bits T20 through T37, therefore, its alias is VT20.
V41140	VCT0	V41140 is the word memory reference for discrete bits CT0 through CT17, therefore, its alias is VCT0. V41141 is the word memory reference for discrete bits CT20 through CT37, therefore, its alias is VCT20.
V41200	VSP0	V41200 is the word memory reference for discrete bits SP0 through SP17, therefore, its alias is VSP0. V41201 is the word memory reference for discrete bits SP20 through SP37, therefore, its alias is VSP20.

DL06 Memory Map

Memory Type	Discrete Memory Reference (octal)	Word Memory Reference (octal)	Decimal	Symbol
Input Points	X0 – X777	V40400 - V40437	512	
Output Points	Y0 – Y777	V40500 – V40537	512	
Control Relays	C0 – C1777	V40600 - V40677	1024	
Special Relays	SP0 – SP777	V41200 – V41237	512	
Timers	T0 – T377	V41100 – V41117	256	
Timer Current Values	None	V0 – V377	256	
Timer Status Bits	T0 – T377	V41100 – V41117	256	
Counters	CT0 – CT177	V41140 – V41147	128	
Counter Current Values	None	V1000 – V1177	128	
Counter Status Bits	CT0 – CT177	V41140 – V41147	128	
Data Words (See Appendix F)	None	V400-V677 V1200 – V7377 V10000 - V17777	192 3200 4096	None specific, used with many instructions.
Data Words EEPROM (See Appendix F)	None	V7400 – V7577	128	None specific, used with many instructions. May be non-volatile if MOV inst. is used. Data can be rewritten to EEPROM at least 100,000 times before it fails.
Stages	S0 – S1777	V41000 – V41077	1024	
Remote I/O (future use) (See Note 1)	GX0-GX3777 GY0-GY3777	V40000-V40177 V40200-V40377	2048 2048	
System parameters	None	V700-V777 V7600 – V7777 V36000-V37777	64 128 1024	None specific, used for various purposes

NOTE 1: *This area can be used for additional Data Words.*
NOTE 2: *The DL06 systems have 20 fixed discrete inputs and 16 fixed discrete outputs, but the total can be increased by up to 64 inputs or 64 outputs, or a combination of both.*

X Input/Y Output Bit Map

This table provides a listing of individual input and output points associated with each V-memory address bit for the DL06's twenty integrated physical inputs and 16 integrated physical outputs in addition to up to 64 inputs and 64 outputs for option cards. Actual available references are X0 to X777 (V40400 – V40437) and Y0 to Y777 (V40500 - V40537).

MSB	DL06 Input (X) and Output (Y) Points														LSB	X Input	Y Output
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	Address
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V40400	V40500
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V40401	V40501
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V40402	V40502
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V40403	V40503
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V40404	V40504
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V40405	V40505
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V40406	V40506
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V40407	V40507
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200	V40410	V40510
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220	V40411	V40511
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240	V40412	V40512
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260	V40413	V40513
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300	V40414	V40514
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320	V40415	V40515
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340	V40416	V40516
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360	V40417	V40517
417	416	415	414	413	412	411	410	407	406	405	404	403	402	401	400	V40420	V40520
437	436	435	434	433	432	431	430	427	426	425	424	423	422	421	420	V40421	V40521
457	456	455	454	453	452	451	450	447	446	445	444	443	442	441	440	V40422	V40522
477	476	475	474	473	472	471	470	467	466	465	464	463	462	461	460	V40423	V40523
517	516	515	514	513	512	511	510	507	506	505	504	503	502	501	500	V40424	V40524
537	536	535	534	533	532	531	530	527	526	525	524	523	522	521	520	V40425	V40525
557	556	555	554	553	552	551	550	547	546	545	544	543	542	541	540	V40426	V40526
577	576	575	574	573	572	571	570	567	566	565	564	563	562	561	560	V40427	V40527
617	616	615	614	613	612	611	610	607	606	605	604	603	602	601	600	V40430	V40530
637	636	635	634	633	632	631	630	627	626	625	624	623	622	621	620	V40431	V40531
657	656	655	654	653	652	651	650	647	646	645	644	643	642	641	640	V40432	V40532
677	676	675	674	673	672	671	670	667	666	665	664	663	662	661	660	V40433	V40533
717	716	715	714	713	712	711	710	707	706	705	704	703	702	701	700	V40434	V40534
737	736	735	734	733	732	731	730	727	726	725	724	723	722	721	720	V40435	V40535
757	756	755	754	753	752	751	750	747	746	745	744	743	742	741	740	V40436	V40536
777	776	775	774	773	772	771	770	767	766	765	764	763	762	761	760	V40437	V40537

Stage Control/Status Bit Map

This table provides a listing of individual Stage control bits associated with each V-memory address bit.

MSB	DL06 Stage (S) Control Bits														LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V41000
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V41001
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V41002
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V41003
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V41004
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V41005
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V41006
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V41007
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200	V41010
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220	V41011
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240	V41012
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260	V41013
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300	V41014
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320	V41015
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340	V41016
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360	V41017
417	416	415	414	413	412	411	410	407	406	405	404	403	402	401	400	V41020
437	436	435	434	433	432	431	430	427	426	425	424	423	422	421	420	V41021
457	456	455	454	453	452	451	450	447	446	445	444	443	442	441	440	V41022
477	476	475	474	473	472	471	470	467	466	465	464	463	462	461	460	V41023
517	516	515	514	513	512	511	510	507	506	505	504	503	502	501	500	V41024
537	536	535	534	533	532	531	530	527	526	525	524	523	522	521	520	V41025
557	556	555	554	553	552	551	550	547	546	545	544	543	542	541	540	V41026
577	576	575	574	573	572	571	570	567	566	565	564	563	562	561	560	V41027
617	616	615	614	613	612	611	610	607	606	605	604	603	602	601	600	V41030
637	636	635	634	633	632	631	630	627	626	625	624	623	622	621	620	V41031
657	656	655	654	653	652	651	650	647	646	645	644	643	642	641	640	V41032
677	676	675	674	673	672	671	670	667	666	665	664	663	662	661	660	V41033
717	716	715	714	713	712	711	710	707	706	705	704	703	702	701	700	V41034
737	736	735	734	733	732	731	730	727	726	725	724	723	722	721	720	V41035
757	756	755	754	753	752	751	750	747	746	745	744	743	742	741	740	V41036
777	776	775	774	773	772	771	770	767	766	765	764	763	762	761	760	V41037

This table is continued on the next page.

MSB	DL06 Stage (S) Control Bits (cont'd)														LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1017	1016	1015	1014	1013	1012	1011	1010	1007	1006	1005	1004	1003	1002	1001	1000	V41040
1037	1036	1035	1034	1033	1032	1031	1030	1027	1026	1025	1024	1023	1022	1021	1020	V41041
1057	1056	1055	1054	1053	1052	1051	1050	1047	1046	1045	1044	1043	1042	1041	1040	V41042
1077	1076	1075	1074	1073	1072	1071	1070	1067	1066	1065	1064	1063	1062	1061	1060	V41043
1117	1116	1115	1114	1113	1112	1111	1110	1107	1106	1105	1104	1103	1102	1101	1100	V41044
1137	1136	1135	1134	1133	1132	1131	1130	1127	1126	1125	1124	1123	1122	1121	1120	V41045
1157	1156	1155	1154	1153	1152	1151	1150	1147	1146	1145	1144	1143	1142	1141	1140	V41046
1177	1176	1175	1174	1173	1172	1171	1170	1167	1166	1165	1164	1163	1162	1161	1160	V41047
1217	1216	1215	1214	1213	1212	1211	1210	1207	1206	1205	1204	1203	1202	1201	1200	V41050
1237	1236	1235	1234	1233	1232	1231	1230	1227	1226	1225	1224	1223	1222	1221	1220	V41051
1257	1256	1255	1254	1253	1252	1251	1250	1247	1246	1245	1244	1243	1242	1241	1240	V41052
1277	1276	1275	1274	1273	1272	1271	1270	1267	1266	1265	1264	1263	1262	1261	1260	V41053
1317	1316	1315	1314	1313	1312	1311	1310	1307	1306	1305	1304	1303	1302	1301	1300	V41054
1337	1336	1335	1334	1333	1332	1331	1330	1327	1326	1325	1324	1323	1322	1321	1320	V41055
1357	1356	1355	1354	1353	1352	1351	1350	1347	1346	1345	1344	1343	1342	1341	1340	V41056
1377	1376	1375	1374	1373	1372	1371	1370	1367	1366	1365	1364	1363	1362	1361	1360	V41057
1417	1416	1415	1414	1413	1412	1411	1410	1407	1406	1405	1404	1403	1402	1401	1400	V41060
1437	1436	1435	1434	1433	1432	1431	1430	1427	1426	1425	1424	1423	1422	1421	1420	V41061
1457	1456	1455	1454	1453	1452	1451	1450	1447	1446	1445	1444	1443	1442	1441	1440	V41062
1477	1476	1475	1474	1473	1472	1471	1470	1467	1466	1465	1464	1463	1462	1461	1460	V41063
1517	1516	1515	1514	1513	1512	1511	1510	1507	1506	1505	1504	1503	1502	1501	1500	V41064
1537	1536	1535	1534	1533	1532	1531	1530	1527	1526	1525	1524	1523	1522	1521	1520	V41065
1557	1556	1555	1554	1553	1552	1551	1550	1547	1546	1545	1544	1543	1542	1541	1540	V41066
1577	1576	1575	1574	1573	1572	1571	1570	1567	1566	1565	1564	1563	1562	1561	1560	V41067
1617	1616	1615	1614	1613	1612	1611	1610	1607	1606	1605	1604	1603	1602	1601	1600	V41070
1637	1636	1635	1634	1633	1632	1631	1630	1627	1626	1625	1624	1623	1622	1621	1620	V41071
1657	1656	1655	1654	1653	1652	1651	1650	1647	1646	1645	1644	1643	1642	1641	1640	V41072
1677	1676	1675	1674	1673	1672	1671	1670	1667	1666	1665	1664	1663	1662	1661	1660	V41073
1717	1716	1715	1714	1713	1712	1711	1710	1707	1706	1705	1704	1703	1702	1701	1700	V41074
1737	1736	1735	1734	1733	1732	1731	1730	1727	1726	1725	1724	1723	1722	1721	1720	V41075
1757	1756	1755	1754	1753	1752	1751	1750	1747	1746	1745	1744	1743	1742	1741	1740	V41076
1777	1776	1775	1774	1773	1772	1771	1770	1767	1766	1765	1764	1763	1762	1761	1760	V41077

Control Relay Bit Map

This table provides a listing of the individual control relays associated with each V-memory address bit.

MSB	DL06 Control Relays (C)														LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V40600
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V40601
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V40602
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V40603
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V40604
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V40605
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V40606
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V40607
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200	V40610
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220	V40611
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240	V40612
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260	V40613
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300	V40614
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320	V40615
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340	V40616
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360	V40617
417	416	415	414	413	412	411	410	407	406	405	404	403	402	401	400	V40620
437	436	435	434	433	432	431	430	427	426	425	424	423	422	421	420	V40621
457	456	455	454	453	452	451	450	447	446	445	444	443	442	441	440	V40622
477	476	475	474	473	472	471	470	467	466	465	464	463	462	461	460	V40623
517	516	515	514	513	512	511	510	507	506	505	504	503	502	501	500	V40624
537	536	535	534	533	532	531	530	527	526	525	524	523	522	521	520	V40625
557	556	555	554	553	552	551	550	547	546	545	544	543	542	541	540	V40626
577	576	575	574	573	572	571	570	567	566	565	564	563	562	561	560	V40627
617	616	615	614	613	612	611	610	607	606	605	604	603	602	601	600	V40630
637	636	635	634	633	632	631	630	627	626	625	624	623	622	621	620	V40631
657	656	655	654	653	652	651	650	647	646	645	644	643	642	641	640	V40632
677	676	675	674	673	672	671	670	667	666	665	664	663	662	661	660	V40633
717	716	715	714	713	712	711	710	707	706	705	704	703	702	701	700	V40634
737	736	735	734	733	732	731	730	727	726	725	724	723	722	721	720	V40635
757	756	755	754	753	752	751	750	747	746	745	744	743	742	741	740	V40636
777	776	775	774	773	772	771	770	767	766	765	764	763	762	761	760	V40637

This table is continued on the next page.

MSB	DL06 Control Relays (C) (cont'd)														LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1017	1016	1015	1014	1013	1012	1011	1010	1007	1006	1005	1004	1003	1002	1001	1000	V40640
1037	1036	1035	1034	1033	1032	1031	1030	1027	1026	1025	1024	1023	1022	1021	1020	V40641
1057	1056	1055	1054	1053	1052	1051	1050	1047	1046	1045	1044	1043	1042	1041	1040	V40642
1077	1076	1075	1074	1073	1072	1071	1070	1067	1066	1065	1064	1063	1062	1061	1060	V40643
1117	1116	1115	1114	1113	1112	1111	1110	1107	1106	1105	1104	1103	1102	1101	1100	V40644
1137	1136	1135	1134	1133	1132	1131	1130	1127	1126	1125	1124	1123	1122	1121	1120	V40645
1157	1156	1155	1154	1153	1152	1151	1150	1147	1146	1145	1144	1143	1142	1141	1140	V40646
1177	1176	1175	1174	1173	1172	1171	1170	1167	1166	1165	1164	1163	1162	1161	1160	V40647
1217	1216	1215	1214	1213	1212	1211	1210	1207	1206	1205	1204	1203	1202	1201	1200	V40650
1237	1236	1235	1234	1233	1232	1231	1230	1227	1226	1225	1224	1223	1222	1221	1220	V40651
1257	1256	1255	1254	1253	1252	1251	1250	1247	1246	1245	1244	1243	1242	1241	1240	V40652
1277	1276	1275	1274	1273	1272	1271	1270	1267	1266	1265	1264	1263	1262	1261	1260	V40653
1317	1316	1315	1314	1313	1312	1311	1310	1307	1306	1305	1304	1303	1302	1301	1300	V40654
1337	1336	1335	1334	1333	1332	1331	1330	1327	1326	1325	1324	1323	1322	1321	1320	V40655
1357	1356	1355	1354	1353	1352	1351	1350	1347	1346	1345	1344	1343	1342	1341	1340	V40656
1377	1376	1375	1374	1373	1372	1371	1370	1367	1366	1365	1364	1363	1362	1361	1360	V40657
1417	1416	1415	1414	1413	1412	1411	1410	1407	1406	1405	1404	1403	1402	1401	1400	V40660
1437	1436	1435	1434	1433	1432	1431	1430	1427	1426	1425	1424	1423	1422	1421	1420	V40661
1457	1456	1455	1454	1453	1452	1451	1450	1447	1446	1445	1444	1443	1442	1441	1440	V40662
1477	1476	1475	1474	1473	1472	1471	1470	1467	1466	1465	1464	1463	1462	1461	1460	V40663
1517	1516	1515	1514	1513	1512	1511	1510	1507	1506	1505	1504	1503	1502	1501	1500	V40664
1537	1536	1535	1534	1533	1532	1531	1530	1527	1526	1525	1524	1523	1522	1521	1520	V40665
1557	1556	1555	1554	1553	1552	1551	1550	1547	1546	1545	1544	1543	1542	1541	1540	V40666
1577	1576	1575	1574	1573	1572	1571	1570	1567	1566	1565	1564	1563	1562	1561	1560	V40667
1617	1616	1615	1614	1613	1612	1611	1610	1607	1606	1605	1604	1603	1602	1601	1600	V40670
1637	1636	1635	1634	1633	1632	1631	1630	1627	1626	1625	1624	1623	1622	1621	1620	V40671
1657	1656	1655	1654	1653	1652	1651	1650	1647	1646	1645	1644	1643	1642	1641	1640	V40672
1677	1676	1675	1674	1673	1672	1671	1670	1667	1666	1665	1664	1663	1662	1661	1660	V40673
1717	1716	1715	1714	1713	1712	1711	1710	1707	1706	1705	1704	1703	1702	1701	1700	V40674
1737	1736	1735	1734	1733	1732	1731	1730	1727	1726	1725	1724	1723	1722	1721	1720	V40675
1757	1756	1755	1754	1753	1752	1751	1750	1747	1746	1745	1744	1743	1742	1741	1740	V40676
1777	1776	1775	1774	1773	1772	1771	1770	1767	1766	1765	1764	1763	1762	1761	1760	V40677

Timer Status Bit Map

This table provides a listing of individual timer contacts associated with each V-memory address bit.

MSB	DL06 Timer (T) Contacts														LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V41100
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V41101
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V41102
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V41103
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V41104
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V41105
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V41106
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V41107
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200	V41110
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220	V41111
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240	V41112
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260	V41113
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300	V41114
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320	V41115
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340	V41116
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360	V41117

Counter Status Bit Map

MSB	DL06 Counter (CT) Contacts														LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V41140
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V41141
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V41142
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V41143
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V41144
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V41145
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V41146
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V41147

This table provides a listing of individual counter contacts associated with each V-memory address bit.

GX and GY I/O Bit Map

This table provides a listing of the individual global I/O points associated with each V-memory address bit.

MSB				DL06 GX and GY I/O Points								LSB				GX Address	GY Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V40000	V40200
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V40001	V40201
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V40002	V40202
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V40003	V40203
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V40004	V40204
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V40005	V40205
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V40006	V40206
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V40007	V40207
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200	V40010	V40210
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220	V40011	V40211
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240	V40012	V40212
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260	V40013	V40213
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300	V40004	V40214
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320	V40015	V40215
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340	V40016	V40216
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360	V40007	V40217
417	416	415	414	413	412	411	410	407	406	405	404	403	402	401	400	V40020	V40220
437	436	435	434	433	432	431	430	427	426	425	424	423	422	421	420	V40021	V40221
457	456	455	454	453	452	451	450	447	446	445	444	443	442	441	440	V40022	V40222
477	476	475	474	473	472	471	470	467	466	465	464	463	462	461	460	V40023	V40223
517	516	515	514	513	512	511	510	507	506	505	504	503	502	501	500	V40024	V40224
537	536	535	534	533	532	531	530	527	526	525	524	523	522	521	520	V40025	V40225
557	556	555	554	553	552	551	550	547	546	545	544	543	542	541	540	V40026	V40226
577	576	575	574	573	572	571	570	567	566	565	564	563	562	561	560	V40027	V40227
617	616	615	614	613	612	611	610	607	606	605	604	603	602	601	600	V40030	V40230
637	636	635	634	633	632	631	630	627	626	625	624	623	622	621	620	V40031	V40231
657	656	655	654	653	652	651	650	647	646	645	644	643	642	641	640	V40032	V40232
677	676	675	674	673	672	671	670	667	666	665	664	663	662	661	660	V40033	V40233
717	716	715	714	713	712	711	710	707	706	705	704	703	702	701	700	V40034	V40234
737	736	735	734	733	732	731	730	727	726	725	724	723	722	721	720	V40035	V40235
757	756	755	754	753	752	751	750	747	746	745	744	743	742	741	740	V40036	V40236
777	776	775	774	773	772	771	770	767	766	765	764	763	762	761	760	V40037	V40237

This table is continued on the next page.

NOTE: *This memory area can be used for additional Data Words.*

MSB	DL06 GX and GY I/O Points (cont'd)														LSB	GX	GY
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	Address
1017	1016	1015	1014	1013	1012	1011	1010	1007	1006	1005	1004	1003	1002	1001	1000	V40040	V40240
1037	1036	1035	1034	1033	1032	1031	1030	1027	1026	1025	1024	1023	1022	1021	1020	V40041	V40241
1057	1056	1055	1054	1053	1052	1051	1050	1047	1046	1045	1044	1043	1042	1041	1040	V40042	V40242
1077	1076	1075	1074	1073	1072	1071	1070	1067	1066	1065	1064	1063	1062	1061	1060	V40043	V40243
1117	1116	1115	1114	1113	1112	1111	1110	1107	1106	1105	1104	1103	1102	1101	1100	V40044	V40244
1137	1136	1135	1134	1133	1132	1131	1130	1127	1126	1125	1124	1123	1122	1121	1120	V40045	V40245
1157	1156	1155	1154	1153	1152	1151	1150	1147	1146	1145	1144	1143	1142	1141	1140	V40046	V40246
1177	1176	1175	1174	1173	1172	1171	1170	1167	1166	1165	1164	1163	1162	1161	1160	V40047	V40247
1217	1216	1215	1214	1213	1212	1211	1210	1207	1206	1205	1204	1203	1202	1201	1200	V40050	V40250
1237	1236	1235	1234	1233	1232	1231	1230	1227	1226	1225	1224	1223	1222	1221	1220	V40051	V40251
1257	1256	1255	1254	1253	1252	1251	1250	1247	1246	1245	1244	1243	1242	1241	1240	V40052	V40252
1277	1276	1275	1274	1273	1272	1271	1270	1267	1266	1265	1264	1263	1262	1261	1260	V40053	V40253
1317	1316	1315	1314	1313	1312	1311	1310	1307	1306	1305	1304	1303	1302	1301	1300	V40054	V40254
1337	1336	1335	1334	1333	1332	1331	1330	1327	1326	1325	1324	1323	1322	1321	1320	V40055	V40255
1357	1356	1355	1354	1353	1352	1351	1350	1347	1346	1345	1344	1343	1342	1341	1340	V40056	V40256
1377	1376	1375	1374	1373	1372	1371	1370	1367	1366	1365	1364	1363	1362	1361	1360	V40057	V40257
1417	1416	1415	1414	1413	1412	1411	1410	1407	1406	1405	1404	1403	1402	1401	1400	V40060	V40260
1437	1436	1435	1434	1433	1432	1431	1430	1427	1426	1425	1424	1423	1422	1421	1420	V40061	V40261
1457	1456	1455	1454	1453	1452	1451	1450	1447	1446	1445	1444	1443	1442	1441	1440	V40062	V40262
1477	1476	1475	1474	1473	1472	1471	1470	1467	1466	1465	1464	1463	1462	1461	1460	V40063	V40263
1517	1516	1515	1514	1513	1512	1511	1510	1507	1506	1505	1504	1503	1502	1501	1500	V40064	V40264
1537	1536	1535	1534	1533	1532	1531	1530	1527	1526	1525	1524	1523	1522	1521	1520	V40065	V40265
1557	1556	1555	1554	1553	1552	1551	1550	1547	1546	1545	1544	1543	1542	1541	1540	V40066	V40266
1577	1576	1575	1574	1573	1572	1571	1570	1567	1566	1565	1564	1563	1562	1561	1560	V40067	V40267
1617	1616	1615	1614	1613	1612	1611	1610	1607	1606	1605	1604	1603	1602	1601	1600	V40070	V40270
1637	1636	1635	1634	1633	1632	1631	1630	1627	1626	1625	1624	1623	1622	1621	1620	V40071	V40271
1657	1656	1655	1654	1653	1652	1651	1650	1647	1646	1645	1644	1643	1642	1641	1640	V40072	V40272
1677	1676	1675	1674	1673	1672	1671	1670	1667	1666	1665	1664	1663	1662	1661	1660	V40073	V40273
1717	1716	1715	1714	1713	1712	1711	1710	1707	1706	1705	1704	1703	1702	1701	1700	V40074	V40274
1737	1736	1735	1734	1733	1732	1731	1730	1727	1726	1725	1724	1723	1722	1721	1720	V40075	V40275
1757	1756	1755	1754	1753	1752	1751	1750	1747	1746	1745	1744	1743	1742	1741	1740	V40076	V40276
1777	1776	1775	1774	1773	1772	1771	1770	1767	1766	1765	1764	1763	1762	1761	1760	V40077	V40277

This table is continued on the next page.

NOTE: *This memory area can be used for additional Data Words.*

MSB	DL06 GX and GY I/O Points (cont'd)														LSB	GX	GY
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	Address
2017	2016	2015	2014	2013	2012	2011	2010	2007	2006	2005	2004	2003	2002	2001	2000	V40100	V40300
2037	2036	2035	2034	2033	2032	2031	2030	2027	2026	2025	2024	2023	2022	2021	2020	V40101	V40301
2057	2056	2055	2054	2053	2052	2051	2050	2047	2046	2045	2044	2043	2042	2041	2040	V40102	V40302
2077	2076	2075	2074	2073	2072	2071	2070	2067	2066	2065	2064	2063	2062	2061	2060	V40103	V40303
2117	2116	2115	2114	2113	2112	2111	2110	2107	2106	2105	2104	2103	2102	2101	2100	V40104	V40304
2137	2136	2135	2134	2133	2132	2131	2130	2127	2126	2125	2124	2123	2122	2121	2120	V40105	V40305
2157	2156	2155	2154	2153	2152	2151	2150	2147	2146	2145	2144	2143	2142	2141	2140	V40106	V40306
2177	2176	2175	2174	2173	2172	2171	2170	2167	2166	2165	2164	2163	2162	2161	2160	V40107	V40307
2217	2216	2215	2214	2213	2212	2211	2210	2207	2206	2205	2204	2203	2202	2201	2200	V40110	V40310
2237	2236	2235	2234	2233	2232	2231	2230	2227	2226	2225	2224	2223	2222	2221	2220	V40111	V40311
2257	2256	2255	2254	2253	2252	2251	2250	2247	2246	2245	2244	2243	2242	2241	2240	V40112	V40312
2277	2276	2275	2274	2273	2272	2271	2270	2267	2266	2265	2264	2263	2262	2261	2260	V40113	V40313
2317	2316	2315	2314	2313	2312	2311	2310	2307	2306	2305	2304	2303	2302	2301	2300	V40114	V40314
2337	2336	2335	2334	2333	2332	2331	2330	2327	2326	2325	2324	2323	2322	2321	2320	V40115	V40315
2357	2356	2355	2354	2353	2352	2351	2350	2347	2346	2345	2344	2343	2342	2341	2340	V40116	V40316
2377	2376	2375	2374	2373	2372	2371	2370	2367	2366	2365	2364	2363	2362	2361	2360	V40117	V40317
2417	2416	2415	2414	2413	2412	2411	2410	2407	2406	2405	2404	2403	2402	2401	2400	V40120	V40320
2437	2436	2435	2434	2433	2432	2431	2430	2427	2426	2425	2424	2423	2422	2421	2420	V40121	V40321
2457	2456	2455	2454	2453	2452	2451	2450	2447	2446	2445	2444	2443	2442	2441	2440	V40122	V40322
2477	2476	2475	2474	2473	2472	2471	2470	2467	2466	2465	2464	2463	2462	2461	2460	V40123	V40323
2517	2516	2515	2514	2513	2512	2511	2510	2507	2506	2505	2504	2503	2502	2501	2500	V40124	V40324
2537	2536	2535	2534	2533	2532	2531	2530	2527	2526	2525	2524	2523	2522	2521	2520	V40125	V40325
2557	2556	2555	2554	2553	2552	2551	2550	2547	2546	2545	2544	2543	2542	2541	2540	V40126	V40326
2577	2576	2575	2574	2573	2572	2571	2570	2567	2566	2565	2564	2563	2562	2561	2560	V40127	V40327
2617	2616	2615	2614	2613	2612	2611	2610	2607	2606	2605	2604	2603	2602	2601	2600	V40130	V40330
2637	2636	2635	2634	2633	2632	2631	2630	2627	2626	2625	2624	2623	2622	2621	2620	V40131	V40331
2657	2656	2655	2654	2653	2652	2651	2650	2647	2646	2645	2644	2643	2642	2641	2640	V40132	V40332
2677	2676	2675	2674	2673	2672	2671	2670	2667	2666	2665	2664	2663	2662	2661	2660	V40133	V40333
2717	2716	2715	2714	2713	2712	2711	2710	2707	2706	2705	2704	2703	2702	2701	2700	V40134	V40334
2737	2736	2735	2734	2733	2732	2731	2730	2727	2726	2725	2724	2723	2722	2721	2720	V40135	V40335
2757	2756	2755	2754	2753	2752	2751	2750	2747	2736	2735	2734	2733	2732	2731	2730	V40136	V40336
2777	2776	2775	2774	2773	2772	2771	2770	2767	2766	2765	2764	2763	2762	2761	2760	V40137	V40337

This table is continued on the next page.

NOTE: *This memory area can be used for additional Data Words.*

MSB	DL06 GX and GY I/O Points (cont'd)														LSB	GX	GY
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	Address
3017	3016	3015	3014	3013	3012	3011	3010	3007	3006	3005	3004	3003	3002	3001	3000	V40140	V40340
3037	3036	3035	3034	3033	3032	3031	3030	3027	3026	3025	3024	3023	3022	3021	3020	V40141	V40341
3057	3056	3055	3054	3053	3052	3051	3050	3047	3046	3045	3044	3043	3042	3041	3040	V40142	V40342
3077	3076	3075	3074	3073	3072	3071	3070	3067	3066	3065	3064	3063	3062	3061	3060	V40143	V40343
3117	3116	3115	3114	3113	3112	3111	3110	3107	3106	3105	3104	3103	3102	3101	3100	V40144	V40344
3137	3136	3135	3134	3133	3132	3131	3130	3127	3126	3125	3124	3123	3122	3121	3120	V40145	V40345
3157	3156	3155	3154	3153	3152	3151	3150	3147	3146	3145	3144	3143	3142	3141	3140	V40146	V40346
3177	3176	3175	3174	3173	3172	3171	3170	3167	3166	3165	3164	3163	3162	3161	3160	V40147	V40347
3217	3216	3215	3214	3213	3212	3211	3210	3207	3206	3205	3204	3203	3202	3201	3200	V40150	V40350
3237	3236	3235	3234	3233	3232	3231	3230	3227	3226	3225	3224	3223	3222	3221	3220	V40151	V40351
3257	3256	3255	3254	3253	3252	3251	3250	3247	3246	3245	3244	3243	3242	3241	3240	V40152	V40352
3277	3276	3275	3274	3273	3272	3271	3270	3267	3266	3265	3264	3263	3262	3261	3260	V40153	V40353
3317	3316	3315	3314	3313	3312	3311	3310	3307	3306	3305	3304	3303	3302	3301	3300	V40154	V40354
3337	3336	3335	3334	3333	3332	3331	3330	3327	3326	3325	3324	3323	3322	3321	3320	V40155	V40355
3357	3356	3355	3354	3353	3352	3351	3350	3347	3346	3345	3344	3343	3342	3341	3340	V40156	V40356
3377	3376	3375	3374	3373	3372	3371	3370	3367	3366	3365	3364	3363	3362	3361	3360	V40157	V40357
3417	3416	3415	3414	3413	3412	3411	3410	3407	3406	3405	3404	3403	3402	3401	3400	V40160	V40360
3437	3436	3435	3434	3433	3432	3431	3430	3427	3426	3425	3424	3423	3422	3421	3420	V40161	V40361
3457	3456	3455	3454	3453	3452	3451	3450	3447	3446	3445	3444	3443	3442	3441	3440	V40162	V40362
3477	3476	3475	3474	3473	3472	3471	3470	3467	3466	3465	3464	3463	3462	3461	3460	V40163	V40363
3517	3516	3515	3514	3513	3512	3511	3510	3507	3506	3505	3504	3503	3502	3501	3500	V40164	V40364
3537	3536	3535	3534	3533	3532	3531	3530	3527	3526	3525	3524	3523	3522	3521	3520	V40165	V40365
3557	3556	3555	3554	3553	3552	3551	3550	3547	3546	3545	3544	3543	3542	3541	3540	V40166	V40366
3577	3576	3575	3574	3573	3572	3571	3570	3567	3566	3565	3564	3563	3562	3561	3560	V40167	V40367
3617	3616	3615	3614	3613	3612	3611	3610	3607	3606	3605	3604	3603	3602	3601	3600	V40170	V40370
3637	3636	3635	3634	3633	3632	3631	3630	3627	3626	3625	3624	3623	3622	3621	3620	V40171	V40371
3657	3656	3655	3654	3653	3652	3651	3650	3647	3646	3645	3644	3643	3642	3641	3640	V40172	V40372
3677	3676	3675	3674	3673	3672	3671	3670	3667	3666	3665	3664	3663	3662	3661	3660	V40173	V40373
3717	3716	3715	3714	3713	3712	3711	3710	3707	3706	3705	3704	3703	3702	3701	3700	V40174	V40374
3737	3736	3735	3734	3733	3732	3731	3730	3727	3726	3725	3724	3723	3722	3721	3720	V40175	V40375
3757	3756	3755	3754	3753	3752	3751	3750	3747	3746	3745	3744	3743	3742	3741	3740	V40176	V40376
3777	3776	3775	3774	3773	3772	3771	3770	3767	3766	3765	3764	3763	3762	3761	3760	V40177	V40377

NOTE: *This memory area can be used for additional Data Words.*

CHAPTER 4

SYSTEM DESIGN AND CONFIGURATION

In This Chapter

DL06 System Design Strategies

I/O System Configurations

The DL06 PLCs offer a number of different I/O configurations. Choose the configuration that is right for your application, and keep in mind that the DL06 PLCs offer the ability to add I/O with the use of option cards. Although remote I/O isn't available, there are many option cards available. For instance:

- Various A/C and D/C I/O modules
- Combination I/O modules
- Analog I/O modules
- Combination Analog I/O modules

A DL06 system can be developed using several different arrangements using the option modules. See our DL05/06 Options Modules User Manual (D0-OPTIONS-M) on the website, www.automationdirect.com for detailed selection information.

Networking Configurations

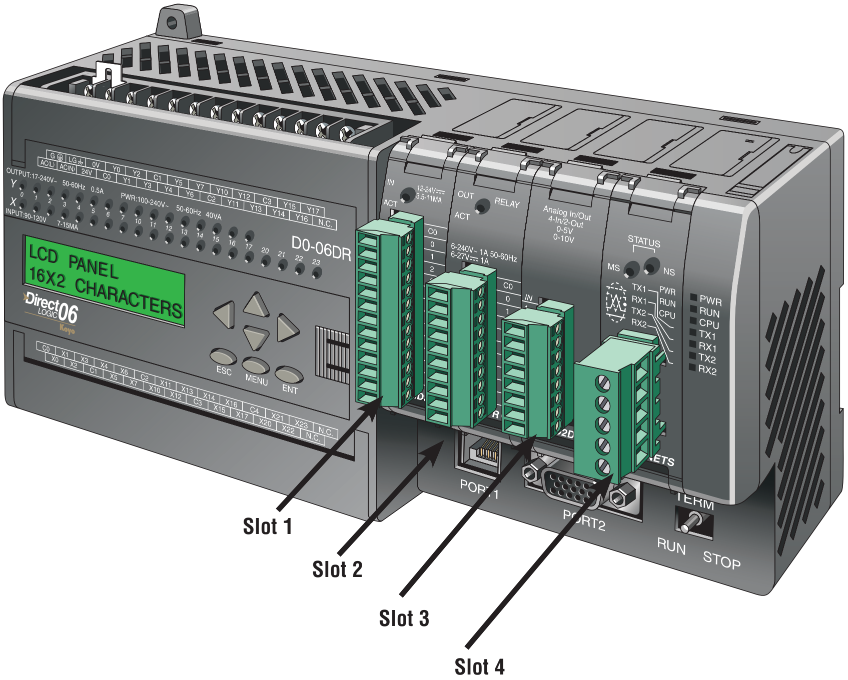
The DL06 PLCs offers the following ways to add networking:

- **Ethernet Communications Module** – connects a DL06 to high-speed peer-to-peer networks. Any PLC can initiate communications with any other PLC or operator interfaces, such as C-more, when using the ECOM modules.
- **Data Communications Modules** – connects a DL06 to devices using either DeviceNet or Profibus to link to master controllers, as well as a D0-DCM.
- **Communications Port 1** – The DL06 has a 6-pin RJ12 connector on Port 1 that supports (as slave) K-sequence, MODBUS RTU or ***Direct***NET protocols.
- **Communications Port 2** – The DL06 has a 15-pin connector on Port 2 that supports either master/slave MODBUS RTU or ***Direct***NET protocols, or K-sequence protocol as slave. (MRX and MWX instructions allow you to enter native MODBUS addressing in your ladder program with no need to perform octal to decimal conversions). Port 2 can also be used for ASCII IN/OUT communictions.

Module Placement

Slot Numbering

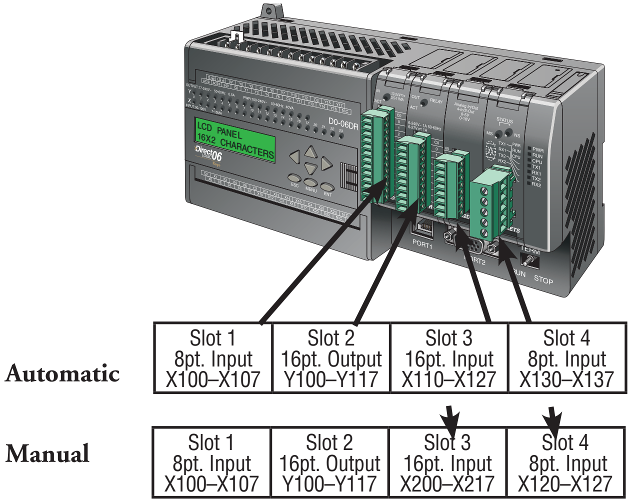
The DL06 has four slots, which are numbered as follows:



Automatic I/O Configuration

The DL06 CPUs automatically detect any installed I/O modules (including specialty modules) at powerup, and establish the correct I/O configuration and addresses. This applies to modules located in the local base. For most applications, you will never have to change the configuration.

I/O addresses use octal numbering, starting at X100 and Y100 in the slot next to the CPU. The addresses are assigned in groups of 8, or 16 depending on the number of points for the I/O module. The discrete input and output modules can be mixed in any order. The following diagram shows the I/O numbering convention for an example system. Both the Handheld Programmer and *Direct*SOFT 5 provide AUX functions that allow you to automatically configure the I/O. For example, with the Handheld Programmer AUX 46 executes an automatic configuration, which allows the CPU to examine the installed modules and determine the I/O configuration and addressing.With *Direct*SOFT 5, the PLC Configure I/O menu option would be used.



Automatic	Slot 1 8pt. Input X100–X107	Slot 2 16pt. Output Y100–Y117	Slot 3 16pt. Input X110–X127	Slot 4 8pt. Input X130–X137
Manual	Slot 1 8pt. Input X100–X107	Slot 2 16pt. Output Y100–Y117	Slot 3 16pt. Input X200–X217	Slot 4 8pt. Input X120–X127

Manual I/O Configuration

It may never become necessary, but DL06 CPUs allow manual I/O address assignments for any I/O slot(s) . You can manually modify an auto configuration to match arbitrary I/O numbering. For example, two adjacent input modules can have starting addresses at X100 and X200.Use *Direct*SOFT 5 PLC Configure I/O menu option to assign manual I/O address. In automatic configuration, the addresses are assigned on 8-point boundaries. Manual configuration, however, assumes that all modules are at least 16 points, so you can only assign addresses that are a multiple of 20 (octal). You can still use 8 point modules, but 16 addresses will be assigned and the upper eight addresses will be unused.

⚠ **WARNING: If you manually configure an I/O slot, the I/O addressing for the other modules may change. This is because the DL06 CPUs do not allow you to assign duplicate I/O addresses. You must always correct any I/O configuration errors before you place the CPU in RUN mode. Uncorrected errors can cause unpredictable machine operation that can result in a risk of personal injury or damage to equipment.**

Power Budgeting

The DL06 has four option card slots. To determine whether the combination of cards you select will have sufficient power, you will need to perform a power budget calculation.

Power supplied

Power is supplied from two sources, the internal base unit power supply and, if required, an external supply (customer furnished). The D0-06xx (AC powered) PLCs supply a limited amount of 24VDC power. The 24VDC output can be used to power external devices.

For power budgeting, start by considering the power supplied by the base unit. All DL06 PLCs supply the same amount of 5VDC power. Only the AC units offer 24VDC auxiliary power. Be aware of the trade-off between 5VDC power and 24VDC power. The amount of 5VDC power available depends on the amount of 24VDC power being used, and the amount of 24VDC power available depends on the amount of 5VDC power consumed. Determine the amount of internally supplied power from the table on the following page.

Power required by base unit

Because of the different I/O configurations available in the DL06 family, the power consumed by the base unit itself varies from model to model. Subtract the amount of power required by the base unit from the amount of power supplied by the base unit. Be sure to subtract 5VDC and 24VDC amounts.

Power required by option cards

Next, subtract the amount of power required by the option cards you are planning to use. Again, remember to subtract both 5VDC and 24VDC. If your power budget analysis shows surplus power available, you should have a workable configuration.

DL06 Power Supplied by Base Units		
Part Number	**5 VDC (mA)**	**24 VDC (mA)**
D0-06xx	<1500mA	300mA
	<2000mA	200mA
D0-06xx-D	1500mA	none

If the 5VDC loading is less than 2000mA, but more than 1500mA, then available 24VDC supply current is 200mA. If the 5VDC loading is less than 1500mA, then the available 24VDC current is 300mA.

DL06 Base Unit Power Required		
Part Number	**5 VDC (mA)**	**24 VDC (mA)**
D0-06AA	800mA	none
D0-06AR	900mA	none
D0-06DA	800mA	none
D0-06DD1	600mA	280mA, note 1
D0-06DD2	600mA	none
D0-06DR	950mA	none
D0-06DD1-D	600mA	280mA, note 1
D0-06DD2-D	600mA	none
D0-06DR-D	950mA	none

Power Budgeting Example			
Power Source		**5VDC power (mA)**	**24VDC power (mA)**
D0-06DD1 (select row A or row B)	A	1500mA	300mA
	B	2000mA	200mA
Current Required		**5VDC power (mA)**	**24VDC power (mA)**
D0-06DD1		600mA	280mA, note 1
D0-16ND3		35mA	0
D0-10TD1		150mA	0
D0-08TR		280mA	0
F0-4AD2DA-2		100mA	0
D0-06LCD		50mA	0
Total Used		1215mA	280mA
Remaining	A	285mA	20mA
	B	785mA	note 2

NOTE: See the DL05/DL06 OPTIONS manual for the module data for your project.

DL06 Power Consumed by Option Cards		
Part Number	**5 VDC (mA)**	**24 VDC (mA)**
D0-07CDR	130mA	none
D0-08CDD1	100mA	none
D0-08TR	280mA	none
D0-10ND3	35mA	none
D0-10ND3F	35mA	none
D0-10TD1	150mA	none
D0-10TD2	150mA	none
D0-16ND3	35mA	none
D0-16TD1	200mA	none
D0-16TD2	200mA	none
D0-DCM	250mA	none
D0-DEVNETS	45mA	none
F0-04TRS	250mA	none
F0-08NA-1	5mA	none
F0-04AD-1	50mA	none
F0-04AD-2	75mA	none
F0-04DAH-1	25mA	150mA
F0-04DAH-2	25mA	30mA
F0-08ADH-1	25mA	25mA
F0-08ADH-2	25mA	25mA
F0-08DAH-1	25mA	220mA
F0-08DAH-2	25mA	30mA
F0-2AD2DA-2	50mA	30mA
F0-4AD2DA-1	100mA	40mA
F0-4AD2DA-2	100mA	none
F0-04RTD	70mA	none
F0-04THM	30mA	none
F0-CP128	150mA	none
H0-CTRIO(2)	250mA	none
H0-ECOM	250mA	none
H0-ECOM100	300mA	none
H0-PSCM	530mA	none

DL06 Power Consumed by Other Devices		
Part Number	**5 VDC (mA)**	**24 VDC (mA)**
D0-06LCD	50mA	none
D2-HPP	200mA	none
DV-1000	150mA	none
EA1-S3ML	210mA	none
EA1-S3MLW	210mA	none

NOTE 1: Auxiliary 24VDC used to power V+ terminal of D0-06DD1/-D sinking outputs.
NOTE 2: If the PLC's auxiliary 24VDC power source is used to power the sinking outputs, use power choice A, above.

Configuring the DL06's Comm Ports

This section describes how to configure the CPU's built-in networking ports for either MODBUS or *Direct*NET. This will allow you to connect the DL06 PLC system directly to MODBUS networks using the RTU protocol, or to other devices on a *Direct*NET network. MODBUS masters on the network must be capable of issuing the MODBUS commands to read or write the appropriate data. For details on the MODBUS protocol, please refer to the Gould MODBUS Protocol reference Guide (P1–MBUS–300 Rev. B). In the event a more recent version is available, check with your MODBUS supplier before ordering the documentation. For more details on *Direct*NET, order our *Direct*NET manual, part number DA–DNET–M.

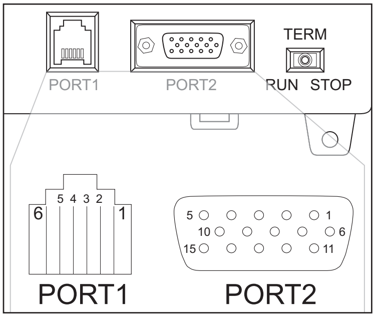
NOTE: For information about the MODBUS protocol see the Group Schneider Web site at: www.schneiderautomation.com. At the main menu, select Support/Services, Modbus, Modbus Technical Manuals, PI-MBUS-300 Modbus Protocol Reference Guide or search for PIMBUS300. For more information about the ***Direct****NET protocol, order our* ***Direct****NET user manual, part number DA–DNET–M, or download it free from our Web site: www.automationdirect.com. Select Documentation/Misc./DA-DNET-M.*

DL06 Port Specifications

Communications Port 1	
Port 1	Connects to HPP, ***Direct***SOFT 5, operator interfaces, etc.
	6-pin, RS232C
	Communication speed (baud): 9600 (fixed)
	Parity: odd (fixed)
	Station Address: 1 (fixed)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocol (auto-select): K-sequence (slave only), ***Direct***NET (slave only), MODBUS (slave only)

Communications Port 2	
Port 2	Connects to HPP, ***Direct***SOFT 5, operator interfaces, etc.
	15-pin, multifunction port, RS232C, RS422, RS485
	Communication speed (baud): 300, 600, 1200, 2400, 4800, 9600, 19200, 38400
	Parity: odd (default), even, none
	Station Address: 1 (default)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocol (auto-select): K-sequence (slave only), ***Direct***NET (master/slave), MODBUS (master/slave), non-sequence/print/ASCII in/out

DL06 Port Pinouts



Port 1 Pin Descriptions		
1	0V	Power (-) connection (GND)
2	5V	Power (+) connection
3	RXD	Receive data (RS-232C)
4	TXD	Transmit data (RS-232C)
5	5V	Power (+) connection
6	0V	Power (-) connection (GND)

Port 2 Pin Descriptions		
1	5V	Power (+) connection
2	TXD	Transmit data (RS-232C)
3	RXD	Receive data (RS-232C)
4	RTS	Ready to send (RS-232C)
5	CTS	Clear to send (RS232C)
6	RXD-	Receive data (-) (RS-422/485)
7	0V	Power (-) connection (GND)
8	0V	Power (-) connection (GND)
9	TXD+	Transmit data (+) (RS-422/485)
10	TXD-	Transmit data (-) (RS-422/485)
11	RTS+	Ready to send (+) (RS-422/485)
12	RTS-	Ready to send (-) (RS-422/485)
13	RXD+	Receive data (+) (RS-422/485)
14	CTS+	Clear to send (+) (RS-422/485)
15	CTS-	Clear to send (-) (RS-422/485)

Choosing a Network Specification

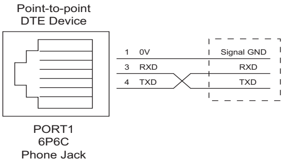
The DL06 PLC's multi-function port gives you the option of using RS-232C, RS-422, or RS-485 specifications. First, determine whether the network will be a 2-wire RS–232C type, a 4-wire RS–422 type, or a 2-wire/4-wire RS-485 type.

The RS–232C specification is simple to implement for networks of shorter distances (15 meters max) and where communication is only required between two devices. The RS–422 and RS-485 signals are for networks that cover longer distances (1000 meters max.) and for multi-drop networks (from 2 to 247 devices).

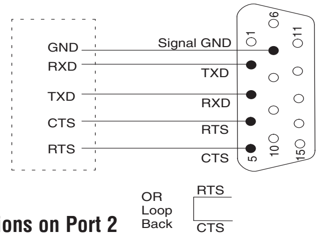
NOTE: Termination resistors are required at both ends of RS–422 and RS-485 networks. It is necessary to select resistors that match the impedance rating of the cable (between 100 and 500 ohms).

RS-232 Network

Normally, the RS-232 signals are used for shorter distances (15 meters maximum), for communications between two devices.



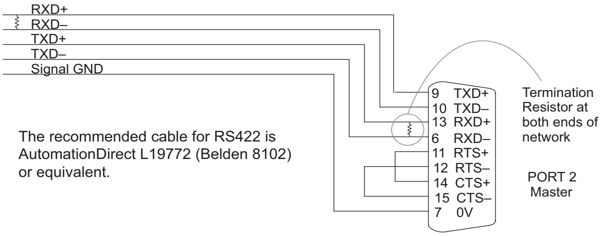
Connections on Port 1



Connections on Port 2

RS-422 Network

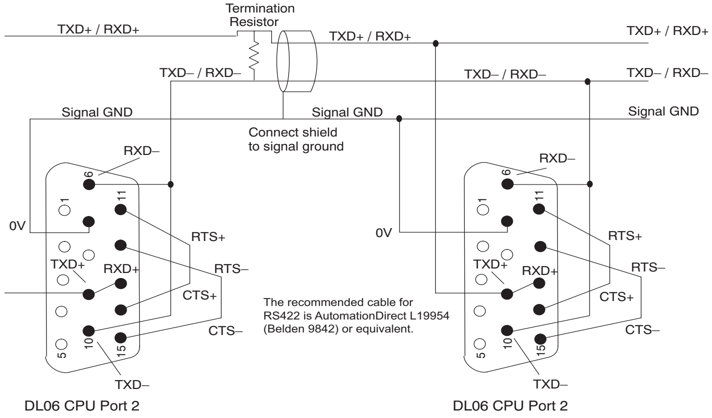
RS-422 signals are for long distances (1000 meters maximum). Use terminator resistors at both ends of RS-422 network wiring, matching the impedence rating of the cable (between 100 and 500 ohms).



The recommended cable for RS422 is AutomationDirect L19772 (Belden 8102) or equivalent.

RS-485 Network

RS-485 signals are for longer distances (1000 meters max) and for multi-drop networks. Use termination resistors at both ends of RS-485 network wiring, matching the impedance rating of the cable (between 100 and 500 ohms).



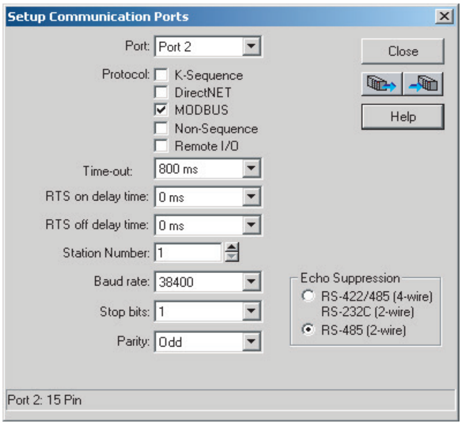
The recommended cable for RS422 is AutomationDirect L19954 (Belden 9842) or equivalent.

Connecting to MODBUS and *Direct*NET Networks

MODBUS Port Configuration

In *Direct*SOFT, choose the PLC menu, then Setup, then “Secondary Comm Port”.

- **Port**: From the port number list box at the top, choose “Port 2”.
- **Protocol**: Check the box to the left of “MODBUS” (use AUX 56 on the HPP, and select “MBUS”), and then you’ll see the box below.



Setup Communication Ports

Port: Port 2

Protocol: ☐ K-Sequence ☐ DirectNET ☑ MODBUS ☐ Non-Sequence ☐ Remote I/O

Time-out: 800 ms

RTS on delay time: 0 ms

RTS off delay time: 0 ms

Station Number: 1

Baud rate: 38400

Stop bits: 1

Parity: Odd

Echo Suppression: ○ RS-422/485 (4-wire) RS-232C (2-wire) ● RS-485 (2-wire)

Port 2: 15 Pin

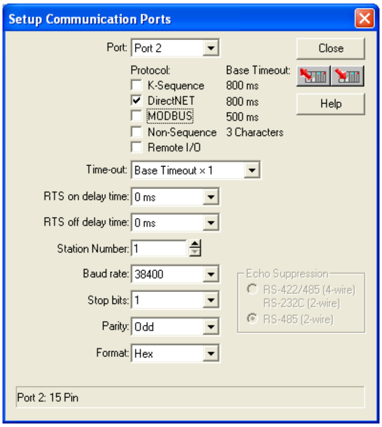
- **Timeout**: amount of time the port will wait after it sends a message to get a response before logging an error.
- **RTS ON / OFF Delay Time:** The RTS ON Delay Time specifies the time the DL06 waits to send the data after it has raised the RTS signal line. The RTS OFF Delay Time specifies the time the DL06 waits to release the RTS signal line after the data has been sent. *When using the DL06 on a multi-drop network, the RTS ON Delay time must be set to **5ms** or more and the RTS OFF Delay time must be set to **2ms** or more. If you encounter problems, the time can be increased.*
- **Station Number**: For making the CPU port a MODBUS master, choose “1”. The possible range for MODBUS slave numbers is from 1 to 247, but the DL06 network instructions used in Master mode will access only slaves 1 to 99. Each slave must have a unique number. At powerup, the port is automatically a slave, unless and until the DL06 executes ladder logic network instructions which use the port as a master. Thereafter, the port reverts back to slave mode until ladder logic uses the port again.
- **Baud Rate**: The available baud rates include 300, 600, 1200, 2400, 4800, 9600, 19200, and 38400 baud. Choose a higher baud rate initially, reverting to lower baud rates if you experience data errors or noise problems on the network. Important: You must configure the baud rates of all devices on the network to the same value. Refer to the appropriate product manual for details.
- **Stop Bits**: Choose 1 or 2 stop bits for use in the protocol.
- **Parity**: Choose none, even, or odd parity for error checking.
- **Echo Suppression:** Select the appropriate wiring configuration used on Port 2.

▶ Then click the button indicated to send the Port configuration to the CPU, and click Close.

*Direct*NET Port Configuration

In *Direct*SOFT, choose the PLC menu, then Setup, then "Secondary Comm Port".

- **Port**: From the port number list box, choose "Port 2 ".
- **Protocol**: Check the box to the left of "*Direct*NET" (use AUX 56 on the HPP, then select "DNET"), and then you'll see the dialog below.



Setup Communication Ports

Port: Port 2

Close

Protocol: Base Timeout:

☐ K-Sequence 800 ms

☑ DirectNET 800 ms

☐ MODBUS 500 ms

☐ Non-Sequence 3 Characters

☐ Remote I/O

Help

Time-out: Base Timeout x 1

RTS on delay time: 0 ms

RTS off delay time: 0 ms

Station Number: 1

Baud rate: 38400

Stop bits: 1

Parity: Odd

Format: Hex

Echo Suppression

○ RS-422/485 (4-wire)
RS-232C (2-wire)

● RS-485 (2-wire)

Port 2: 15 Pin

- **Timeout**: Amount of time the port will wait after it sends a message to get a response before logging an error.
- **RTS ON / OFF Delay Time:** The RTS ON Delay Time specifies the time the DL06 waits to send the data after it has raised the RTS signal line. The RTS OFF Delay Time specifies the time the DL06 waits to release the RTS signal line after the data has been sent. *When using the DL06 on a multi-drop network, the RTS ON Delay time must be set to 5ms or more and the RTS OFF Delay time must be set to 2ms or more. If you encounter problems, the time can be increased.*
- **Station Number**: For making the CPU port a *Direct*NET master, choose "1". The allowable range for *Direct*NET slaves is from 1 to 90 (each slave must have a unique number). At powerup, the port is automatically a slave, unless and until the DL06 executes ladder logic instructions which attempt to use the port as a master. Thereafter, the port reverts back to slave mode until ladder logic uses the port again.
- **Baud Rate**: The available baud rates include 300, 600, 1200, 2400, 4800, 9600, 19200, and 38400 baud. Choose a higher baud rate initially, reverting to lower baud rates if you experience data errors or noise problems on the network. Important: You must configure the baud rates of all devices on the network to the same value.
- **Stop Bits**: Choose 1 or 2 stop bits for use in the protocol.
- **Parity**: Choose none, even, or odd parity for error checking.
- **Format**: Choose between hex or ASCII formats.

Then click the button indicated to send the Port configuration to the CPU, and click Close.

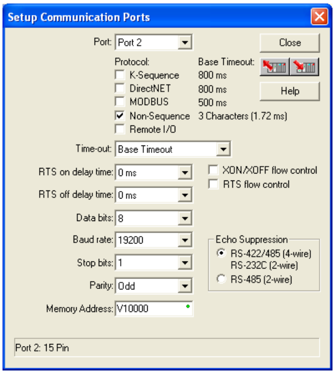
Non–Sequence Protocol (ASCII In/Out and PRINT)

Non-Sequence Port Configuration

Configuring port 2 on the DL06 for Non–Sequence allows the CPU to use port 2 to either read or write raw ASCII strings using the ASCII instructions. See the ASCII In/Out instructions and the PRINT instruction in chapter 5.

In *Direct*SOFT, choose the PLC menu, then Setup, then "Secondary Comm Port".

- **Port**: From the port number list box at the top, choose "Port 2".
- **Protocol**: Check the box to the left of "Non–Sequence".



Setup Communication Ports

Port: Port 2

Protocol: Base Timeout:
□ K-Sequence 800 ms
□ DirectNET 800 ms
□ MODBUS 500 ms
☑ Non-Sequence 3 Characters (1.72 ms)
□ Remote I/O

Time-out: Base Timeout

RTS on delay time: 0 ms

RTS off delay time: 0 ms

Data bits: 8

Baud rate: 19200

Stop bits: 1

Parity: Odd

Memory Address: V10000

□ XON/XOFF flow control
□ RTS flow control

Echo Suppression
● RS-422/485 (4-wire) RS-232C (2-wire)
○ RS-485 (2-wire)

Port 2: 15 Pin

- **Timeout**: Amount of time the port will wait after it sends a message to get a response before logging an error.
- **RTS On Delay Time**: The amount of time between raising the RTS line and sending the data.
- **RTS Off Delay Time**: The amount of time between resetting the RTS line after sending the data.
- **Data Bits**: Select either 7–bits or 8–bits to match the number of data bits specified for the connected devices.
- **Baud Rate**: The available baud rates include 300, 600, 900, 2400, 4800, 9600, 19200, and 38400 baud. Choose a higher baud rate initially, reverting to lower baud rates if you experience data errors or noise problems on the network. Important: You must configure the baud rates of all devices on the network to the same value. Refer to the appropriate product manual for details.
- **Stop Bits**: Choose 1 or 2 stop bits to match the number of stop bits specified for the connected devices.
- **Parity**: Choose none, even, or odd parity for error checking. Be sure to match the parity specified for the connected devices.
- **Echo Suppression**: Select the appropriate radio button based on the wiring configuration used on port 2.
- **Xon/Xoff Flow Control**: Choose this selection if you have Port 2 wired for Hardware Flow Control (Xon/Xoff) with RTS and CTS signal connected between all devices.
- **RTS Flow Control**: Choose this selection if you have Port 2 RTS signal wired between all devices.

Click the button indicated to send the port configuration to the CPU, and click Close.

- **Memory Address:** Please choose a memory address with 64 words of contiguous free memory for use by Non-Sequence Protocol.

Network Slave Operation

This section describes how other devices on a network can communicate with a CPU port that you have configured as a *Direct*NET slave or MODBUS slave (DL06). A MODBUS host must use the MODBUS RTU protocol to communicate with the DL06 as a slave. The host software must send a MODBUS function code and MODBUS address to specify a PLC memory location the DL06 comprehends. The *Direct*NET host uses normal I/O addresses to access applicable DL06 CPU and system. No CPU ladder logic is required to support either MODBUS slave or *Direct*NET slave operation.

*NOTE: For more intformation on **Direct**NET proprietary protocol, see the **Direct**NET reference manual, DA-DNET-M, available on our website.*

MODBUS Function Codes Supported

MODBUS Function Code	Function	DL06 Data Types Available
01	Read a group of coils	Y, CR, T, CT
02	Read a group of inputs	X, SP
05	Set / Reset a single coil	Y, CR, T, CT
15	Set / Reset a group of coils Y,	CR, T, CT
03, 04	Read a value from one or more registers	V
06	Write a value into a single register	V
16	Write a value into a group of registers	V

The MODBUS function code determines whether the access is a read or a write, and whether to access a single data point or a group of them. The DL06 supports the MODBUS function codes described below.

Determining the MODBUS Address

There are typically two ways that most host software conventions allow you to specify a PLC memory location. These are:

- By specifying the MODBUS data type and address
- By specifying a MODBUS address only

Word Data Types			
Registers	**PLC Range (Octal)**	**Input/Holding (484 Mode)***	**Input/Holding (584/984 Mode)***
V-Memory (Timers)	V0 - V377	3001 / 4001	30001 / 40001
V-Memory (Counters)	V1000 - V1177	3513 / 4513	30513 / 40513
V-Memory (Data Words)	V400 - V677	3257 / 4257	30257 / 40257
	V1200 - V7377	3641 / 4641	30641 / 40641
	V10000 - V17777	-	34097 / 44097
*** Modbus: Function 04**			

If Your Host Software Requires the Data Type and Address

Many host software packages allow you to specify the MODBUS data type and the MODBUS address that corresponds to the PLC memory location. This is the easiest method, but not all packages allow you to do it this way.

The actual equation used to calculate the address depends on the type of PLC data you are using. The PLC memory types are split into two categories for this purpose.

- Discrete – X, SP, Y, CR, S, T, C (contacts)
- Word – V, Timer current value, Counter current value

In either case, you basically convert the PLC octal address to decimal and add the appropriate MODBUS address (if required). The table below shows the exact equation used for each group of data.

DL06 Memory Type	QTY (Decimal)	PLC Range (Octal)	MODBUS Address Range (Decimal)	MODBUS Data Type
For Discrete Data Types …. Convert PLC Addr. to Dec. + Start of Range + Data Type				
Inputs (X)	512	X0 – X777	2048 – 2559	Input
Special Relays(SP)	512	SP0 – SP777	3072 – 3583	Input
Outputs (Y)	512	Y0 – Y777	2048 – 2559	Coil
Control Relays (CR)	1024	C0 – C1777	3072 – 4095	Coil
Timer Contacts (T)	256	T0 – T377	6144 – 6399	Coil
Counter Contacts (CT)	128	CT0 – CT177	6400 – 6527	Coil
Stage Status Bits(S)	1024	S0 – S1777	5120 – 6143	Coil
For Word Data Types …. Convert PLC Addr. to Dec. + Data Type				
Timer Current Values (V)	256	V0 – V377	0 – 255	Input Register
Counter Current Values (V)	128	V1000 – V1177	512 – 639	Input Register
V-Memory, user data (V)	3200	V1200 – V7377	640 – 3839	Holding Register
	4096	V10000 - V17777	4096 - 8191	Holding Register
V-Memory, non-volatile (V)	128	V7400 – V7577	3840 – 3967	Holding Register

The following examples show how to generate the MODBUS address and data type for hosts which require this format.

Example 1: V2100

Find the MODBUS address for User V location V2100.

1. Find V-memory in the table.
2. Convert V2100 into decimal (1088).
3. Use the MODBUS data type from the table.

Holding Reg 1088

V-memory, user data (V)	3200	V1200 – V7377	640 – 3839	Holding Register

Example 2: Y20

Find the MODBUS address for output Y20.

1. Find Y outputs in the table.
2. Convert Y20 into decimal (16).
3. Add the starting address for the range (2048).
4. Use the MODBUS data type from the table.

Coil 2064

Outputs (V)	256	Y0 – Y377	2048 - 2303	Coil

Example 3: T10 Current Value

Find the MODBUS address to obtain the current value from Timer T10.

1. Find Timer Current Values in the table.
2. Convert T10 into decimal (8).
3. Use the MODBUS data type from the table.

Input Reg. 8

Timer Current Values (V)	128	V0 – V177	0 - 127	Input Register

Example 4: C54

Find the MODBUS address for Control Relay C54.

1. Find Control Relays in the table.
2. Convert C54 into decimal (44).
3. Add the starting address for the range (3072).
4. Use the MODBUS data type from the table.

Coil 3116

Control Relays (CR)	512	C0 – C77	3072 – 3583	Coil

If Your MODBUS Host Software Requires an Address ONLY

Some host software does not allow you to specify the MODBUS data type and address. Instead, you specify an address only. This method requires another step to determine the address, but it's still fairly simple. Basically, MODBUS also separates the data types by address ranges as well. So this means an address alone can actually describe the type of data and location. This is often referred to as "adding the offset". One important thing to remember here is that two different addressing modes may be available in your host software package. These are:

- 484 Mode
- 584/984 Mode

We recommend that you use the 584/984 addressing mode if your host software allows you to choose. This is because the 584/984 mode allows access to a higher number of memory locations within each data type. If your software only supports 484 mode, then there may be some PLC memory locations that will be unavailable. The actual equation used to calculate the address depends on the type of PLC data you are using. The PLC memory types are split into two categories for this purpose.

- Discrete – X, SP, Y, CR, S, T (contacts), C (contacts)
- Word – V, Timer current value, Counter current value

In either case, you basically convert the PLC octal address to decimal and add the appropriate MODBUS addresses (as required). The table below shows the exact equation used for each group of data.

Discrete Data Types				
DL06 Memory Type	**PLC Range (Octal)**	**Address (484 Mode)**	**Address (584/984 Mode)**	**MODBUS Data Type**
Global Inputs (GX)	GX0-GX1746	1001 - 1999	10001 - 10999	Input
	GX1747-GX3777	–	11000 - 12048	–
Inputs (X)	X0 – X777	–	12049 - 12304	Input
Special Relays (SP)	SP0 – SP777	–	13073 - 13584	Input
Global Outputs (GY)	GY0 - GX3777	1 - 2048	1 - 2048	–
Outputs (Y)	Y0 – Y777	2049 - 2560	2049 - 2560	Output
Control Relays (CR)	C0 – C1777	3073 - 4096	3073 - 3584	Output
Timer Contacts (T)	T0 – T377	6145 - 6400	6145 - 6400	Output
Counter Contacts (CT)	CT0 – CT177	6401 - 6656	6401 - 6656	Output
Stage Status Bits (S)	S0 – S1777	5121 - 6144	5121 - 6144	Output

Word Data Types			
Registers	PLC Range (Octal)	Input/Holding (484 Mode)*	Input/Holding (584/984 Mode)*
V-memory (Timers)	V0 - V377	3001/4001	30001/40001
V-memory (Counters)	V1000 - V1177	3513/4513	30513/40513
V-memory (Data Words)	V1200 - V1377	3641/4641	30641/40641
	V1400 - V1746	3769/4769	30769/40769
	V1747 - V1777	---	31000/41000
	V2000 - V7377	---	41025
	V10000 - V17777	---	44097

*MODBUS: Function 04

1. Refer to your PLC user manual for the correct memory mapping size of your PLC. Some of the addresses shown above might not pertain to your particular CPU.

2. For an automated MODBUS/Koyo address conversion utility, go to our website **wwww.automationdirect.com**, and down load download the EXCEL file **modbus_conversion.xls** located at: Tech Support > Technical Support Home page.

Example 1: V2100 584/984 Mode

Find the MODBUS address for User V location V2100.

1. Find V-memory in the table.
2. Convert V2100 into decimal (1088).
3. Add the MODBUS starting address for the mode (40001).

PLC Address (Dec.) + Mode Address

V2100 = 1088 decimal

1088 + 40001 = **41089**

For Word Data Types....	PLC Address (Dec.)		+	Appropriate Mode Address		
Timer Current Values (V)	128	V0 – V177	0 – 127	3001	30001	Input Register
Counter Current Values (V)	128	V1200 – V7377	640 – 3839	3001	30001	Input Register
V-memory, user data (V)	1024	V2000 – V3777	1024 – 2047	4001	40001	Holding Register

Example 2: Y20 584/984 Mode

Find the MODBUS address for output Y20.

1. Find Y outputs in the table.
2. Convert Y20 into decimal (16).
3. Add the starting address for the range (2048).
4. Add the MODBUS address for the mode (1).

PLC Addr. (Dec.) + Start Address + Mode

Y20 = 16 decimal

16 + 2048 + 1 = **2065**

Outputs (Y)	320	Y0 - Y477	2048 – 2367	1	1	Coil
Control Relays (CR)	256	C0 - C377	3072 – 3551	1	1	Coil
Timer Contacts (T)	128	T0 - T177	6144 – 6271	1	1	Coil

Example 3: T10 Current Value 484 Mode

Find the MODBUS address to obtain the current value from Timer T10.

PLC Address (Dec.) + Mode Address

TA10 = 8 decimal

8 + 3001 **3009**

1. Find Timer Current Values in the table.

=

2. Convert T10 into decimal (8).

3. Add the MODBUS starting address for the mode (3001).

For Word Data Types....	PLC Address (Dec.)		+	Appropriate Mode Address		
Timer Current Values (V)	128	V0 – V177	0 – 127	3001	30001	Input Register
Counter Current Values (V)	128	V1200 – V7377	512 – 639	3001	30001	Input Register
V-memory, user data (V)	1024	V2000 – V3777	1024 – 2047	4001	40001	Holding Register

Example 4: C54 584/984 Mode

Find the MODBUS address for Control Relay C54.

PLC Addr. (Dec.) + Start Address + Mode

C54 = 44 decimal

44 + 3072 + 1 **3117**

1. Find Control Relays in the table.

2. Convert C54 into decimal (44).

=

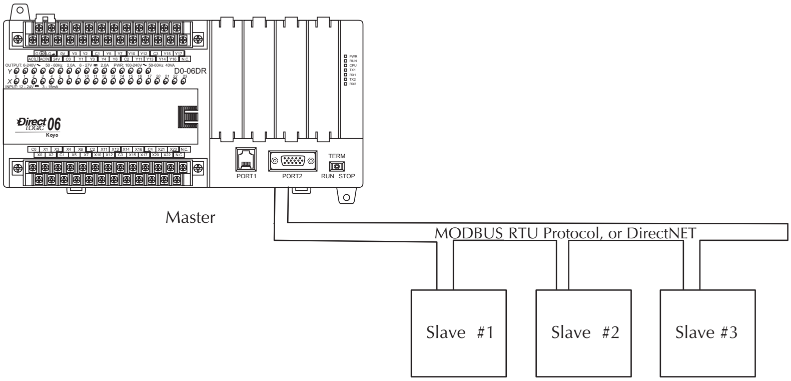
3. Add the starting address for the range (3072).

4. Add the MODBUS address for the mode (1).

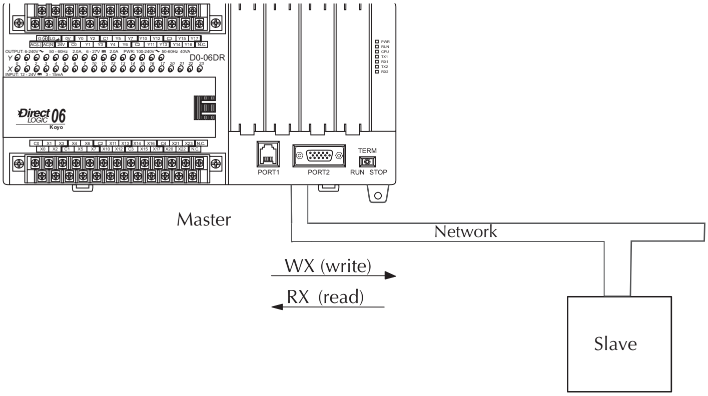
Outputs (Y)	320	Y0 – Y477	2048 – 2367	1	1	Coil
Control Relays (CR)	256	C0 – C377	3072 – 3551	1	1	Coil
Timer Contacts (T)	128	T0– T177	6144 – 6271	1	1	Coil

Network Master Operation

This section describes how the DL06 can communicate on a MODBUS or *Direct*NET network as a master. For MODBUS networks, it uses the MODBUS RTU protocol, which must be interpreted by all the slaves on the network. Both MODBUS and *Direct*Net are single master/multiple slave networks. The master is the only member of the network that can initiate requests on the network. This section teaches you how to design the required ladder logic for network master operation.



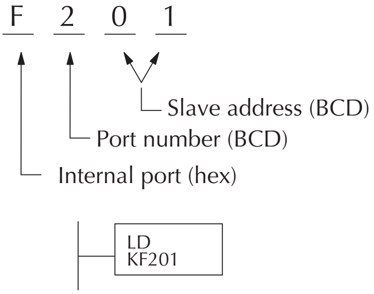
When using the DL06 PLC as the master station, simple RLL instructions are used to initiate the requests. The WX instruction initiates network write operations, and the RX instruction initiates network read operations. Before executing either the WX or RX commands, we will need to load data related to the read or write operation onto the CPU's accumulator stack. When the WX or RX instruction executes, it uses the information on the stack combined with data in the instruction box to completely define the task, which goes to the port.



The following step-by-step procedure will provide you the information necessary to set up your ladder program to receive data from a network slave.

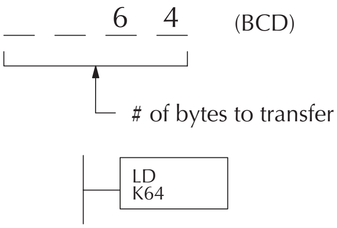
Step 1: Identify Master Port # and Slave

The first Load (LD) instruction identifies the communications port number on the network master (DL06) and the address of the slave station. This instruction can address up to 99 MODBUS slaves, or 90 *Direct*NET slaves. The format of the word is shown to the right. The "F2" in the upper byte indicates the use of the right port of the DL06 PLC, port number 2. The lower byte contains the slave address number in BCD (01 to 99).



Step 2: Load Number of Bytes to Transfer

The second Load (LD) instruction determines the number of bytes which will be transferred between the master and slave in the subsequent WX or RX instruction. The value to be loaded is in BCD format (decimal), from 1 to 128 bytes.



The number of bytes specified also depends on the type of data you want to obtain. For example, the DL06 Input points can be accessed by V-memory locations or as X input locations. However, if you only want X0 – X27, you'll have to use the X input data type because the V-memory locations can only be accessed in 2-byte increments. The following table shows the byte ranges for the various types of *Direct*LOGIC products.

DL05 / 06 / 205 / 350 / 405 Memory	Bits per unit	Bytes
V-memory T / C current value	16 16	2 2
Inputs (X, SP)	8	1
Outputs (Y, C, Stage, T/C bits)	8	1
Scratch Pad Memory	8	1
Diagnostic Status	8	1

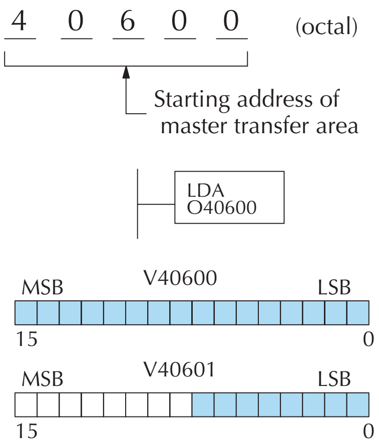
DL330 / 340 Memory	Bits per unit	Bytes
Data registers T / C accumulator	8 16	1 2
I/O, internal relays, shift register bits, T/C bits, stage bits	1	1
Scratch Pad Memory	8	1
Diagnostic Status(5 word R/W)	16	10

Step 3: Specify Master Memory Area

The third instruction in the RX or WX sequence is a Load Address (LDA) instruction. Its purpose is to load the starting address of the memory area to be transferred. Entered as an octal number, the LDA instruction converts it to hex and places the result in the accumulator.

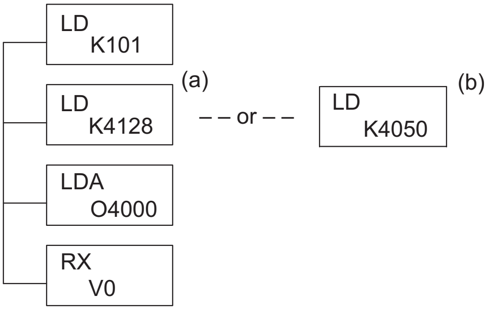
For a WX instruction, the DL06 CPU sends the number of bytes previously specified from its memory area beginning at the LDA address specified.

For an RX instruction, the DL06 CPU reads the number of bytes previously specified from the slave, placing the received data into its memory area beginning at the LDA address specified.



NOTE: Since V-memory words are always 16 bits, you may not always use the whole word. For example, if you only specify 3 bytes and you are reading Y outputs from the slave, you will only get 24 bits of data. In this case, only the 8 least significant bits of the last word location will be modified. The remaining 8 bits are not affected.

When using MODBUS, the RX instructions use function 3 by default, to read MODBUS holding registers (Address 40001). The DL05/DL06, DL250-1/260, DL350, DL454 support function 04, read input register (Address 30001). To use function 04, put the number “4” into the most significant position (4xxx) of the total number of bytes. Four digits must be entered for the instruction to work properly with this mode.

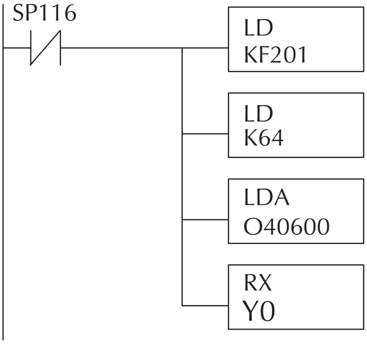


The (a) K4128 indicates the instruction will read 128 bytes of MODBUS input registers (30001). The (b) K4050 indicates the instruction will read 50 bytes of MODBUS input registers (30001). The value of 4 in the most significant position will cause the RX to use MODBUS function 4 (30001 range).

Step 4: Specify Slave Memory Area

The last instruction in our sequence is the WX or RX instruction itself. Use WX to write to the slave, and RX to read from the slave. All four of our instructions are shown to the right. In the last instruction, you must specify the starting address and a valid data type for the slave.

- ***Direct*NET slaves – specify the same address in the WX and RX instruction as the slave’s native I/O address**
- MODBUS DL405, DL205, or DL06 slaves – specify the same address in the WX and RX instruction as the slave’s native I/O address
- MODBUS 305 slaves – use the following table to convert DL305 addresses to MODBUS addresses

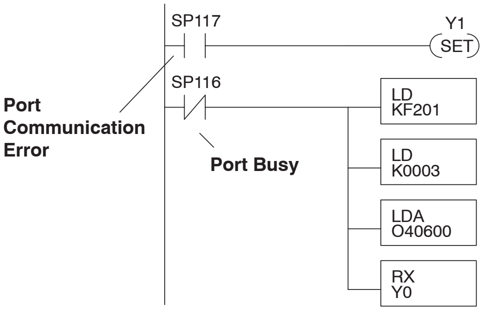


DL305 Series CPU Memory Type–to–MODBUS Cross Reference (excluding 350 CPU)

PLC Memory Type	PLC Base Address	MODBUS Base Address	PLC Memory Type	PLC Base Address	MODBUS Base Address
TMR/CNT Current Values	R600	V0	TMR/CNT Status Bits	CT600	GY600
I/O Points	IO 000	GY0	Control Relays	CR160	GY160
Data Registers	R401,R400	V100	Shift Registers	SR400	GY400
Stage Status Bits (D3-330P only)	S0	GY200			

Communications from a Ladder Program

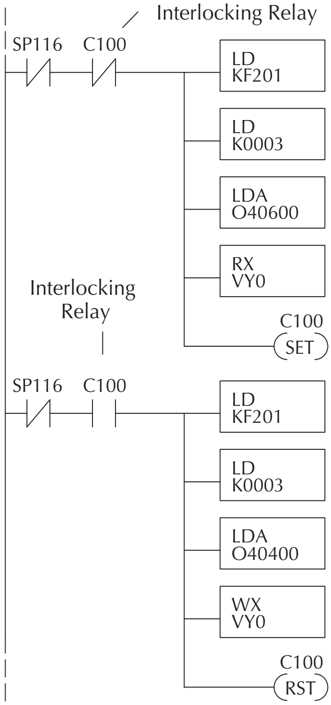
Typically network communications will last longer than 1 scan. The program must wait for the communications to finish before starting the next transaction.



Port 2, which can be a master, has two Special Relay contacts associated with it (see Appendix D for comm port special relays). One indicates “Port busy”(SP116), and the other indicates ”Port Communication Error”(SP117). The example above shows the use of these contacts for a network master that only reads a device (RX). The “Port Busy” bit is on while the PLC communicates with the slave. When the bit is off the program can initiate the next network request.

The “Port Communication Error” bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an RX or WX instruction is executed

Multiple Read and Write Interlocks



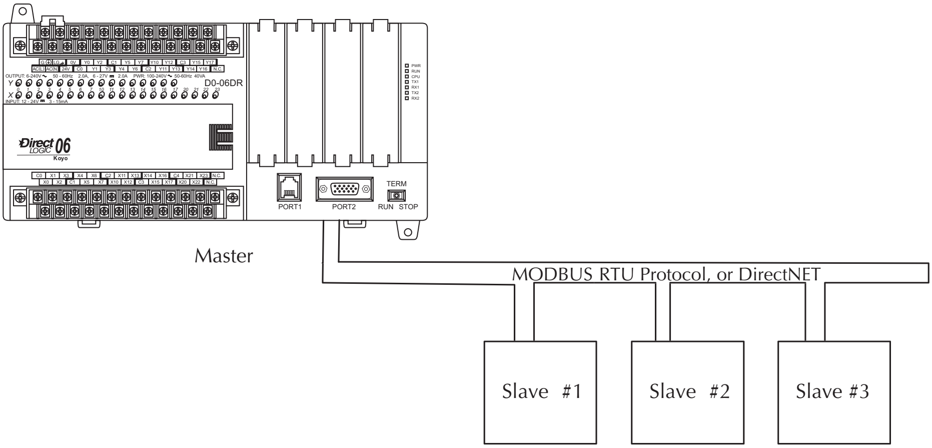
If you are using multiple reads and writes in the RLL program, you have to interlock the routines to make sure all the routines are executed. If you don’t use the interlocks, then the CPU will only execute the first routine. This is because each port can only handle one transaction at a time.

In the example to the right, after the RX instruction is executed, C100 is set. When the port has finished the communication task, the second routine is executed and C100 is reset.

If you’re using RLL^{PLUS} Stage Programming, you can put each routine in a separate program stage to ensure proper execution and switch from stage to stage allowing only one of them to be active at a time.

Network Master Operation (using MRX and MWX Instructions)

This section describes how the DL06 can communicate on a MODBUS RTU network as a master using the MRX and MWX read/write instructions. These instructions allow you to enter native MODBUS addressing in your ladder logic program with no need to perform octal to decimal conversions. MODBUS is a single master/multiple slave network. The master is the only member of the network that can initiate requests on the network. This section teaches you how to design the required ladder logic for network master operation.



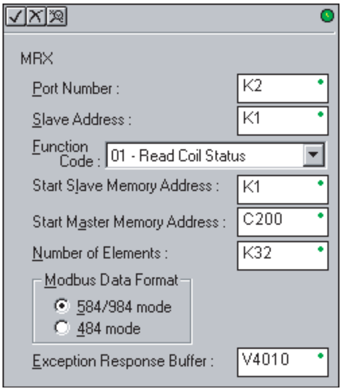
MODBUS Function Codes Supported

The MODBUS function code determines whether the access is a read or a write, and whether to access a single data point or a group of them. The DL06 supports the MODBUS function codes described below.

MODBUS Function Code	Function	DL06 Data Types Available
01	Read a group of coils	Y, CR, T, CT
02	Read a group of inputs	X, SP
05	Set / Reset a single coil (slave only)	Y, CR, T, CT
15	Set / Reset a group of coils	Y, CR, T, CT
03, 04	Read a value from one or more registers	V
06	Write a value into a single register (slave only)	V
07	Read Exception Status	V
08	Diagnostics	V
16	Write a value into a group of registers	V

MODBUS Read from Network(MRX)

The MODBUS Read from Network (MRX) instruction is used by the DL06 network master to read a block of data from a connected slave device and to write the data into V–memory addresses within the master. The instruction allows the user the to specify the MODBUS Function Code, slave station address, starting master and slave memory addresses, number of elements to transfer, MODBUS data format and the Exception Response Buffer.



MRX

Port Number : K2

Slave Address : K1

Function Code : 01 - Read Coil Status

Start Slave Memory Address : K1

Start Master Memory Address : C200

Number of Elements : K32

Modbus Data Format

● 584/984 mode

○ 484 mode

Exception Response Buffer : V4010

- **Port Number:** must be DL06 Port 2 (K2)
- **Slave Address:** specify a slave station address (0–247)
- **Function Code**: The following MODBUS function codes are supported by the MRX instruction:

  01 – Read a group of coils

  02 – Read a group of inputs

  03 – Read holding registers

  04 – Read input registers

  07 – Read Exception status

  08 – Diagnostics
- **Start Slave Memory Address**: specifies the starting slave memory address of the data to be read. See the table on the following page.
- **Start Master Memory Address**: specifies the starting memory address in the master where the data will be placed. See the table on the following page.
- **Number of Elements**: specifies how many coils, input, holding registers or input register will be read. See the table on the following page.
- **MODBUS Data Format**: specifies MODBUS 584/984 or 484 data format to be used
- **Exception Response Buffer**: specifies the master memory address where the Exception Response will be placed. See the table on the following page.

MRX Slave Memory Address

MRX Slave Address Ranges		
Function Code	**MODBUS Data Format**	**Slave Address Range(s)**
01 – Read Coil	484 Mode	1–999
01 – Read Coil	584/984 Mode	1–65535
02 – Read Input Status	484 Mode	1001–1999
02 – Read Input Status	584/984 Mode	10001–19999 (5 digit) or 100001–165535 (6 digit)
03 – Read Holding Register	484 Mode	4001–4999
03 – Read Holding Register	584/984	40001–49999 (5 digit) or 4000001–465535 (6 digit)
04 – Read Input Register	484 Mode	3001–3999
04 – Read Input Register	584/984 Mode	30001–39999 (5 digit) or 3000001–365535 (6 digit)
07 – Read Exception Status	484 and 584/984 Mode	n/a
08 – Diagnostics	484 and 584/984 Mode	0–65535

MRX Master Memory Addresses

MRX Master Memory Address Ranges	
Operand Data Type	**DL06 Range**
Inputs X	0–1777
Outputs Y	0–1777
Control Relays C	0–3777
Stage Bits S	0–1777
Timer Bits T	0–377
Counter Bits CT	0–377
Special Relays SP	0–777
V–memory V	All
Global Inputs GX	0–3777
Global Outputs GY	0–3777

MRX Number of Elements

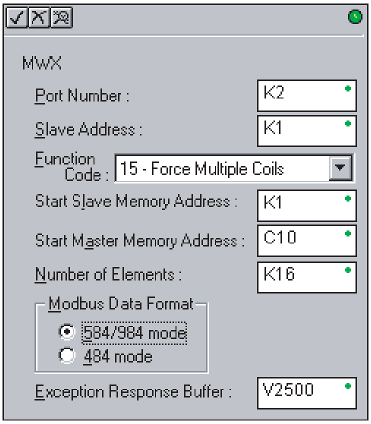
MRX Number of Elements		
Operand Data Type		**DL06 Range**
V–memory	V	All
Constant	K	1–2000

MRX Exception Response Buffer

MRX Exception Response Buffer		
Operand Data Type		**DL06 Range**
V–memory	V	All

MODBUS Write to Network (MWX)

The MODBUS Write to Network (MWX) instruction is used to write a block of data from the network masters's (DL06) memory to MODBUS memory addresses within a slave device on the network. The instruction allows the user to specify the MODBUS Function Code, slave station address, starting master and slave memory addresses, number of elements to transfer, MODBUS data format and the Exception Response Buffer.



MWX

Port Number : K2

Slave Address : K1

Function Code : 15 - Force Multiple Coils

Start Slave Memory Address : K1

Start Master Memory Address : C10

Number of Elements : K16

Modbus Data Format

● 584/984 mode

○ 484 mode

Exception Response Buffer : V2500

- **Port Number**: must be DL06 Port 2 (K2)
- **Slave Address**: specify a slave station address (0–247)
- **Function Code**: The following MODBUS function codes are supported by the MWX instruction:

  05 – Force Single coil

  06 – Preset Single Register

  08 – Diagnostics

  15 – Force Multiple Coils

  16 – Preset Multiple Registers
- **Start Slave Memory Address:** specifies the starting slave memory address where the data will be written.
- **Start Master Memory Address:** specifies the starting address of the data in the master that is to written to the slave.
- **Number of Elements**: specifies how many consecutive coils or registers will be written to. This field is only active when either function code 15 or 16 is selected.
- **MODBUS Data Format**: specifies MODBUS 584/984 or 484 data format to be used.
- **Exception Response Buffer**: specifies the master memory address where the Exception Response will be placed.

MWX Slave Memory Address

MWX Slave Address Ranges		
Function Code	**MODBUS Data Format**	**Slave Address Range(s)**
05 – Force Single Coil	484 Mode	1–999
05 – Force Single Coil	584/984 Mode	1–65535
06 – Preset Single Register	484 Mode	4001–4999
06 – Preset Single Register	84/984 Mode	40001–49999 (5 digit) or 400001–465535 (6 digit)
08 – Diagnostics	484 and 584/984 Mode	0–65535
15 – Force Multiple Coils	484	1–999
15 – Force Multiple Coils	585/984 Mode	1–65535
16 – Preset Multiple Registers	484 Mode	4001–4999
16 – Preset Multiple Registers	584/984 Mode	40001–49999 (5 digit) or 4000001–465535 (6 digit)

MWX Master Memory Addresses

MWX Master Memory Address Ranges		
Operand Data Type		**DL06 Range**
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage Bits	S	0–1777
Timer Bits	T	0–377
Counter Bits	CT	0–177
Special Relays	SP	0–777
V–memory	V	All
Global Inputs	GX	0–3777
Global Outputs	GY	0–3777

MWX Number of Elements

MWX Number of Elements		
Operand Data Type		**DL06 Range**
V–memory	V	All
Constant	K	1–2000

MWX Exception Response Buffer

MWX Exception Response Buffer		
Operand Data Type		DL06 Range
V–memory	V	All

MRX/MWX Example in *Direct*SOFT

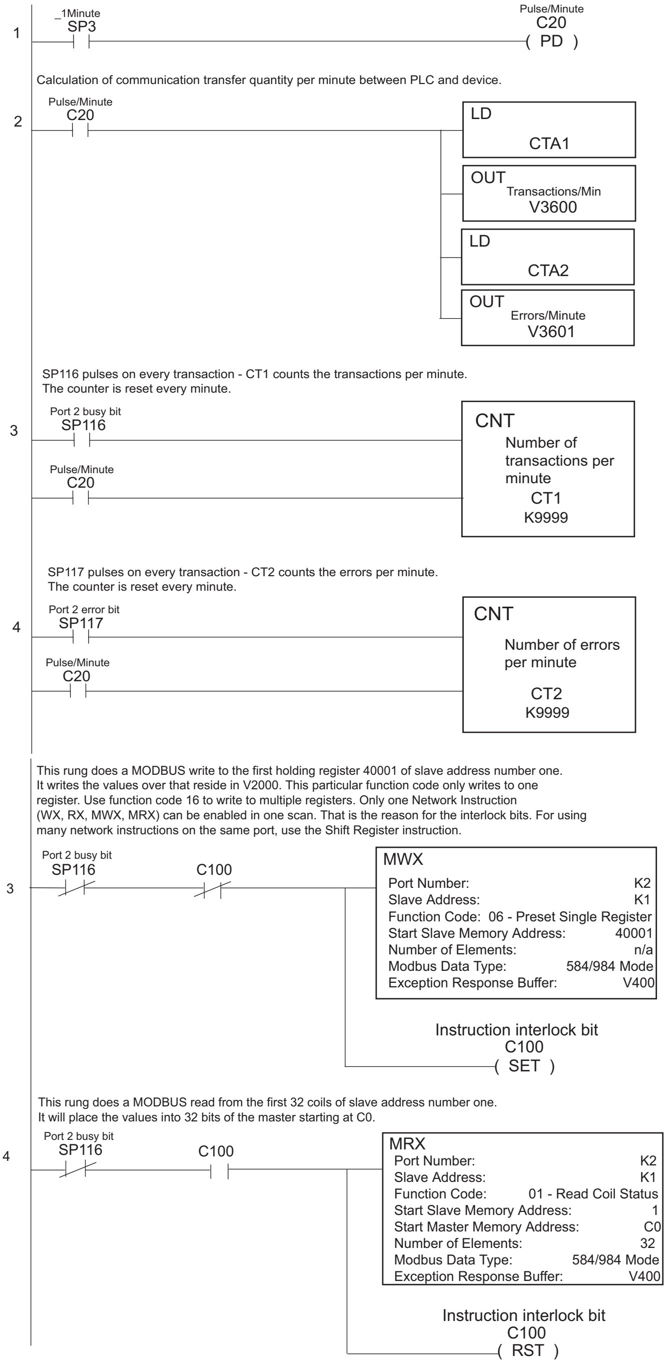
DL06 port 2 has two Special Relay contacts associated with it (see Appendix D for comm port special relays). One indicates “Port busy”(SP116), and the other indicates ”Port Communication Error”(SP117). The “Port Busy” bit is on while the PLC communicates with the slave. When the bit is off the program can initiate the next network request. The “Port Communication Error” bit turns on when the PLC has detected an error and use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an MRX or MWX instruction is executed. Typically network communications will last longer than 1 CPU scan. The program must wait for the communications to finish before starting the next transaction.

The “Port Communication Error” bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an RX or WX instruction is executed.

Multiple Read and Write Interlocks

If you are using multiple reads and writes in the RLL program, you have to interlock the routines to make sure all the routines are executed. If you don’t use the interlocks, then the CPU will only execute the first routine. This is because each port can only handle one transaction at a time. In the example below, after the MRX instruction is executed, C100 is set. When the port has finished the communication task, the second routine is executed and C100 is reset. If you’re using RLL^{PLUS} Stage Programming, you can put each routine in a separate program stage to ensure proper execution and switch from stage to stage allowing only one of them to be active at a time.

See example on the next page.



Calculation of communication transfer quantity per minute between PLC and device.

SP116 pulses on every transaction - CT1 counts the transactions per minute.
The counter is reset every minute.

SP117 pulses on every transaction - CT2 counts the errors per minute.
The counter is reset every minute.

This rung does a MODBUS write to the first holding register 40001 of slave address number one. It writes the values over that reside in V2000. This particular function code only writes to one register. Use function code 16 to write to multiple registers. Only one Network Instruction (WX, RX, MWX, MRX) can be enabled in one scan. That is the reason for the interlock bits. For using many network instructions on the same port, use the Shift Register instruction.

This rung does a MODBUS read from the first 32 coils of slave address number one. It will place the values into 32 bits of the master starting at C0.

1
_1Minute
SP3
Pulse/Minute
C20
(PD)
2
Pulse/Minute
C20
LD
CTA1
OUT
Transactions/Min
V3600
LD
CTA2
OUT
Errors/Minute
V3601
3
Port 2 busy bit
SP116
Pulse/Minute
C20
CNT
Number of transactions per minute
CT1
K9999
4
Port 2 error bit
SP117
Pulse/Minute
C20
CNT
Number of errors per minute
CT2
K9999
3
Port 2 busy bit
SP116
C100
MWX
Port Number: K2
Slave Address: K1
Function Code: 06 - Preset Single Register
Start Slave Memory Address: 40001
Number of Elements: n/a
Modbus Data Type: 584/984 Mode
Exception Response Buffer: V400
Instruction interlock bit
C100
(SET)
4
Port 2 busy bit
SP116
C100
MRX
Port Number: K2
Slave Address: K1
Function Code: 01 - Read Coil Status
Start Slave Memory Address: 1
Start Master Memory Address: C0
Number of Elements: 32
Modbus Data Type: 584/984 Mode
Exception Response Buffer: V400
Instruction interlock bit
C100
(RST)

STANDARD RLL INSTRUCTIONS

CHAPTER

5

In This Chapter

Introduction

DL06 Micro PLCs offer a wide variety of instructions to perform many different types of operations. This chapter shows you how to use each standard Relay Ladder Logic (RLL) instruction. In addition to these instructions, you may also need to refer to the Drum instruction in Chapter 6, the Stage programming instructions in Chapter 7, PID in Chapter 8, LCD in Chapter 10 and programming for analog modules in D0-OPTIONS-M.

There are two ways to quickly find the instruction you need.

- If you know the instruction category (Boolean, Comparative Boolean, etc.), just use the title at the top of the page to find the pages that discuss the instructions in that category.
- If you know the individual instruction name, use the following table to find the page(s) that discusses the instruction.

Instruction	Page	Instruction	Page
Accumulating Fast Timer (TMRAF)	5–42	And Store (AND STR)	5–16
Accumulating Timer (TMRA)	5–42	And with Stack (ANDS)	5–72
Add (ADD)	5–86	Arc Cosine Real (ACOSR)	5–119
Add Binary (ADDB)	5–99	Arc Sine Real (ASINR)	5–118
Add Binary Double (ADDBD)	5–100	Arc Tangent Real (ATANR)	5–119
Add Binary Top of Stack (ADDBS)	5–114	ASCII Clear Buffer (ACRB)	5–228
Add Double (ADDD)	5–87	ASCII Compare (CMPV)	5–220
Add Formatted (ADDF)	5–106	ASCII Constant (ACON)	5–187
Add Real (ADDR)	5–88	ASCII Extract (AEX)	5–219
Add to Top (ATT)	5–162	ASCII Find (AFIND)	5–216
Add Top of Stack (ADDS)	5–110	ASCII Input (AIN)	5–212
And (AND)	5–14	ASCII Print from V–memory (PRINTV)	5–226
And Bit-of-Word (AND)	5–15	ASCII Print to V–memory (VPRINT)	5–221
And (AND)	5–31	ASCII Swap Bytes (SWAPB)	5–227
AND (AND logical)	5–69	ASCII to HEX (ATH)	5–134
And Double (ANDD)	5–70	Binary (BIN)	5–127
And Formatted (ANDF)	5–71	Binary Coded Decimal (BCD)	5–128
And If Equal (ANDE)	5–28	Binary to Real Conversion (BTOR)	5–131
And If Not Equal (ANDNE)	5–28	Compare (CMP)	5–81
And Immediate (ANDI)	5–33	Compare Double (CMPD)	5–82
AND Move (ANDMOV)	5–167	Compare Formatted (CMPF)	5–83
And Negative Differential (ANDND)	5–22	Compare Real Number (CMPR)	5–85
And Not (ANDN)	5–14	Compare with Stack (CMPS)	5–84
And Not Bit-of-Word (ANDN)	5–15	Cosine Real (COSR)	5–118
And Not (ANDN)	5–31	Counter (CNT)	5–45
And Not Immediate (ANDNI)	5–33	Data Label (DLBL)	5–187
And Positive Differential (ANDPD)	5–22	Date (DATE)	5–171

Instruction	Page	Instruction	Page
Decode (DECO)	5–126	Load Accumulator Indexed from Data Constants (LDSX)	5–62
Decrement (DEC)	5–98	Load Address (LDA)	5–60
Decrement Binary (DECB)	5–105	Load Double (LDD)	5–58
Degree Real Conversion (DEGR)	5–133	Load Formatted (LDF)	5–59
Disable Interrupts (DISI)	5–184	Load Immediate (LDI)	5–37
Divide (DIV)	5–95	Load Immediate Formatted (LDIF)	5–38
Divide Binary (DIVB)	5–104	Load Label (LDLBL)	5–142
Divide Binary by Top OF Stack (DIVBS)	5–117	Load Real Number (LDR)	5–63
Divide by Top of Stack (DIVS)	5–113	Master Line Reset (MLR)	5–181
Divide Double (DIVD)	5–96	Master Line Set (MLS)	5–181
Divide Formatted (DIVF)	5–109	MODBUS Read from Network (MRX)	5–204
Divide Real (DIVR)	5–97	MODBUS Write to Network (MWX)	5–207
Enable Interrupts (ENI)	5–183	Move Block (MOVBLK)	5-189
Encode (ENCO)	5–125	Move (MOV)	5–141
End (END)	5–173	Move Memory Cartridge (MOVMC)	5–142
Exclusive Or (XOR)	5–77	Multiply (MUL)	5–92
Exclusive Or Double (XORD)	5–78	Multiply Binary (MULB)	5–103
Exclusive Or Formatted (XORF)	5–79	Multiply Binary Top of Stack (MULBS)	5–116
Exclusive OR Move (XORMOV)	5–167	Multiply Double (MULD)	5–93
Exclusive Or with Stack (XORS)	5–80	Multiply Formatted (MULF)	5–108
Fault (FAULT)	5–186	Multiply Real (MULR)	5–94
Fill (FILL)	5–146	Multiply Top of Stack (MULS)	5–112
Find (FIND)	5–147	No Operation (NOP)	5–173
Find Block (FINDB)	5–169	Not (NOT)	5–19
Find Greater Than (FDGT)	5–148	Numerical Constant (NCON)	5–187
For / Next (FOR) (NEXT)	5–176	Or (OR)	5–12
Goto Label (GOTO) (LBL)	5–175	Or (OR)	5–30
Goto Subroutine (GTS) (SBR)	5–178	Or (OR logical)	5–73
Gray Code (GRAY)	5–138	Or Bit-of-Word (OR)	5–13
HEX to ASCII (HTA)	5–135	Or Double (ORD)	5–74
Increment (INC)	5–98	Or Formatted (ORF)	5–75
Increment Binary (INCB)	5–105	Or If Equal (ORE)	5–27
Interrupt (INT)	5–183	Or If Not Equal (ORNE)	5–27
Interrupt Return (IRT)	5–183	Or Immediate (ORI)	5–32
Interrupt Return Conditional (IRTC)	5–183	OR Move (ORMOV)	5–167
Invert (INV)	5–129	Or Negative Differential (ORND)	5–21
LCD	5–200	Or Not (ORN)	5–12
Load (LD)	5–57	Or Not (ORN)	5–30
Load Accumulator Indexed (LDX)	5–61	Or Not Bit-of-Word (ORN)	5–13

Instruction	Page
Or Not Immediate (ORNI)	5–32
Or Out (OROUT)	5–17
Or Out Immediate (OROUTI)	5–34
Or Positive Differential (ORPD)	5–21
Or Store (ORSTR)	5–16
Or with Stack (ORS)	5–76
Out (OUT)	5–17
Out Bit-of-Word (OUT)	5–18
Out (OUT)	5–64
Out Double (OUTD)	5–64
Out Formatted (OUTF)	5–65
Out Immediate (OUTI)	5–34
Out Immediate Formatted (OUTIF)	5–35
Out Indexed (OUTX)	5–67
Out Least (OUTL)	5–68
Out Most (OUTM)	5–68
Pause (PAUSE)	5–25
Pop (POP)	5–65
Positive Differential (PD)	5–19
Print Message (PRINT)	5–190
Radian Real Conversion (RADR)	5–133
Read from Intelligent I/O Module (RD)	5-194
Read from Network (RX)	5–196
Real to Binary Conversion (RTOB)	5–132
Remove from Bottom (RFB)	5–153
Remove from Table (RFT)	5–159
Reset (RST)	5–23
Reset Bit-of-Word (RST)	5–24
Reset Immediate (RSTI)	5–36
Reset Watch Dog Timer (RSTWT)	5–174
Rotate Left (ROTL)	5–123
Rotate Right (ROTR)	5–124
RSTBIT	5–144
Segment (SEG)	5–137
Set (SET)	5–23
Set Bit-of-Word (SET)	5–24
Set Immediate (SETI)	5–36
SETBIT	5–144
Shift Left (SHFL)	5–121
Shift Register (SR)	5–51
Shift Right (SHFR)	5–122
Shuffle Digits (SFLDGT)	5–139
Sine Real (SINR)	5–118
Source to Table (STT)	5–156
Square Root Real (SQRTR)	5–119
Stage Counter (SGCNT)	5–47
Stop (STOP)	5–173
Store (STR)	5–10
Store (STR)	5–29
Store Bit-of-Word (STRB)	5–11
Store If Equal (STRE)	5–26
Store If Not Equal (STRNE)	5–26
Store Immediate (STRI)	5–32
Store Negative Differential (STRND)	5–20
Store Not (STRN)	5–29
Store Not (STRN)	5–10
Store Not Bit-of-Word (STRNB)	5–11
Store Not Immediate (STRNI)	5–32
Store Positive Differential (STRPD)	5–20
Subroutine Return (RT)	5–178
Subroutine Return Conditional (RTC)	5–178
Subtract (SUB)	5–89
Subtract Binary (SUBB)	5–101
Subtract Binary Double (SUBBD)	5–102
Subtract Binary Top of Stack (SUBBS)	5–115
Subtract Double (SUBD)	5–90
Subtract Formatted (SUBF)	5–107
Subtract Real (SUBR)	5–91
Subtract Top of Stack (SUBS)	5–111
Sum (SUM)	5–120
Swap (SWAP)	5–170
Table Shift Left (TSHFL)	5–165
Table Shift Right (TSHFR)	5–165
Table to Destination (TTD)	5–150
Tangent Real (TANR)	5–118
Ten's Complement (BCDCPL)	5–130
Time (TIME)	5–172
Timer (TMR) and Timer Fast (TMRF)	5–40
Up Down Counter (UDC)	5–49
Write to Intelligent I/O Module (WT)	5-195
Write to Network (WX)	5–198

Using Boolean Instructions

Do you ever wonder why so many PLC manufacturers always quote the scan time for a 1K Boolean program? Simple. Most programs utilize many Boolean instructions. These are typically very simple instructions designed to join input and output contacts in various series and parallel combinations. Our *Direct*SOFT software is a similar program. It uses graphic symbols to develop a program; therefore, you don't necessarily have to know the instruction mnemonics in order to develop your program.

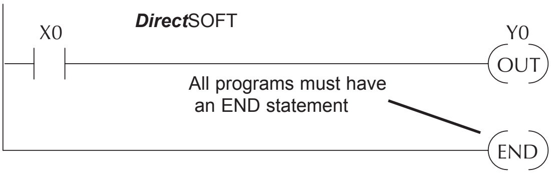
Many of the instructions in this chapter are not program instructions used in *Direct*SOFT, but are implied. In other words, they are not actually keyboard commands, however, they can be seen in a Mnemonic View of the program once the *Direct*SOFT program has been developed and accepted (compiled). Each instruction listed in this chapter will have a small chart to indicate how the instruction is used with *Direct*SOFT and the HPP.

DS	Implied
HPP	Used

The following paragraphs show how these instructions are used to build simple ladder programs.

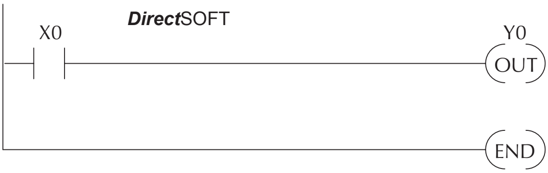
END Statement

All DL06 programs require an END statement as the last instruction. This tells the CPU that this is the end of the program. Normally, any instructions placed after the END statement will not be executed. There are exceptions to this, such as interrupt routines, etc. This chapter will discuss the instruction set in detail.



Simple Rungs

You use a contact to start rungs that contain both contacts and coils. The boolean instruction that does this is called a Store or, STR instruction. The output point is represented by the Output or, OUT instruction. The following example shows how to enter a single contact and a single output coil.

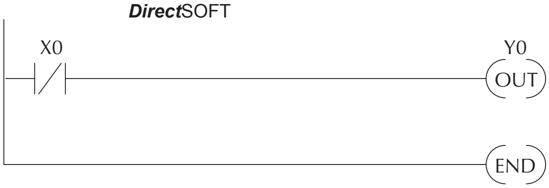


Handheld Mnemonics

```
STR X0
OUT Y0
END
```

Normally Closed Contact

Normally closed contacts are also very common. This is accomplished with the Store Not, or STRN instruction. The following example shows a simple rung with a normally closed contact.

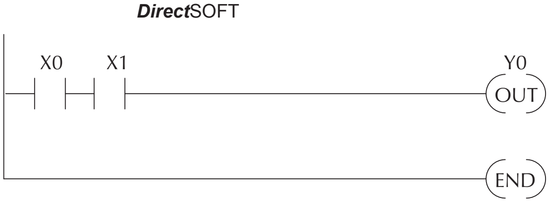


Handheld Mnemonics

```
STRN X0
OUT Y0
END
```

Contacts in Series

Use the AND instruction to join two or more contacts in series. The following example shows two contacts in series and a single output coil. The instructions used would be STR X0, AND X1, followed by OUT Y0.

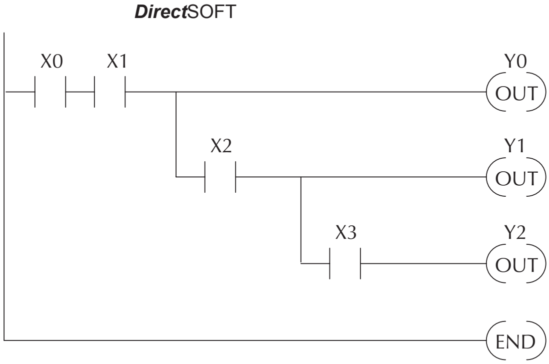


Handheld Mnemonics

```
STR X0
AND X1
OUT Y0
END
```

Midline Outputs

Sometimes, it is necessary to use midline outputs to get additional outputs that are conditional on other contacts. The following example shows how you can use the AND instruction to continue a rung with more conditional outputs.

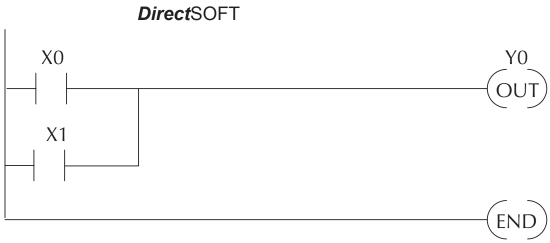


Handheld Mnemonics

```
STR X0
AND X1
OUT Y0
AND X2
OUT Y1
AND X3
OUT Y2
END
```

Parallel Elements

You may also have to join contacts in parallel. The OR instruction allows you to do this. The following example shows two contacts in parallel and a single output coil. The instructions would be STR X0, OR X1, followed by OUT Y0.

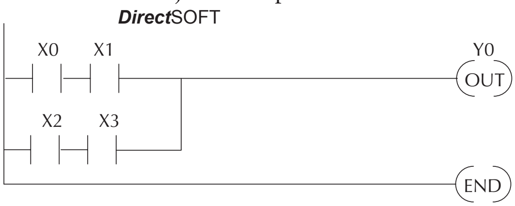


Handheld Mnemonics

```
STR X0
OR X1
OUT Y0
END
```

Joining Series Branches in Parallel

Quite often, it is necessary to join several groups of series elements in parallel. The Or Store (ORSTR) instruction allows this operation. The following example shows a simple network consisting of series elements joined in parallel.

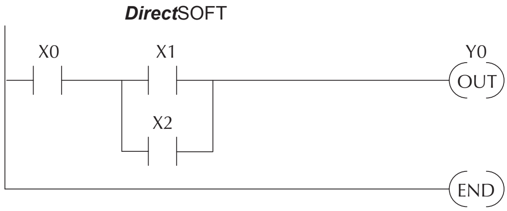


Handheld Mnemonics

```
STR X0
AND X1
STR X2
AND X3
ORSTR
OUT Y0
END
```

Joining Parallel Branches in Series

You can also join one or more parallel branches in series. The And Store (ANDSTR) instruction allows this operation. The following example shows a simple network with contact branches in series with parallel contacts.

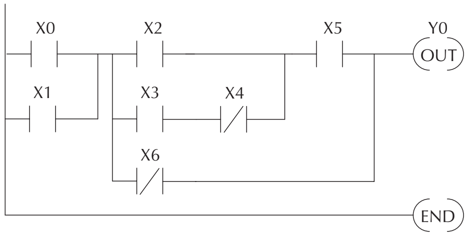


Handheld Mnemonics

```
STR X0
STR X1
OR X2
ANDSTR
OUT Y0
END
```

Combination Networks

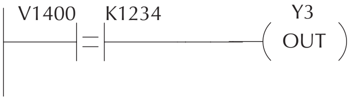
You can combine the various types of series and parallel branches to solve almost any application problem.The following example shows a simple combination network.



Comparative Boolean

Some PLC manufacturers make it really difficult to do a simple comparison of two numbers. Some of them require you to move the data all over the place before you can actually perform the comparison. The DL06 Micro PLCs provide Comparative Boolean instructions that allow you to quickly and easily solve this problem. The Comparative Boolean provides evaluation of two BCD values using boolean contacts. The valid evaluations are: equal to, not equal to, equal to or greater than, and less than.

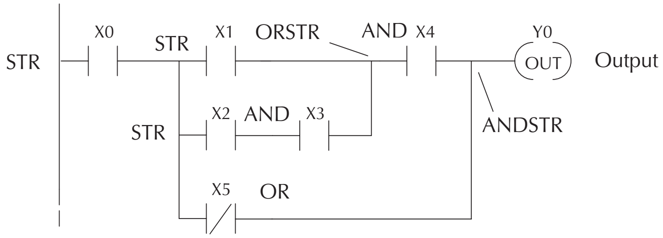
In the example, when the BCD value in V-memory location V1400 is equal to the constant value 1234, Y3 will energize.



Boolean Stack

There are limits to how many elements you can include in a rung. This is because the DL06 PLCs use an 8-level boolean stack to evaluate the various logic elements. The boolean stack is a temporary storage area that solves the logic for the rung. Each time the program encounters a STR instruction, the instruction is placed on the top of the stack. Any other STR instructions already on the boolean stack are pushed down a level. The ANDSTR, and ORSTR instructions combine levels of the boolean stack when they are encountered. An error will occur during program compilation if the CPU encounters a rung that uses more than the eight levels of the boolean stack.

The following example shows how the boolean stack is used to solve boolean logic.



STR X0

1	STR X0
2	
3	
4	

STR X1

1	STR X1
2	STR X0
3	
4	

STR X2

1	STR X2
2	STR X1
3	STR X0
4	

AND X3

1	X2 AND X3
2	STR X1
3	STR X0
4	

ORSTR

1	X1 or (X2 AND X3)
2	STR X0
3	

AND X4

1	X4 AND {X1 or (X2 AND X3)}
2	STR X0
3	

ORNOT X5

1	NOT X5 OR X4 AND {X1 OR (X2 AND X3)}
2	STR X0
3	

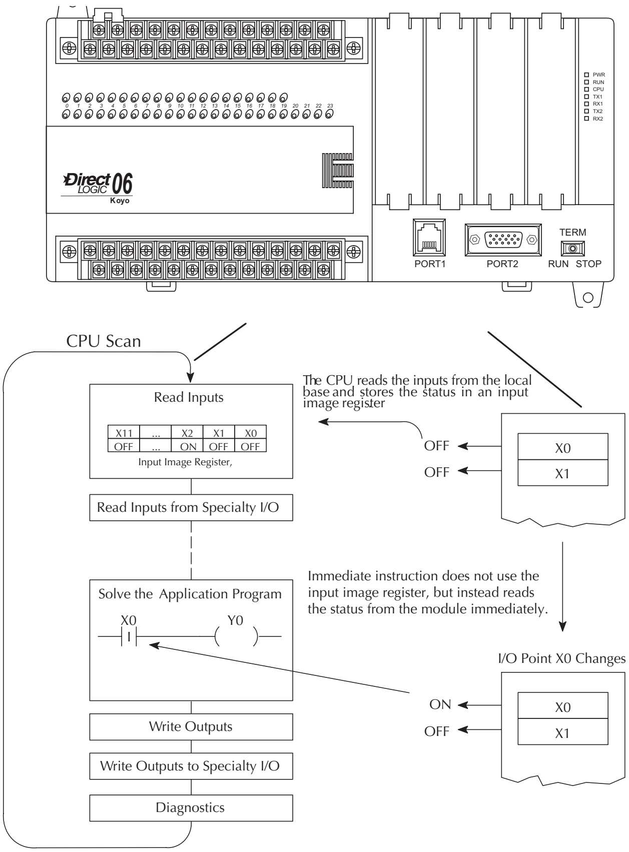
ANDSTR

1	XO AND (NOT X5 or X4) AND {X1 or (X2 AND X3)}
2	
3	

Immediate Boolean

The DL06 Micro PLCs can usually complete an operation cycle in a matter of milliseconds. However, in some applications you may not be able to wait a few milliseconds until the next I/O update occurs. The DL06 PLCs offer Immediate input and outputs which are special boolean instructions that allow reading directly from inputs and writing directly to outputs during the program execution portion of the CPU cycle. You may recall that this is normally done during the input or output update portion of the CPU cycle. The immediate instructions take longer to execute because the program execution is interrupted while the CPU reads or writes the I/O point. This function is not normally done until the read inputs or the write outputs portion of the CPU cycle.

***NOTE**: Even though the immediate input instruction reads the most current status from the input point, it only uses the results to solve that one instruction. It does not use the new status to update the image register. Therefore, any regular instructions that follow will still use the image register values. Any immediate instructions that follow will access the I/O again to update the status. The immediate output instruction will write the status to the I/O and update the image register.*

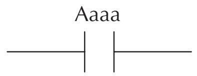

PWR
RUN
CPU
TX1
RX1
TX2
RX2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
DirectLOGIC 06
Koyo
TERM
PORT1
PORT2
RUN STOP
CPU Scan
Read Inputs
X11 ... X2 X1 X0
OFF ... ON OFF OFF
Input Image Register,
Read Inputs from Specialty I/O
Solve the Application Program
X0
Y0
Write Outputs
Write Outputs to Specialty I/O
Diagnostics
The CPU reads the inputs from the local base and stores the status in an input image register
OFF
OFF
X0
X1
Immediate instruction does not use the input image register, but instead reads the status from the module immediately.
I/O Point X0 Changes
ON
OFF
X0
X1

Boolean Instructions

Store (STR)

DS	Used
HPP	Used

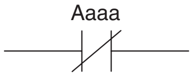
The Store instruction begins a new rung or an additional branch in a rung with a normally open contact. Status of the contact will be the same state as the associated image register point or memory location.



Store Not (STRN)

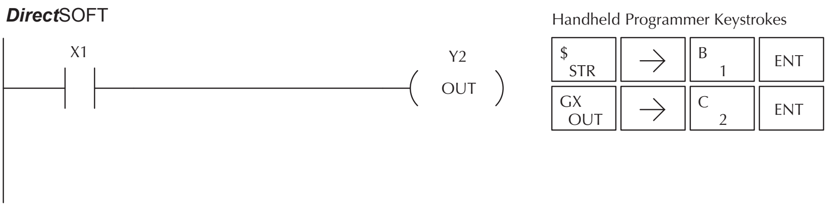
DS	Used
HPP	Used

The Store Not instruction begins a new rung or an additional branch in a rung with a normally closed contact. Status of the contact will be opposite the state of the associated image register point or memory location.



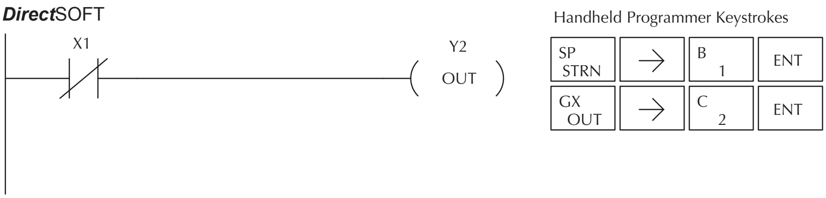
Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter C	CT	0–177
Special Relay	SP	0–777

In the following Store example, when input X1 is on, output Y2 will energize.



$ STR	→	B 1	ENT
GX OUT	→	C 2	ENT

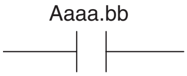
In the following Store Not example, when input X1 is off output Y2 will energize.



SP STRN	→	B 1	ENT
GX OUT	→	C 2	ENT

Store Bit-of-Word (STRB)

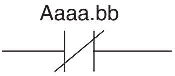
DS	Used
HPP	Used



The Store Bit-of-Word instruction begins a new rung or an additional branch in a rung with a normally open contact. Status of the contact will be the same state as the bit referenced in the associated memory location.

Store Not Bit-of-Word (STRNB)

DS	Used
HPP	Used

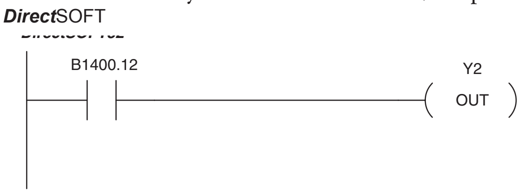


The Store Not instruction begins a new rung or an additional branch in a rung with a normally closed contact. Status of the contact will be opposite the state of the bit referenced in the associated memory location.

Operand Data Type		DL06 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

These instructions look like the STR and STRN instructions only the address is different. Take note how the address is set up in the following Store Bit-of-Word example.

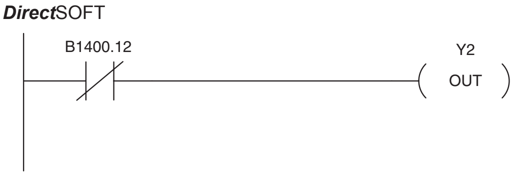
When bit 12 of V-memory location V1400 is on, output Y2 will energize.



Handheld Programmer Keystrokes

STR	SHFT	B	→	V	1	4	0	0
→	K	1	2	ENT				
OUT	→	2	ENT					

In the following Store Not Bit-of-Word example, when bit 12 of V-memory location V1400 is off, output Y2 will energize.



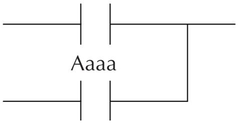
Handheld Programmer Keystrokes

STRN	SHFT	B	→	V	1	4	0	0
→	K	1	2	ENT				
OUT	→	2	ENT					

Or (OR)

DS	Implied
HPP	Used

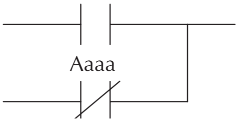
The Or instruction will logically OR a normally open contact in parallel with another contact in a rung. The status of the contact will be the same state as the associated image register point or memory location.



Or Not (ORN)

DS	Implied
HPP	Used

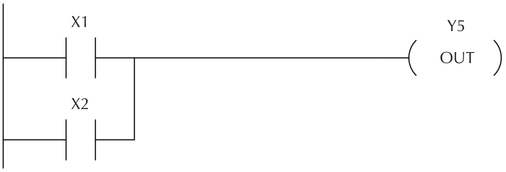
The Or Not instruction will logically OR a normally closed contact in parallel with another contact in a rung. The status of the contact will be opposite the state of the associated image register point or memory location.



Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0-777
Outputs	Y	0-777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0-777

In the following Or example, when input X1 or X2 is on, output Y5 will energize.

*Direct*SOFT

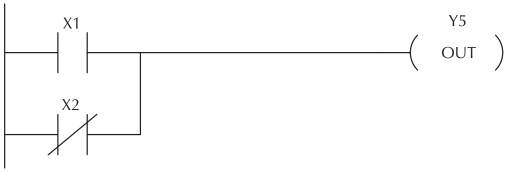


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
Q OR	→	C 2	ENT
GX OUT	→	F 5	ENT

In the following Or Not example, when input X1 is on or X2 is off, output Y5 will energize.

*Direct*SOFT



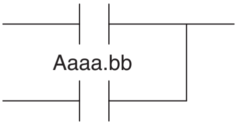
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
R ORN	→	C 2	ENT
GX OUT	→	F 5	ENT

Or Bit-of-Word (OR)

DS	Implied
HPP	Used

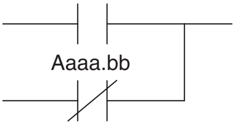
The Or Bit-of-Word instruction will logically OR a normally open contact in parallel with another contact in a rung. Status of the contact will be the same state as the bit referenced in the associated memory location.



Or Not Bit-of-Word (ORN)

DS	Implied
HPP	Used

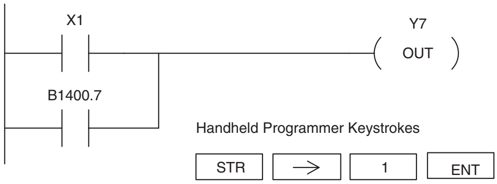
The Or Not Bit-of-Word instruction will logically OR a normally closed contact in parallel with another contact in a rung. Status of the contact will be opposite the state of the bit referenced in the associated memory location.



Operand Data Type		DL06 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

In the following Or Bit-of-Word example, when input X1 or bit 7 of V1400 is on, output Y5 will energize.

*Direct*SOFT

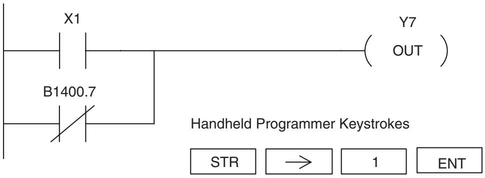


Handheld Programmer Keystrokes

STR	→	1	ENT					
OR	SHFT	B	→	V	1	4	0	0
→	K	7	ENT					
OUT	→	7	ENT					

In the following Or Bit-of-Word example, when input X1 is on or bit 7 of V1400 is off, output Y7 will energize.

*Direct*SOFT



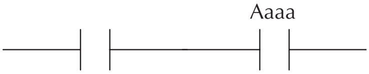
Handheld Programmer Keystrokes

STR	→	1	ENT					
ORN	SHFT	B	→	V	1	4	0	0
→	K	7	ENT					
OUT	→	7	ENT					

AND (AND)

DS	Implied
HPP	Used

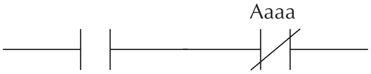
The AND instruction logically ands a normally open contact in series with another contact in a rung. The status of the contact will be the same state as the associated image register point or memory location.



AND NOT (ANDN)

DS	Implied
HPP	Used

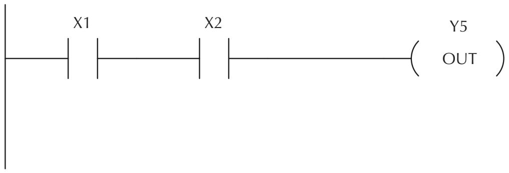
The AND NOT instruction logically ands a normally closed contact in series with another contact in a rung. The status of the contact will be opposite the state of the associated image register point or memory location



Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0–777

In the following And example, when input X1 and X2 are on output Y5 will energize.

*Direct*SOFT

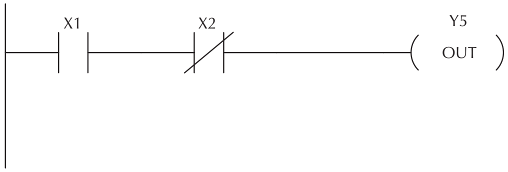


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
V AND	→	C 2	ENT
GX OUT	→	F 5	ENT

In the following And Not example, when input X1 is on and X2 is off output Y5 will energize.

*Direct*SOFT



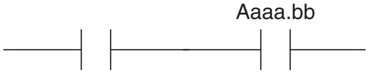
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
W ANDN	→	C 2	ENT
GX OUT	→	F 5	ENT

AND Bit-of-Word (AND)

DS	Implied
HPP	Used

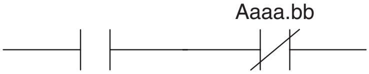
The And Bit-of-Word instruction logically ands a normally open contact in series with another contact in a rung. The status of the contact will be the same state as the bit referenced in the associated memory location.



AND Not Bit-of-Word (ANDN)

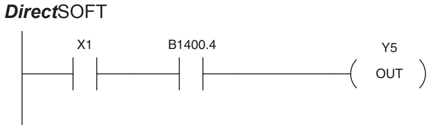
DS	Implied
HPP	Used

The And Not Bit-of-Word instruction logically ands a normally closed contact in series with another contact in a rung. The status of the contact will be opposite the state of the bit referenced in the associated memory location.



Operand Data Type		DL06 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

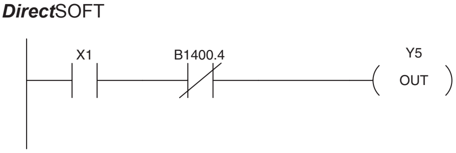
In the following And Bit-of-Word example, when input X1 and bit 4 of V1400 is on output Y5 will energize.



Handheld Programmer Keystrokes

STR → 1 ENT
AND SHFT B → V 1 4 0 0
→ K 4 ENT
OUT → 5 ENT

In the following And Not Bit-of-Word example, when input X1 is on and bit 4 of V1400 is off output Y5 will energize.



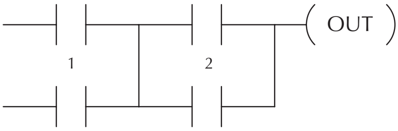
Handheld Programmer Keystrokes

STR → 1 ENT
ANDN SHFT B → V 1 4 0 0
→ K 4 ENT
OUT → 5 ENT

And Store (ANDSTR)

DS	Implied
HPP	Used

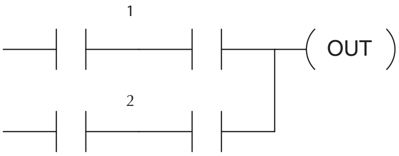
The And Store instruction logically ands two branches of a rung in series. Both branches must begin with the Store instruction.



OR Store (ORSTR)

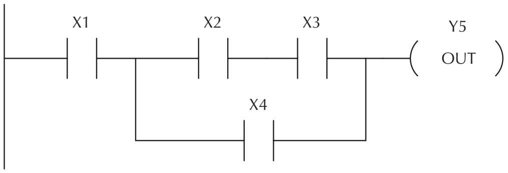
DS	Implied
HPP	Used

The Or Store instruction logically ORs two branches of a rung in parallel. Both branches must begin with the Store instruction.



In the following And Store example, the branch consisting of contacts X2, X3, and X4 have been anded with the branch consisting of contact X1.

*Direct*SOFT

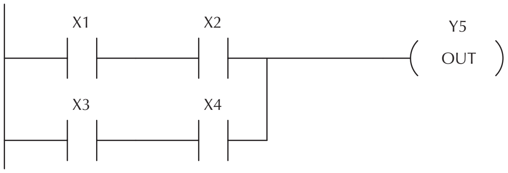


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
$ STR	→	C 2	ENT
V AND	→	D 3	ENT
Q OR	→	E 4	ENT
L ANDST	ENT		
GX OUT	→	F 5	ENT

In the following Or Store example, the branch consisting of X1 and X2 have been ored with the branch consisting of X3 and X4.

*Direct*SOFT

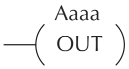


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
V AND	→	C 2	ENT
$ STR	→	D 3	ENT
V AND	→	E 4	ENT
M ORST	ENT		
GX OUT	→	F 5	ENT

Out (OUT)

DS	Used
HPP	Used



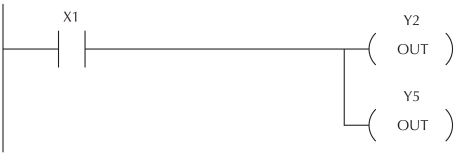
The Out instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) state to the specified image register point or memory location.

Multiple Out instructions referencing the same discrete location should not be used since only the last Out instruction in the program will control the physical output point. Instead, use the next instruction, the Or Out.

Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777

In the following Out example, when input X1 is on, output Y2 and Y5 will energize.

*Direct*SOFT

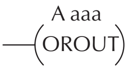


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
GX OUT	→	C 2	ENT
GX OUT	→	F 5	ENT

Or Out (OROUT)

DS	Usied
HPP	Used

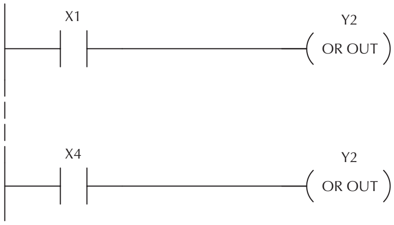


The Or Out instruction allows more than one rung of discrete logic to control a single output. Multiple Or Out instructions referencing the same output coil may be used, since *all* contacts controlling the output are logically OR'd together. If the status of *any* rung is on, the output will also be on.

Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0-777
Control Relays	C	0–1777

In the following example, when X1 or X4 is on, Y2 will energize.

*Direct*SOFT



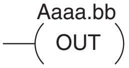
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT				
O INST#	D 3	F 5	ENT	ENT	→	C 2	ENT
$ STR	→	E 4	ENT				
O INST#	D 3	F 5	ENT	ENT	→	C 2	ENT

Out Bit-of-Word (OUT)

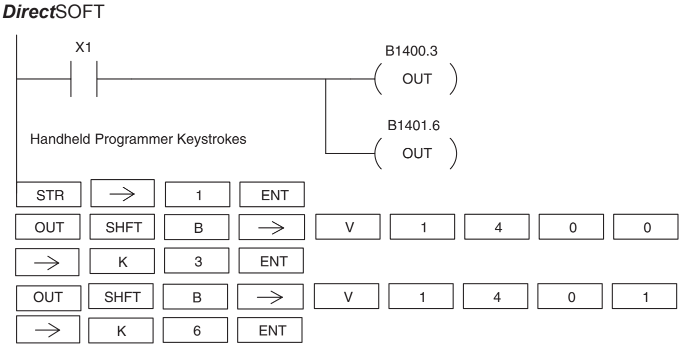
DS	Used
HPP	Used

The Out Bit-of-Word instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) state to the specified bit in the referenced memory location. Multiple Out Bit-of-Word instructions referencing the same bit of the same word generally should not be used since only the last Out instruction in the program will control the status of the bit.



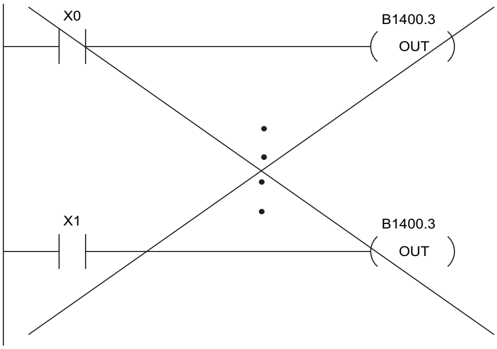
Operand Data Type		DL06 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

***NOTE:** If the Bit-of-Word is entered as V1400.3 in **DirectSOFT**, it will be converted to B1400.3. Bit-of-Word can also be entered as B1400.3.*



In the following Out Bit-of-Word example, when input X1 is on, bit 3 of V1400 and bit 6 of V1401 will turn on.

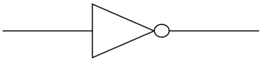
The following Out Bit-of-Word example contains two Out Bit-of-Word instructions using the same bit in the same memory word. The final state bit 3 of V1400 is ultimately controlled by the last rung of logic referencing it. X1 will override the logic state controlled by X0. To avoid this situation, multiple outputs using the same location must not be used in programming.



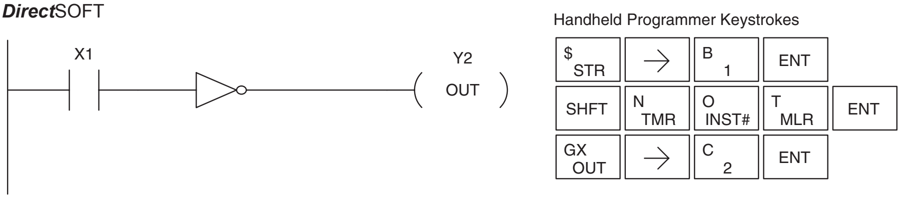
Not (NOT)

DS	Used
HPP	Used

The Not instruction inverts the status of the rung at the point of the instruction.



In the following example, when X1 is off, Y2 will energize. This is because the Not instruction inverts the status of the rung at the Not instruction.

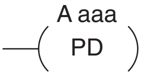


***NOTE**: **Direct**SOFT Release 1.1i and later supports the use of the NOT instruction. The above example rung is merely intended to show the visual representation of the NOT instruction. The NOT instruction can only be selected in **DirectSOFT** from the Instruction Browser. The rung cannot be created or displayed in **Direct**SOFT versions earlier than 1.1i.*

Positive Differential (PD)

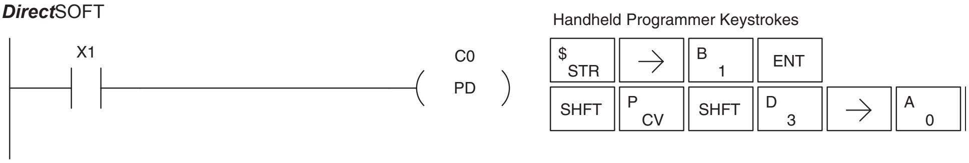
DS	Used
HPP	Used

The Positive Differential instruction is typically known as a one shot. When the input logic produces an off to on transition, the output will energize for one CPU scan.



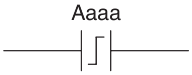
Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777

In the following example, every time X1 makes an Off-to-On transition, C0 will energize for one scan.



Store Positive Differential (STRPD)

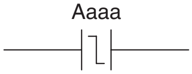
DS	Used
HPP	Used



The Store Positive Differential instruction begins a new rung or an additional branch in a rung with a contact. The contact closes for one CPU scan when the state of the associated image register point makes an Off-to-On transition. Thereafter, the contact remains open until the next Off-to-On transition (the symbol inside the contact represents the transition). This function is sometimes called a “one-shot”. This contact will also close on a program-to-run transition if it is within a retentative range.

Store Negative Differential (STRND)

DS	Used
HPP	Used

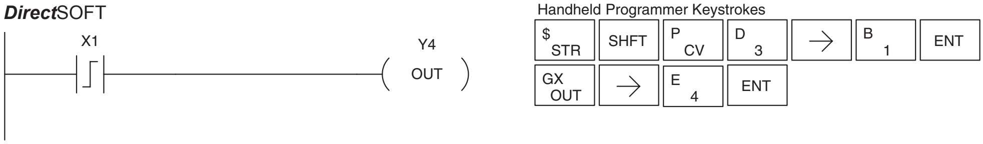


The Store Negative Differential instruction begins a new rung or an additional branch in a rung with a contact. The contact closes for one CPU scan when the state of the associated image register point makes an On-to-Off transition. Thereafter, the contact remains open until the next On-to-Off transition (the symbol inside the contact represents the transition).

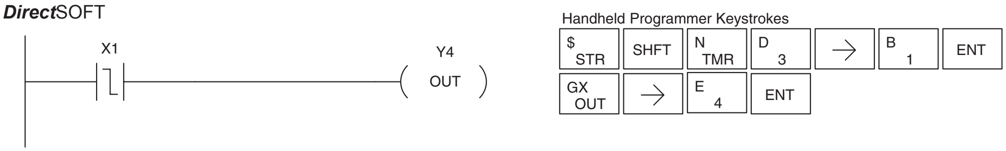
***NOTE:** When using **DirectSOFT**, these instructions can only be entered from the Instruction Browser.*

Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177

In the following example, each time X1 makes an Off-to-On transition, Y4 will energize for one scan.



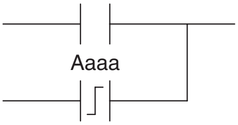
In the following example, each time X1 makes an On-to-Off transition, Y4 will energize for one scan.



Or Positive Differential (ORPD)

DS	Implied
HPP	Used

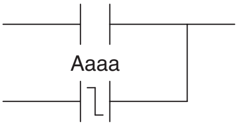
The Or Positive Differential instruction logically ors a contact in parallel with another contact in a rung. The status of the contact will be open until the associated image register point makes an Off-to-On transition, closing it for one CPU scan. Thereafter, it remains open until another Off-to-On transition.



Or Negative Differential (ORND)

DS	Implied
HPP	Used

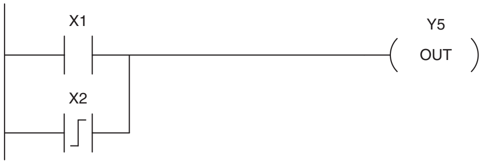
The Or Negative Differential instruction logically ors a contact in parallel with another contact in a rung. The status of the contact will be open until the associated image register point makes an On-to-Off transition, closing it for one CPU scan. Thereafter, it remains open until another On-to-Off transition.



Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177

In the following example, Y 5 will energize whenever X1 is on, or for one CPU scan when X2 transitions from Off to On.

***Direct*SOFT**



Handheld Programmer Keystrokes

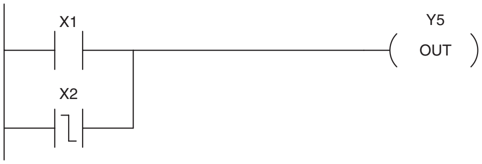
$ STR | → | B 1 | ENT

Q OR | SHFT | P CV | D 3 | → | C 2 | ENT

GX OUT | → | F 5 | ENT

In the following example, Y 5 will energize whenever X1 is on, or for one CPU scan when X2 transitions from On to Off.

***Direct*SOFT**



Handheld Programmer Keystrokes

$ STR | → | B 1 | ENT

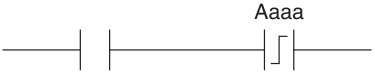
Q OR | SHFT | N TMR | D 3 | → | C 2 | ENT

GX OUT | → | F 5 | ENT

And Positive Differential (ANDPD)

DS	Implied
HPP	Used

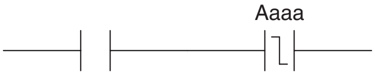
The And Positive Differential instruction logically ands a normally open contact in series with another contact in a rung. The status of the contact will be open until the associated image register point makes an Off-to-On transition, closing it for one CPU scan. Thereafter, it remains open until another Off-to-On transition.



And Negative Differential (ANDND)

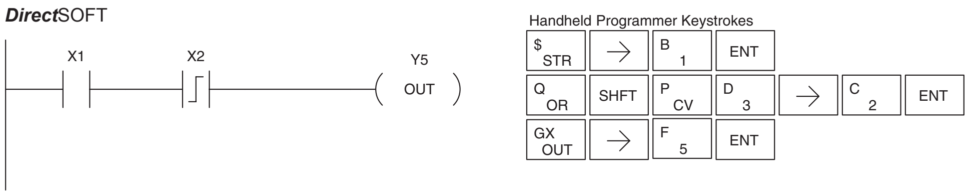
DS	Implied
HPP	Used

The And Negative Differential instruction logically ands a normally open contact in series with another contact 5-22in a rung.The status of the contact will be open until the associated image register point makes an On-to-Off transition, closing it for one CPU scan. Thereafter, it remains open until another On-to-Off transition.

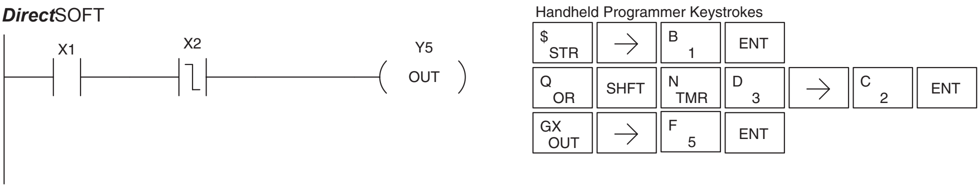


Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177

In the following example, Y5 will energize for one CPU scan whenever X1 is on and X2 transitions from Off to On.



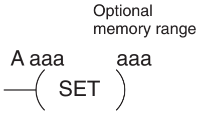
In the following example, Y5 will energize for one CPU scan whenever X1 is on and X2 transitions from On to Off.



Set (SET)

DS	Used
HPP	Used

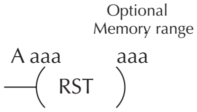
The Set instruction sets or turns on an image register point/memory location or a consecutive range of image register points/memory locations. Once the point/location is set it will remain on until it is reset using the Reset instruction. It is not necessary for the input controlling the Set instruction to remain on.



Reset (RST)

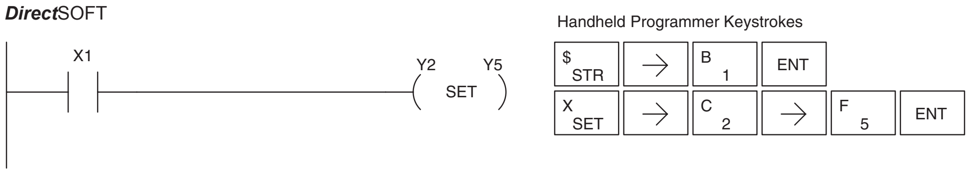
DS	Used
HPP	Used

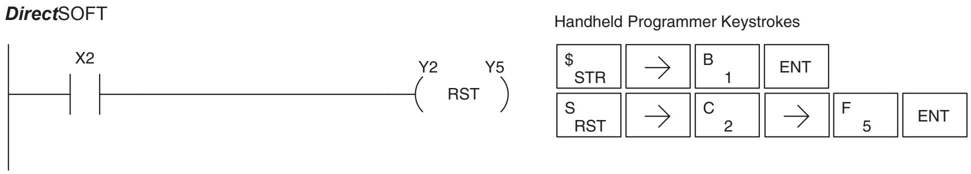
The Reset instruction resets or turns off an image register point/memory location or a range of image registers points/memory locations. Once the point/location is reset, it is not necessary for the input to remain on.



Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177

In the following example when X1 is on, Y2 through Y5 will energize.

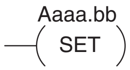




Set Bit-of-Word (SET)

DS	Used
HPP	Used

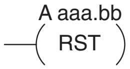
The Set Bit-of-Word instruction sets or turns on a bit in a V-memory location. Once the bit is set, it will remain on until it is reset using the Reset Bit-of-Word instruction. It is not necessary for the input controlling the Set Bit-of-Word instruction to remain on.



Reset Bit-of-Word (RST)

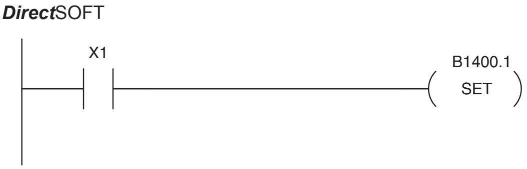
DS	Used
HPP	Used

The Reset Bit-of-Word instruction resets or turns off a bit in a V-memory location. Once the bit is reset. it is not necessary for the input to remain on.

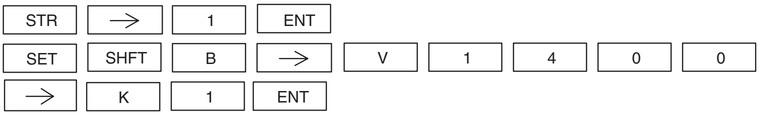


Operand Data Type		DL06 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

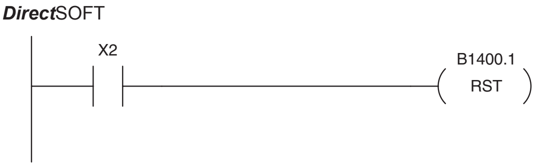
In the following example. when X1 turns on, bit 1 in V1400 is set to the on state.



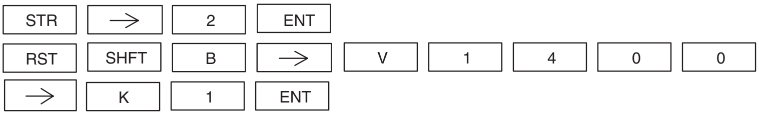
Handheld Programmer Keystrokes


STR → 1 ENT
SET SHFT B → V 1 4 0 0
→ K 1 ENT

In the following example, when X2 turns on, bit 1 in V1400 is reset to the off state.



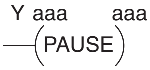
Handheld Programmer Keystrokes


STR → 2 ENT
RST SHFT B → V 1 4 0 0
→ K 1 ENT

Pause (PAUSE)

DS	Used
HPP	Used

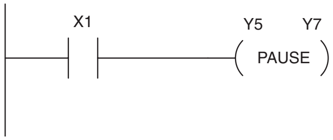
The Pause instruction disables the output update on a range of outputs. The ladder program will continue to run and update the image register. However, the outputs in the range specified in the Pause instruction will be turned off at the output points.



Operand Data Type		DL06 Range
	A	aaa
Outputs	Y	0–777

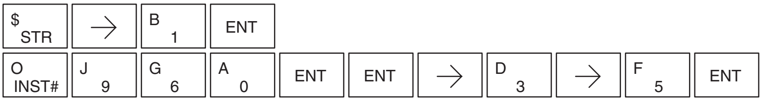
In the following example, when X1 is ON, Y5–Y7 will be turned OFF. The execution of the *Direct*SOFT ladder program will not be affected.

*Direct*SOFT32



Since the D2–HPP Handheld Programmer does not have a specific Pause key, you can use the corresponding instruction number for entry (#960), or type each letter of the command.

Handheld Programmer Keystrokes



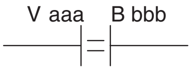
In some cases, you may want certain output points in the specified pause range to operate normally. In that case, use Aux 58 to over-ride the Pause instruction.

Comparative Boolean

Store If Equal (STRE)

DS	Implied
HPP	Used

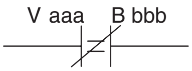
The Store If Equal instruction begins a new rung or additional branch in a rung with a normally open comparative contact. The contact will be on when Vaaa equals Bbbb .



Store If Not Equal (STRNE)

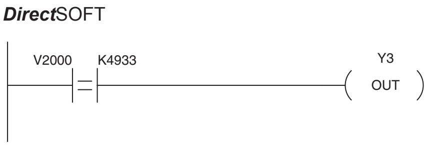
DS	Implied
HPP	Used

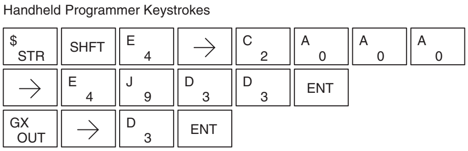
The Store If Not Equal instruction begins a new rung or additional branch in a rung with a normally closed comparative contact. The contact will be on when Vaaa does not equal Bbbb.



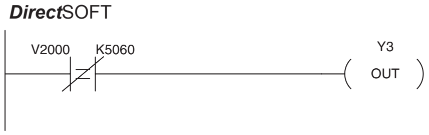
Operand Data Type		DL06 Range	
	B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map
Constant	K	—	0–9999

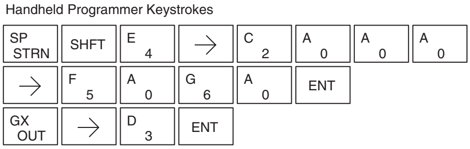
In the following example, when the BCD value in V-memory location V2000 = 4933, Y3 will energize.





In the following example, when the value in V-memory location V2000 ≠ 5060, Y3 will energize.

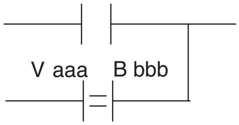




Or If Equal (ORE)

DS	Implied
HPP	Used

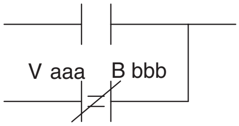
The Or If Equal instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when Vaaa = Bbbb.



Or If Not Equal (ORNE)

DS	Implied
HPP	Used

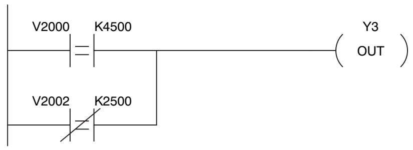
The Or If Not Equal instruction connects a normally closed comparative contact in parallel with another contact. The contact will be on when Vaaa does not equal Bbbb.



Operand Data Type		DL06 Range	
	B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map
Constant	K	—	0–9999

In the following example, when the BCD value in V-memory location V2000 = 4500 or V2002 ≠ 2500, Y3 will energize.

*Direct*SOFT

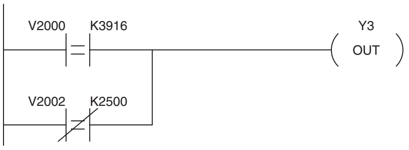


Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
E 4	F 5	A 0	A 0	ENT				
Q OR	SHFT	E 4	→	C 2	A 0	A 0	C 2	→
C 2	D 3	E 4	F 5	ENT				
GX OUT	→	D 3	ENT					

In the following example, when the BCD value in V-memory location V2000 = 3916 or V2002 ≠ 2500, Y3 will energize.

*Direct*SOFT



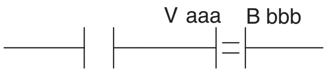
Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
D 3	J 9	B 1	G 6	ENT				
R ORN	SHFT	E 4	→	C 2	A 0	A 0	C 2	→
C 2	F 5	A 0	A 0	ENT				
GX OUT	→	D 3	ENT					

And If Equal (ANDE)

DS	Implied
HPP	Used

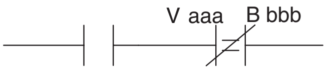
The And If Equal instruction connects a normally open comparative contact in series with another contact. The contact will be on when Vaaa = Bbbb.



And If Not Equal (ANDNE)

DS	Implied
HPP	Used

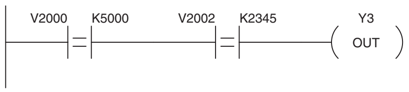
The And If Not Equal instruction connects a normally closed comparative contact in series with another contact. The contact will be on when Vaaa does not equal Bbbb.



Operand Data Type		DL06 Range	
	B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map
Constant	K	—	0–9999

In the following example, when the BCD value in V-memory location V2000 = 5000 and V2002 = 2345, Y3 will energize.

*Direct*SOFT

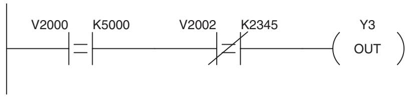


Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
F 5	A 0	A 0	A 0	ENT				
V AND	SHFT	E 4	→	C 2	A 0	A 0	C 2	→
C 2	D 3	E 4	F 5	ENT				
GX OUT	→	D 3	ENT					

In the following example, when the BCD value in V-memory location V2000 = 5000 and V2002 ≠ 2345, Y3 will energize.

*Direct*SOFT



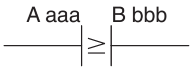
Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
F 5	A 0	A 0	A 0	ENT				
V AND	SHFT	E 4	→	C 2	A 0	A 0	C 2	→
C 2	D 3	E 4	F 5	ENT				
GX OUT	→	D 3	ENT					

Store (STR)

DS	Implied
HPP	Used

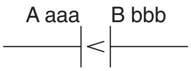
The Comparative Store instruction begins a new rung or additional branch in a rung with a normally open comparative contact. The contact will be on when Aaaa is equal to or greater than Bbbb.



Store Not (STRN)

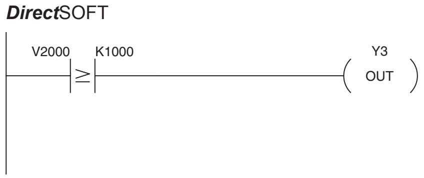
DS	Implied
HPP	Used

The Comparative Store Not instruction begins a new rung or additional branch in a rung with a normally closed comparative contact. The contact will be on when Aaaa < Bbbb.



Operand Data Type		DL06 Range	
	A/B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	p	See memory map	See memory map
Constant	K	—	0–9999
Timer	TA	0–377	
Counter	CTA	0–177	

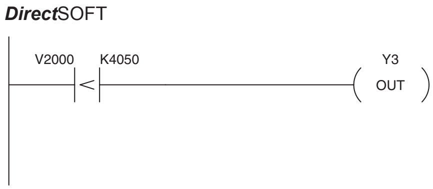
In the following example, when the BCD value in V-memory location V2000 ≥ 1000, Y3 will energize.



Handheld Programmer Keystrokes

$ STR	→	SHFT	V AND	C 2	A 0	A 0	A 0
→	B 1	A 0	A 0	A 0	ENT		
GX OUT	→	D 3	ENT				

In the following example, when the value in V-memory location V2000 < 4050, Y3 will energize.



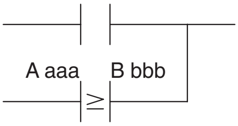
Handheld Programmer Keystrokes

SP STRN	→	SHFT	V AND	C 2	A 0	A 0	A 0
→	E 4	A 0	F 5	A 0	ENT		
GX OUT	→	D 3	ENT				

Or (OR)

DS	Implied
HPP	Used

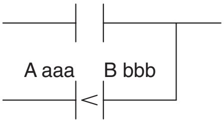
The Comparative Or instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when Aaaa is equal to or greater than Bbbb.



Or Not (ORN)

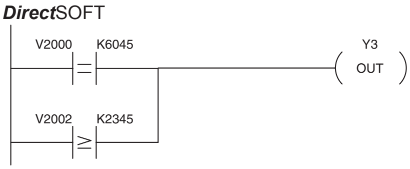
DS	Implied
HPP	Used

The Comparative Or Not instruction connects a normally closed comparative contact in parallel with another contact. The contact will be on when Aaaa < Bbbb.



Operand Data Type		DL06 Range	
	A/B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	p	See memory map	See memory map
Constant	K	—	0–9999
Timer	TA	0–377	
Counter	CTA	0–177	

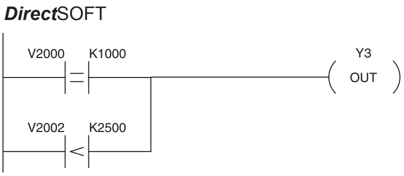
In the following example, when the BCD value in V-memory location V2000 = 6045 or V2002 ≥ 2345, Y3 will energize.



Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
G 6	A 0	E 4	F 5	ENT				
Q OR	→	SHFT	V AND	C 2	A 0	A 0	C 2	→
C 2	D 3	E 4	F 5	ENT				
GX OUT	→	D 3	ENT					

In the following example when the BCD value in V-memory location V2000 = 1000 or V2002 < 2500, Y3 will energize.



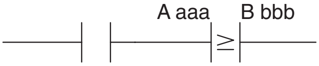
Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
B 1	A 0	A 0	A 0	ENT				
R ORN	→	SHFT	V AND	C 2	A 0	A 0	C 2	→
C 2	F 5	A 0	A 0	ENT				
GX OUT	→	D 3	ENT					

And (AND)

DS	Implied
HPP	Used

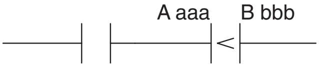
The Comparative And instruction connects a normally open comparative contact in series with another contact. The contact will be on when Aaaa is equal to or greater than Bbbb.



And Not (ANDN)

DS	Implied
HPP	Used

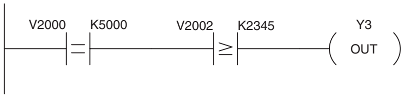
The Comparative And Not instruction connects a normally closed comparative contact in series with another contact. The contact will be on when Aaaa < Bbbb.



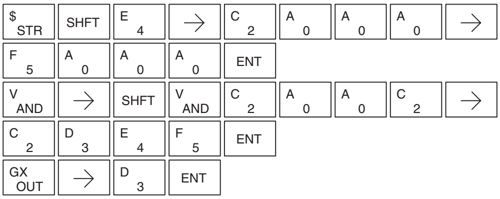
Operand Data Type	A/B	DL06 Range aaa	DL06 Range bbb
V-memory	V	See memory map	See memory map
Pointer	p	See memory map	See memory map
Constant	K	—	0–9999
Timer	TA	0–377	
Counter	CTA	0–177	

In the following example, when the value in BCD V-memory location V2000 = 5000, and V2002 ≥ 2345, Y3 will energize.

*Direct*SOFT

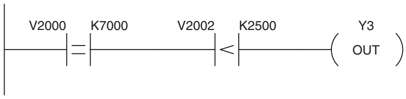


Handheld Programmer Keystrokes

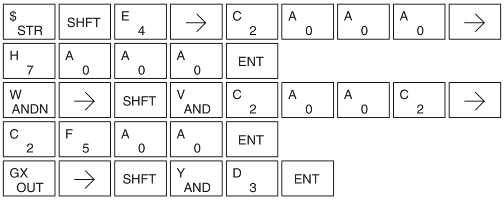

$ STR | SHFT | E 4 | → | C 2 | A 0 | A 0 | A 0 | →
F 5 | A 0 | A 0 | A 0 | ENT
V AND | → | SHFT | V AND | C 2 | A 0 | A 0 | C 2 | →
C 2 | D 3 | E 4 | F 5 | ENT
GX OUT | → | D 3 | ENT

In the following example, when the value in V-memory location V2000 = 7000 and V2002 < 2500, Y3 will energize.

*Direct*SOFT



Handheld Programmer Keystrokes

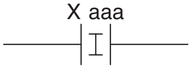

$ STR | SHFT | E 4 | → | C 2 | A 0 | A 0 | A 0 | →
H 7 | A 0 | A 0 | A 0 | ENT
W ANDN | → | SHFT | V AND | C 2 | A 0 | A 0 | C 2 | →
C 2 | F 5 | A 0 | A 0 | ENT
GX OUT | → | SHFT | Y AND | D 3 | ENT

Immediate Instructions

Store Immediate (STRI)

DS	Implied
HPP	Used

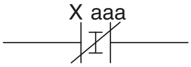
The Store Immediate instruction begins a new rung or additional branch in a rung. The status of the contact will be the same as the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



Store Not Immediate (STRNI)

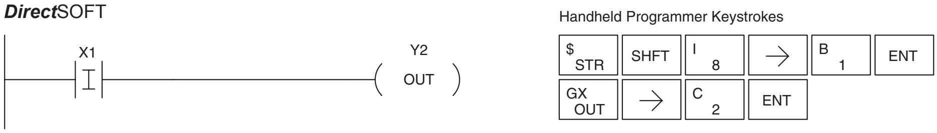
DS	Implied
HPP	Used

The Store Not Immediate instruction begins a new rung or additional branch in a rung. The status of the contact will be opposite the status of the associated input point *at the time the instruction is executed*. The image register is not updated.

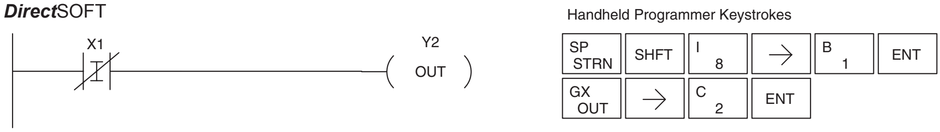


Operand Data Type		DL06 Range
		aaa
Inputs	X	0–777

In the following example, when X1 is on, Y2 will energize.



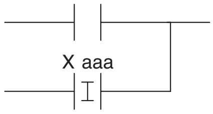
In the following example, when X1 is off, Y2 will energize.



Or Immediate (ORI)

DS	Implied
HPP	Used

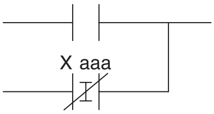
The Or Immediate connects two contacts in parallel. The status of the contact will be the same as the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



Or Not Immediate (ORNI)

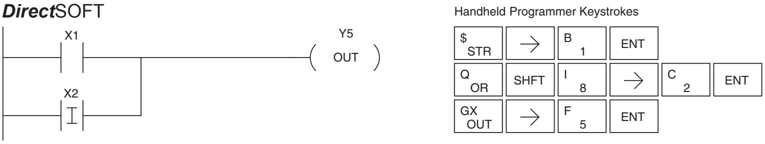
DS	Implied
HPP	Used

The Or Not Immediate connects two contacts in parallel. The status of the contact will be opposite the status of the associated input point *at the time the instruction is executed*. The image register is not updated.

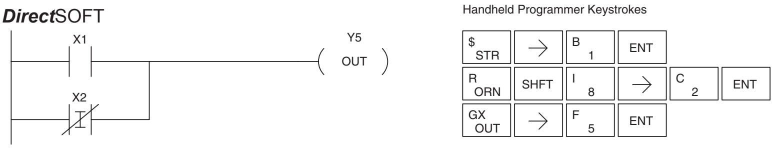


Operand Data Type		DL06 Range
		aaa
Inputs	X	0–777

In the following example, when X1 or X2 is on, Y5 will energize.



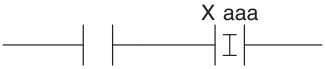
In the following example, when X1 is on or X2 is off, Y5 will energize.



And Immediate (ANDI)

DS	Implied
HPP	Used

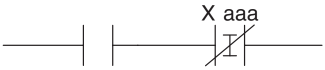
The And Immediate instruction connects two contacts in series. The status of the contact will be the same as the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



And Not Immediate (ANDNI)

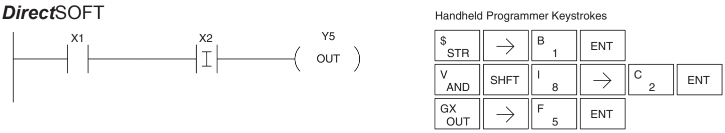
DS	Implied
HPP	Used

The And Not Immediate instruction connects two contacts in series. The status of the contact will be opposite the status of the associated input point *at the time the instruction is executed*. The image register is not updated.

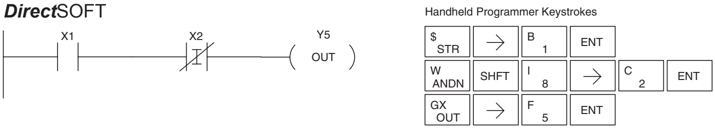


Operand Data Type		DL06 Range
		aaa
Inputs	X	0–777

In the following example, when X1 and X2 are on, Y5 will energize.



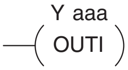
In the following example, when X1 is on and X2 is off, Y5 will energize.



Out Immediate (OUTI)

DS	Used
HPP	Used

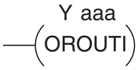
The Out Immediate instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) status to the specified module output point and the image register *at the time the instruction is executed*. If multiple Out Immediate instructions referencing the same discrete point are used, it is possible for the module output status to change multiple times in a CPU scan. See Or Out Immediate.



Or Out Immediate (OROUTI)

DS	Used
HPP	Used

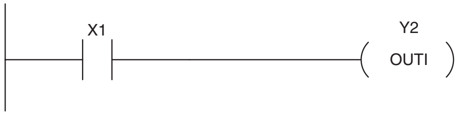
The Or Out Immediate instruction has been designed to use more than 1 rung of discrete logic to control a single output. Multiple Or Out Immediate instructions referencing the same output coil may be used, since all contacts controlling the output are ored together. If the status of any rung is on *at the time the instruction is executed*, the output will also be on.



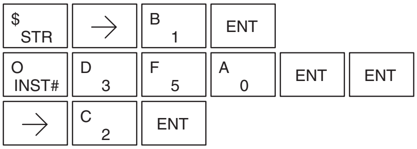
Operand Data Type		DL06 Range
		aaa
Outputs	Y	0–777

In the following example, when X1 is on, output point Y2 on the output module will turn on. For instruction entry on the Handheld Programmer, you can use the instruction number (#350) as shown, or type each letter of the command.

*Direct*SOFT

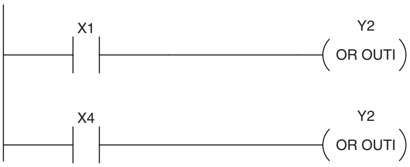


Handheld Programmer Keystrokes

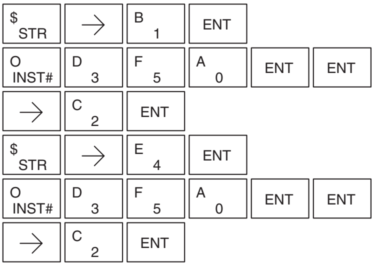


In the following example, when X1 or X4 is on, Y2 will energize.

*Direct*SOFT



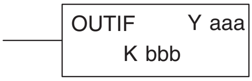
Handheld Programmer Keystrokes



Out Immediate Formatted (OUTIF)

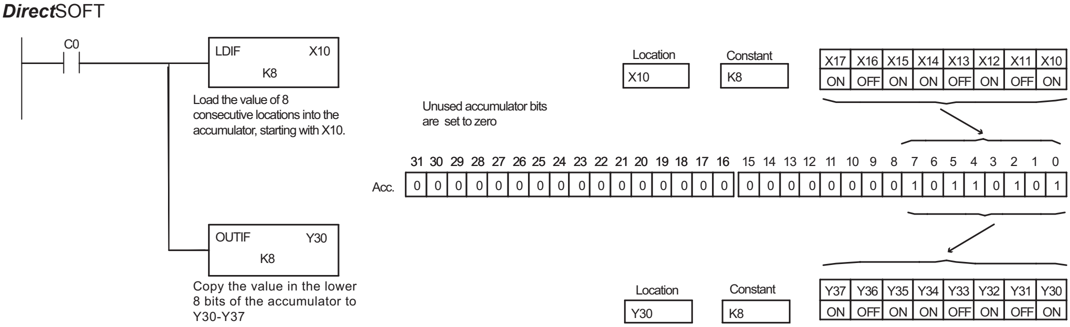
DS	Used
HPP	Used

The Out Immediate Formatted instruction outputs a 1–32 bit binary value from the accumulator to specified output points *at the time the instruction is executed.* Accumulator bits that are not used by the instruction are set to zero.



Operand Data Type		DL06 Range
		aaa
Outputs	Y	0-777
Constant	K	1-32

In the following example, when C0 is on,the binary pattern for X10 –X17 is loaded into the accumulator using the Load Immediate Formatted instruction. The binary pattern in the accumulator is written to Y30–Y37 using the Out Immediate Formatted instruction. This technique is useful to quickly copy an input pattern to outputs (without waiting for the CPU scan).


*Direct*SOFT
C0
LDIF X10
K8
Load the value of 8 consecutive locations into the accumulator, starting with X10.
OUTIF Y30
K8
Copy the value in the lower 8 bits of the accumulator to Y30-Y37
Location
X10
Constant
K8
X17 X16 X15 X14 X13 X12 X11 X10
ON OFF ON ON OFF ON OFF ON
Unused accumulator bits are set to zero
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1
Location
Y30
Constant
K8
Y37 Y36 Y35 Y34 Y33 Y32 Y31 Y30
ON OFF ON ON OFF ON OFF ON

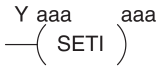
Handheld Programmer Keystrokes

$ STR	→	NEXT	NEXT	NEXT	NEXT	A 0	ENT			
SHFT	L ANDST	D 3	I 8	F 5	→	B 1	A 0	→	I 8	ENT
GX OUT	SHFT	I 8	F 5	→	D 3	A 0	→	I 8	ENT	

Set Immediate (SETI)

DS	Used
HPP	Used

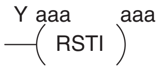
The Set Immediate instruction immediately sets, or turns on an output or a range of outputs in the image register and the corresponding output point(s) *at the time the instruction is executed*. Once the outputs are set, it is not necessary for the input to remain on. The Reset Immediate instruction can be used to reset the outputs.



Reset Immediate (RSTI)

DS	Used
HPP	Used

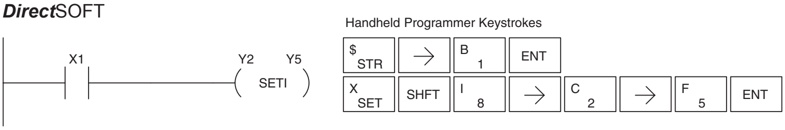
The Reset Immediate instruction immediately resets, or turns off, an output or a range of outputs in the image register and the output point(s) *at the time the instruction is executed.* Once the outputs are reset, it is not necessary for the input to remain on.



Operand Data Type		DL06 Range
		aaa
Ouputs	Y	0–777

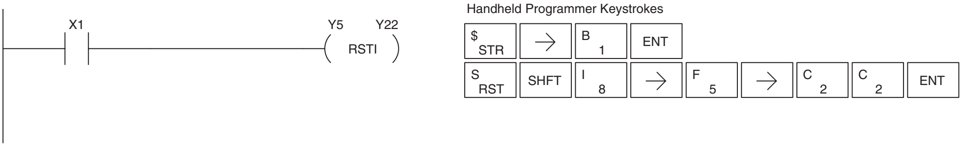
In the following example, when X1 is on, Y2 through Y5 will be set on in the image register and on the corresponding output points.

*Direct*SOFT



In the following example, when X1 is on, Y5 through Y22 will be reset (off) in the image register and on the corresponding output module(s).

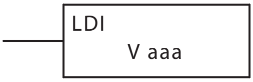
*Direct*SOFT



Load Immediate (LDI)

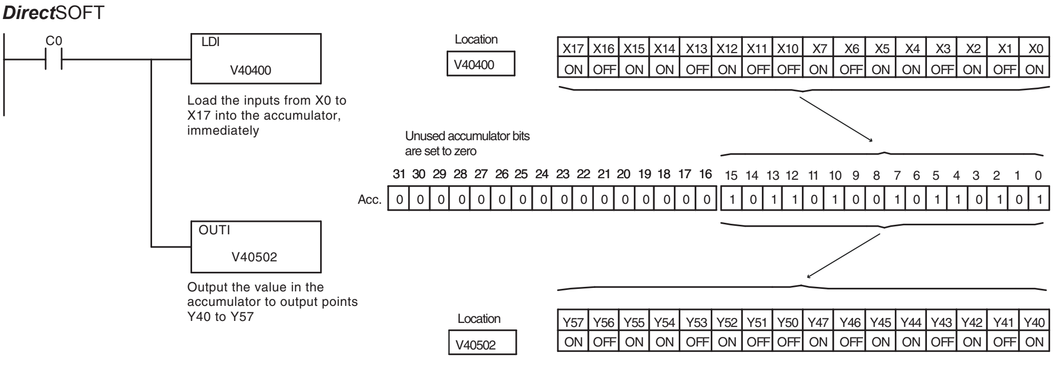
DS	Used
HPP	Used

The Load Immediate instruction loads a 16-bit V-memory value into the accumulator. The valid address range includes all input point addresses on the local base. The value reflects the current status of the input points *at the time the instruction is executed.* This instruction may be used instead of the LDIF instruction, which requires you to specify the number of input points.



Operand Data Type		DL06 Range
		aaa
Inputs	V	40400-40437

In the following example, when C0 is on, the binary pattern of X0–X17 will be loaded into the accumulator using the Load Immediate instruction. The Out Immediate instruction could be used to copy the 16 bits in the accumulator to output points, such as Y40–Y57. This technique is useful to quickly copy an input pattern to output points (without waiting for a full CPU scan to occur).


*Direct*SOFT
C0
LDI
V40400
Load the inputs from X0 to X17 into the accumulator, immediately
OUTI
V40502
Output the value in the accumulator to output points Y40 to Y57
Location
V40400
X17 X16 X15 X14 X13 X12 X11 X10 X7 X6 X5 X4 X3 X2 X1 X0
ON OFF ON ON OFF ON OFF OFF ON OFF ON ON OFF ON OFF ON
Unused accumulator bits are set to zero
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc.
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 1
Location
V40502
Y57 Y56 Y55 Y54 Y53 Y52 Y51 Y50 Y47 Y46 Y45 Y44 Y43 Y42 Y41 Y40
ON OFF ON ON OFF ON OFF OFF ON OFF ON ON OFF ON OFF ON

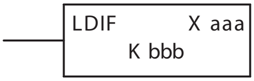
Handheld Programmer Keystrokes

$ STR	→	NEXT	NEXT	NEXT	NEXT	A 0	ENT			
SHFT	L ANDST	D 3	I 8	→	E 4	A 0	E 4	A 0	A 0	ENT
GX OUT	SHFT	I 8	→	NEXT	E 4	A 0	F 5	A 0	C 2	ENT

Load Immediate Formatted (LDIF)

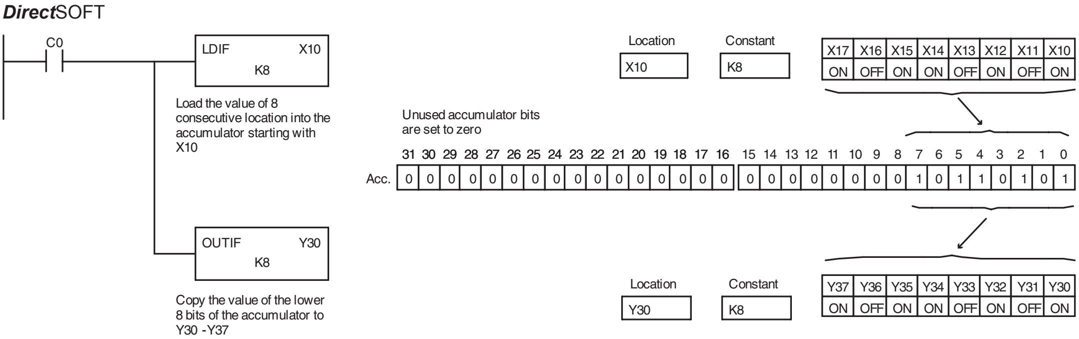
DS	Used
HPP	Used

The Load Immediate Formatted instruction loads a 1–32 bit binary value into the accumulator. The value reflects the current status of the input module(s) *at the time the instruction is executed.* Accumulator bits that are not used by the instruction are set to zero.



Operand Data Type		DL06 Range	
		aaa	bbb
Inputs	X	0-777	- -
Constant	K	- -	1-32

In the following example, when C0 is on, the binary pattern of X10–X17 will be loaded into the accumulator using the Load Immediate Formatted instruction. The Out Immediate Formatted instruction could be used to copy the specified number of bits in the accumulator to the specified outputs on the output module, such as Y30–Y37. This technique is useful to quickly copy an input pattern to outputs (without waiting for the CPU scan).



Load the value of 8 consecutive location into the accumulator starting with X10

Unused accumulator bits are set to zero

Copy the value of the lower 8 bits of the accumulator to Y30 -Y37

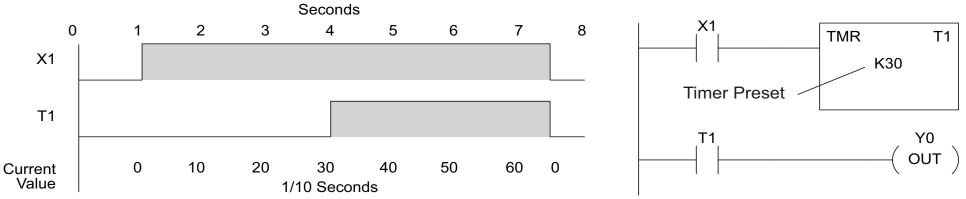
Handheld Programmer Keystrokes

$ STR	→	NEXT	NEXT	NEXT	NEXT	A 0	ENT			
SHFT	L ANDST	D 3	I 8	F 5	→	B 1	A 0	→	I 8	ENT
GX OUT	SHFT	I 8	F 5	→	D 3	A 0	→	I 8	ENT	

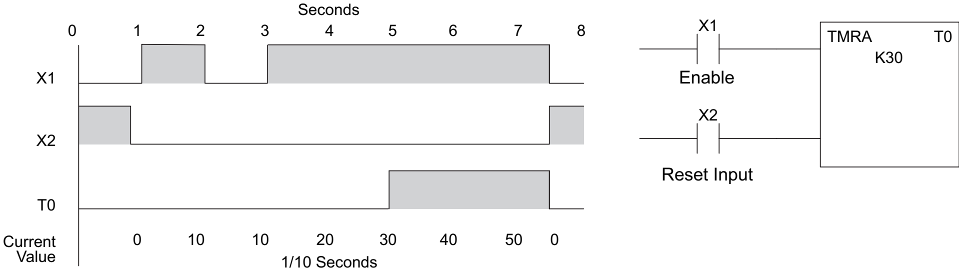
Timer, Counter and Shift Register Instructions

Using Timers

Timers are used to time an event for a desired period. The single input timer will time as long as the input is on. When the input changes from on to off, the timer current value is reset to 0. There is a tenth of a second and a hundredth of a second timer available with a maximum time of 999.9 and 99.99 seconds respectively. There is a discrete bit associated with each timer to indicate that the current value is equal to or greater than the preset value. The timing diagram below shows the relationship between the timer input, associated discrete bit, current value and timer preset.



There are those applications that need an accumulating timer, meaning it has the ability to time, stop, and then resume from where it previously stopped. The accumulating timer works similarly to the regular timer, but two inputs are required. The enable input starts and stops the timer. When the timer stops, the elapsed time is maintained. When the timer starts again, the timing continues from the elapsed time. When the reset input is turned on, the elapsed time is cleared and the timer will start at 0 when it is restarted. There is a tenth of a second and a hundredth of a second timer available with a maximum time of 9999999.9 and 999999.99 seconds respectively. The timing diagram below shows the relationship between the timer input, timer reset, associated discrete bit, current value and timer preset.

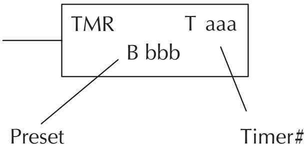


NOTE: *Decimal points are not used in these timers, but the decimal point is implied. The preset and current value for all four timers is in BCD format.*

Timer (TMR) and Timer Fast (TMRF)

DS	Used
HPP	Used

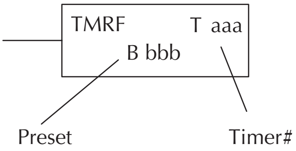
The Timer instruction is a 0.1 second single input timer that times to a maximum of 999.9 seconds. The Timer Fast instruction is a 0.01 second single input timer that times up to a maximum of 99.99 seconds. These timers will be enabled if the input logic is true (on) and will be reset to 0 if the input logic is false (off). Both timers use single word BCD values for the preset and current value. The decimal place is implied.



Instruction Specifications

Timer Reference (Taaa): Specifies the timer number.

Preset Value (Bbbb): Constant value (K) or a V-memory location specified in BCD.



Current Value: Timer current values, in BCD format, are accessed by referencing the associated V or T memory location*. For example, the timer current value for T3 physically resides in V-memory location V3.

Discrete Status Bit: The discrete status bit is referenced by the associated T memory location. Operating as a "timer done bit", it will be on if the current value is equal to or greater than the preset value. For example, the discrete status bit for Timer 2 is T2.

NOTE*: A V-memory preset is required only if the ladder program or an Operator Interface unit must change the preset.*

Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Timers	T	0–777	—
V-memory for preset values	V	—	400-677 1200–7377 7400–7577 10000-17777
Pointers (preset only)	P	—	400-677 1200–7377 7400–7577* 10000-17777
Constants (preset only)	K	—	0–9999
Timer discrete status bits	T/V	0–377 or V41100–41117	
Timer current values	V /T**	0–377	

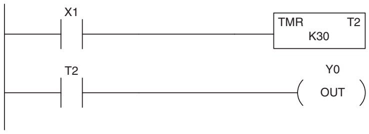
NOTE: **May be non-volatile if MOV instruction is used.*
*** With the HPP, both the Timer discrete status bits and current value are accessed with the same data reference. DirectSOFT uses separate references, such as "T2" for discrete status bit for Timer T2, and "TA2" for the current value of Timer T2.*

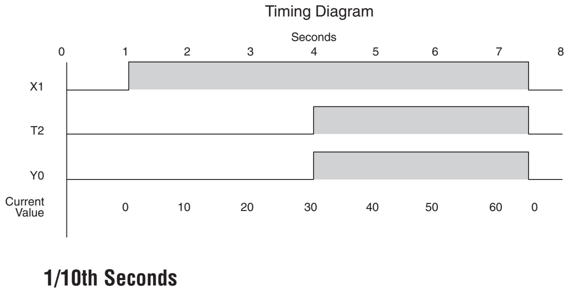
You can perform functions when the timer reaches the specified preset using the discrete status bit. Or, use comparative contacts to perform functions at different time intervals, based on one timer. The examples on the following page show these two methods of programming timers.

Timer Example Using Discrete Status Bits

In the following example, a single input timer is used with a preset of 3 seconds. The timer discrete status bit (T2) will turn on when the timer has timed for 3 seconds. The timer is reset when X1 turns off, turning the discrete status bit off and resetting the timer current value to 0.

*Direct*SOFT





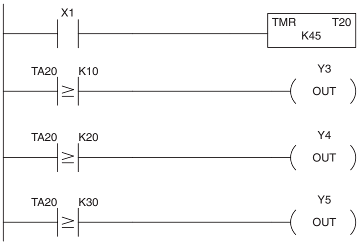
Handheld Programmer Keystrokes

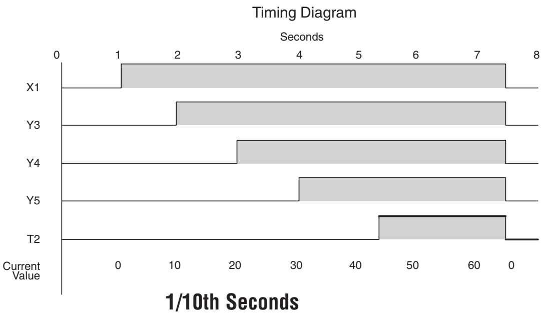
$ STR	→	B 1	ENT			
N TMR	→	C 2	→	D 3	A 0	ENT
$ STR	→	SHFT	T MLR	C 2	ENT	
GX OUT	→	A 0	ENT			

Timer Example Using Comparative Contacts

In the following example, a single input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energize Y3, Y4, and Y5 at one second intervals respectively. When X1 is turned off, the timer will be reset to 0 and the comparative contacts will turn off Y3, Y4, and Y5.

*Direct*SOFT





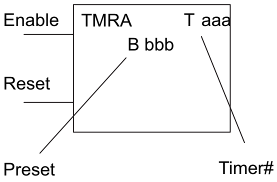
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
N TMR	→	C 2	A 0	→	E 4	F 5	ENT		
$ STR	→	SHFT	T MLR	C 2	A 0	→	B 1	A 0	ENT
GX OUT	→	D 3	ENT						
$ STR	→	SHFT	T MLR	C 2	A 0	→	C 2	A 0	ENT
GX OUT	→	E 4	ENT						
$ STR	→	SHFT	T MLR	C 2	A 0	→	D 3	A 0	ENT
GX OUT	→	F 5	ENT						

Accumulating Timer (TMRA)

DS	Used
HPP	Used

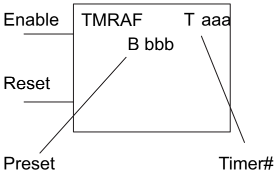
The Accumulating Timer is a 0.1 second two input timer that will time to a maximum of 9999999.9. *The TMRA uses two timer registers in V-memory.*



Accumulating Fast Timer (TMRAF)

DS	Used
HPP	Used

The Accumulating Fast Timer is a 0.01 second two-input timer that will time to a maximum of 99999.99. *The TMRA uses two timer registers in V-memory.*



Each timer uses two timer registers in V-memory. The preset and current values are in double word BCD format, and the decimal point is implied. These timers have two inputs, an enable and a reset. The timer starts timing when the enable is on and stops when the enable is off (without resetting the count). The reset will reset the timer when on and allow the timer to time when off.

Timer Reference (Taaa): Specifies the timer number.

Preset Value (Bbbb): Constant value (K) or V-memory.

Current Value: Timer current values are accessed by referencing the associated V or T memory location*. For example, the timer current value for T3 resides in V-memory, V3.

Discrete Status Bit: The discrete status bit is accessed by referencing the associated T memory location. Operating as a "timer done bit," it will be on if the current value is equal to or greater than the preset value. For example, the discrete status bit for timer 2 would be T2.

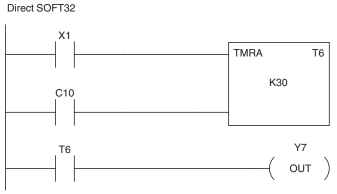
NOTE: *The accumulating timer uses two consecutive V-memory locations for the 8-digit value, therefore two consecutive timer locations. For example, if TMRA T1 is used, the next available timer number is T3.*
NOTE: *A V-memory preset is required if the ladder program or an OIP must be used to change the preset.*

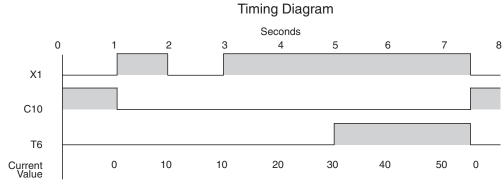
Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Timers	T	0–777	—
V-memory for preset values	V	—	400-677 1200–7377 7400–7577 10000-17777
Pointers (preset only)	P	—	400-677 1200–7377 7400–7577* 10000-17777
Constants (preset only)	K	—	0–99999999
Timer discrete status bits	T/V	0–377 or V41100–41117	
Timer current values	V /T**	0–377	

NOTE: **May be non-volatile if MOV instruction is used.** With the HPP, both the Timer discrete status bits and current value are accessed with the same data reference. DirectSOFT uses separate references, such as "T2" for discrete status bit for Timer T2, and "TA2" for the current value of Timer T2.*

Accumulating Timer Example using Discrete Status Bits

In the following example, a two input timer (accumulating timer) is used with a preset of 3 seconds. The timer discrete status bit (T6) will turn on when the timer has timed for 3 seconds. Notice, in this example, that the timer times for 1 second, stops for one second, then resumes timing. The timer will reset when C10 turns on, turning the discrete status bit off and resetting the timer current value to 0.





Handheld Programmer Keystrokes

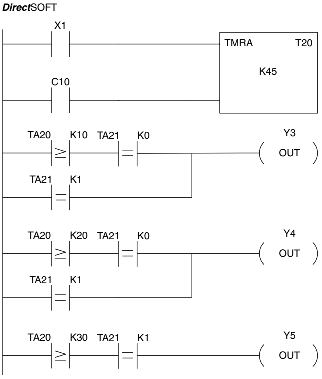
$ STR	→	B 1	ENT			
$ STR	→	SHFT	C 2	B 1	A 0	ENT
N TMR	SHFT	A 0	→	G 6	→	

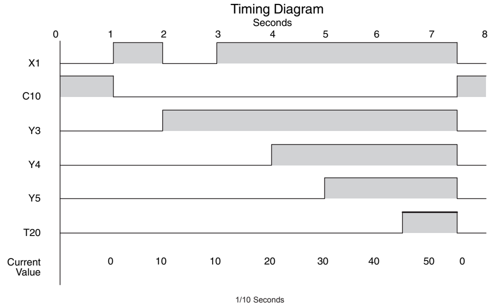
Handheld Programmer Keystrokes (cont)

D 3	A 0	ENT			
$ STR	→	SHFT	T MLR	G 6	ENT
GX OUT	→	B 1	A 0	ENT	

Accumulator Timer Example Using Comparative Contacts

In the following example, a single input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energized Y3, Y4, and Y5 at one second intervals respectively. The comparative contacts will turn off when the timer is reset.





Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
$ STR	→	SHFT	C 2	B 1	A 0	ENT			
N TMR	SHFT	A 0	→	C 2	A 0	→	E 4	F 5	ENT
$ STR	→	SHFT	T MLR	C 2	A 0	→	B 1	A 0	ENT
V AND	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	A 0	ENT
Q OR	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	B 1	ENT
GX OUT	→	D 3	ENT						

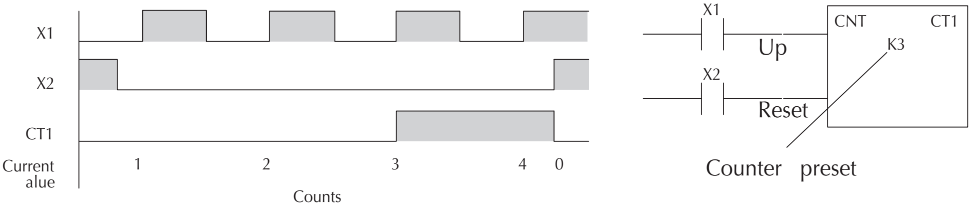
Handheld Programmer Keystrokes (cont'd)

$ STR	→	SHFT	T MLR	C 2	A 0	→	C 2	A 0	ENT	
V AND	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	A 0	ENT
Q OR	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	B 1	ENT
GX OUT	→	E 4	ENT							
$ STR	→	SHFT	T MLR	C 2	A 0	→	D 3	A 0	ENT	
V AND	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	B 1	ENT
GX OUT	→	F 5	ENT							

Using Counters

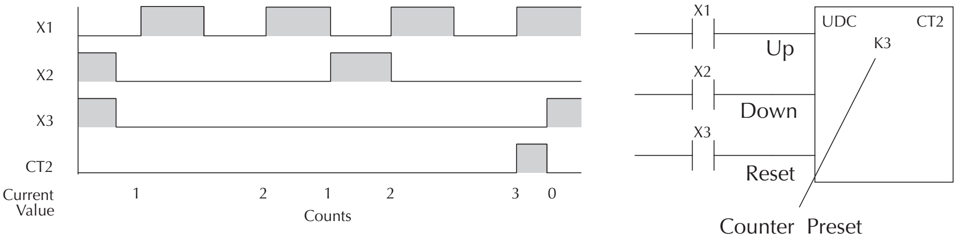
Counters are used to count events . The counters available are up counters, up/down counters, and stage counters (used with RLL^{PLUS} programming).

The up counter (CNT) has two inputs, a count input and a reset input. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, counter reset, associated discrete bit, current value, and counter preset. The CNT counter preset and current value are bothe single word BCD values.

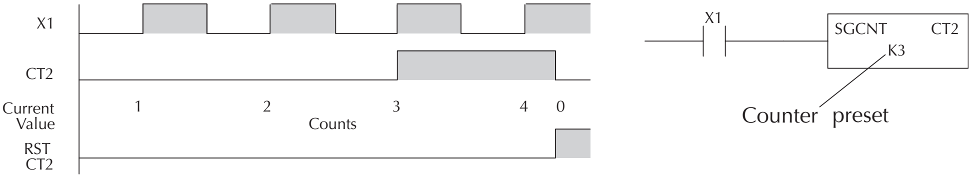


The up down counter (UDC) has three inputs, a count up input, count down input and reset input. The maximum count value is 99999999. The timing diagram below shows the relationship between the counter up and down inputs, counter reset, associated discrete bit, current value, and counter preset. The UDC counter preset and current value are both double word BCD values.

NOTE: *The UDC uses two consecutive V-memory locations for the 8-digit value, therefore, two consecutive timer locations. For example, if UDC CT1 is used, the next available counter number is CT3.*



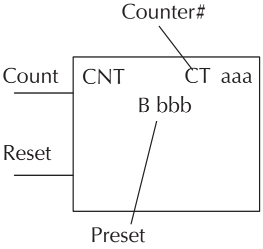
The stage counter (SGCNT) has a count input and is reset by the RST instruction. This instruction is useful when programming using the RLL^{PLUS} structured programming. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, associated discrete bit, current value, counter preset and reset instruction.



Counter (CNT)

DS	Used
HPP	Used

The Counter is a two-input counter that increments when the count input logic transitions from Off to On. When the counter reset input is On, the counter resets to 0. When the current value equals the preset value, the counter status bit comes On and the counter continues to count up to a maximum count of 9999. The maximum value will be held until the counter is reset.



Instruction Specifications

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K) or a V-memory location.

Current Values: Counter current values are accessed by referencing the associated V or CT memory locations.* The V-memory location is the counter location + 1000. For example, the counter current value for CT3 resides in V-memory location V1003.

Discrete Status Bit: The discrete status bit is accessed by referencing the associated CT memory location. It will be On if the value is equal to or greater than the preset value. For example the discrete status bit for counter 2 would be CT2.

***NOTE**: A V-memory preset is required if the ladder program or OIP must change the preset.*

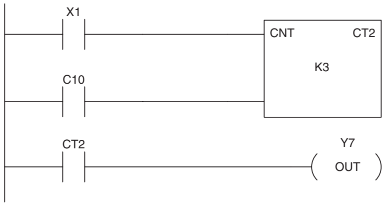
Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Counters	CT	0–177	—
V-memory (preset only)	V	—	400-677 1200–7377 7400–7577 10000-17777
Pointers (preset only)	P	—	400-677 1200–7377 7400–7577* 10000-17777
Constants (preset only)	K	—	0–9999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V /CT**	1000-1177	

***NOTE:** *May be non-volatile if MOV instruction is used.*
*** With the HPP, both the Counter discrete status bits and current value are accessed with the same data reference. DirectSOFT uses separate references, such as “CT2” for discrete status bit for Counter CT2, and “CTA2” for the current value of Counter CT2.*

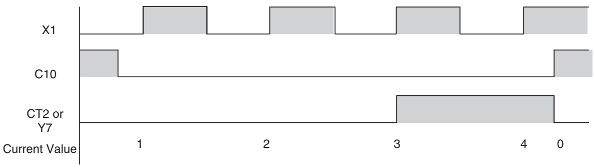
Counter Example Using Discrete Status Bits

In the following example, when X1 makes an Off-to-On transition, counter CT2 will increment by one. When the current value reaches the preset value of 3, the counter status bit CT2 will turn on and energize Y7. When the reset C10 turns on, the counter status bit will turn off and the current value will be 0. The current value for counter CT2 will be held in V-memory location V1002.

*Direct*SOFT



Counting diagram



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
$ STR	→	SHFT	C 2	B 1	A 0	ENT
GY CNT	→	C 2	→	D 3	ENT	

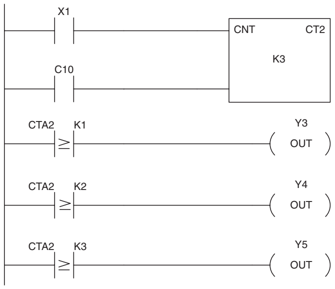
Handheld Programmer Keystrokes (cont)

$ STR	→	SHFT	C 2	SHFT	T MLR	C 2	ENT
GX OUT	→	B 1	A 0	ENT			

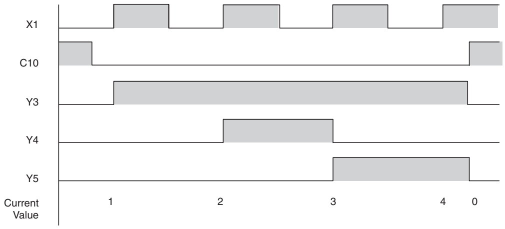
Counter Example Using Comparative Contacts

In the following example, when X1 makes an Off-to-On transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3, Y4, and Y5 at different counts. When the reset C10 turns on, the counter status bit will turn off and the counter current value will be 0, and the comparative contacts will turn off.

*Direct*SOFT



Counting diagram



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
$ STR	→	SHFT	C 2	B 1	A 0	ENT
GY CNT	→	C 2	→	D 3	ENT	
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	B 1	ENT				
GX OUT	→	D 3	ENT			

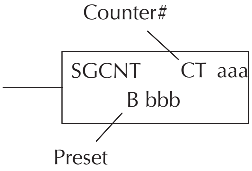
Handheld Programmer Keystrokes (cont)

$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	C 2	ENT				
GX OUT	→	E 4	ENT			
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	D 3	ENT				
GX OUT	→	F 5	ENT			

Stage Counter (SGCNT)

DS	Used
HPP	Used

The Stage Counter is a single input counter that increments when the input logic transitions from off to on. This counter differs from other counters since it will hold its current value until reset using the RST instruction. The Stage Counter is designed for use in RLL^{PLUS} programs but can be used in relay ladder logic programs. When the current value equals the preset value, the counter status bit turns on and the counter continues to count up to a maximum count of 9999. The maximum value will be held until the counter is reset.



Instruction Specifications

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K) or a V-memory location.

Current Values: Counter current values are accessed by referencing the associated V or CT memory locations*. The V-memory location is the counter location + 1000. For example, the counter current value for CT3 resides in V-memory location V1003.

Discrete Status Bit: The discrete status bit is accessed by referencing the associated CT memory location. It will be on if the value is equal to or greater than the preset value. For example, the discrete status bit for counter 2 would be CT2.

***NOTE:** In using a counter inside a stage, the stage must be active for one scan before the input to the counter makes a 0-1 transition. Otherwise, there is no real transition and the counter will not count.*

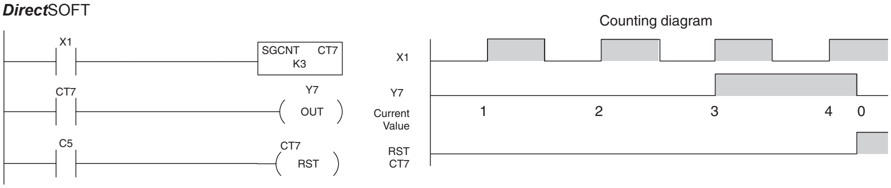
***NOTE**: A V-memory preset is required only if the ladder program or an Operator Interface unit must change the preset.*

Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Counters	CT	0–177	—
V-memory (preset only)	V	—	400-677 1200–7377 7400–7577 10000-17777
Pointers (preset only)	P	—	400-677 1200–7377 7400–7577* 10000-17777
Constants (preset only)	K	—	0–9999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V /CT**	1000-1177	

***NOTE:** *May be non-volatile if MOV instruction is used.*
*** With the HPP, both the Counter discrete status bits and current value are accessed with the same data reference. DirectSOFT uses separate references, such as “CT2” for discrete status bit for Counter CT2, and “CTA2” for the current value of Counter CT2.*

Stage Counter Example Using Discrete Status Bits

In the following example, when X1 makes an off-to-on transition, stage counter CT7 will increment by one. When the current value reaches 3, the counter status bit CT7 will turn on and energize Y7. The counter status bit CT7 will remain on until the counter is reset using the RST instruction. When the counter is reset, the counter status bit will turn off and the counter current value will be 0. The current value for counter CT7 will be held in V-memory location V1007.



Handheld Programmer Keystrokes

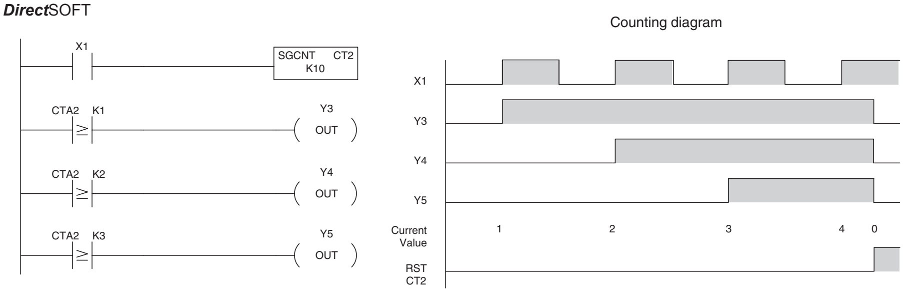
```
$ STR | → | B 1 | ENT
SHFT | S RST | SHFT | G 6 | SHFT | GY CNT | →
H 7 | → | D 3 | ENT
$ STR | → | SHFT | C 2 | SHFT | T MLR | H 7 | ENT
```

Handheld Programmer Keystrokes (cont)

```
GX OUT | → | B 1 | A 0 | ENT
$ STR | → | SHFT | C 2 | F 5 | ENT
S RST | → | SHFT | C 2 | SHFT | T MLR | H 7 | ENT
```

Stage Counter Example Using Comparative Contacts

In the following example, when X1 makes an off-to-on transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3, Y4, and Y5 at different counts. Although this is not shown in the example, when the counter is reset using the Reset instruction, the counter status bit will turn off and the current value will be 0. The current value for counter CT2 will be held in V-memory location V1002 (CTA2).



Handheld Programmer Keystrokes

```
$ STR | → | B 1 | ENT
SHFT | S RST | G 6 | SHFT | GY CNT | →
C 2 | → | B 1 | A 0 | ENT
$ STR | → | SHFT | C 2 | SHFT | T MLR | C 2
→ | B 1 | ENT
GX OUT | → | D 3 | ENT
```

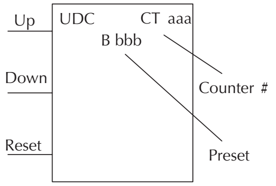
Handheld Programmer Keystrokes (cont)

```
$ STR | → | SHFT | C 2 | SHFT | T MLR | C 2
→ | C 2 | ENT
GX OUT | → | E 4 | ENT
$ STR | → | SHFT | C 2 | SHFT | T MLR | C 2
→ | D 3 | ENT
GX OUT | → | F 5 | ENT
```

Up Down Counter (UDC)

DS	Used
HPP	Used

This Up/Down Counter counts up on each off to on transition of the Up input and counts down on each off-to-on transition of the Down input. The counter is reset to 0 when the Reset input is on. The count range is 0–99999999. The count input not being used must be off in order for the active count input to function.



Caution: The UDC uses two V-memory locations for the 8 digit current value. This means that the UDC uses two consecutive counter locations. If UDC CT1 is used in the program, the next available counter is CT3.

Instruction Specification

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K) or two consecutive V-memory locations, in BCD.

Current Values: Current count is a double word value accessed by referencing the associated V or CT memory locations* in BCD. The V-memory location is the counter location + 1000. For example, the counter current value for CT5 resides in V-memory location V1005 and V1006.

Discrete Status Bit: The discrete status bit is accessed by referencing the associated CT memory location. Operating as a "counter done bit" it will be on if the value is equal to or greater than the preset value. For example the discrete status bit for counter 2 would be CT2.

***NOTE:** The UDC uses two consecutive V-memory locations for the 8-digit value, therefore two consecutive timer locations. For example, if UDC CT1 is used, the next available counter number is CT3.*

***NOTE:** A V-memory preset is required only if the ladder program or an Operator Interface unit must change the preset.*

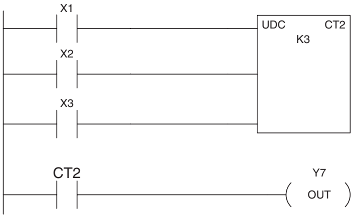
Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Counters	CT	0–177	—
V-memory (preset only)	V	—	400-677 1200–7377 7400–7577 10000-17777
Pointers (preset only)	P	—	400-677 1200–7377* 7400–7577 10000-17777
Constants (preset only)	K	—	0–99999999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V /CT**	1000-1177	

***NOTE:** *May be non-volatile if MOV instruction is used.*
*** With the HPP, both the Counter discrete status bits and current value are accessed with the same data reference. DirectSOFT uses separate references, such as "CT2" for discrete status bit for Counter CT2, and "CTA2" for the current value of Counter CT2.*

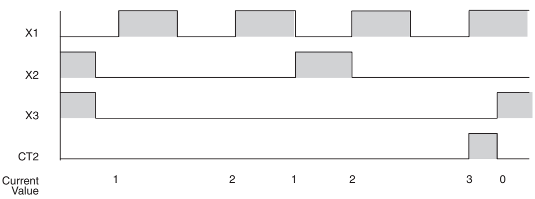
Up / Down Counter Example Using Discrete Status Bits

In the following example, if X2 and X3 are off, the counter will increment by one when X1 toggles from Off to On . If X1 and X3 are off, the counter will decrement by one when X2 toggles from Off to On. When the count value reaches the preset value of 3, the counter status bit will turn on. When the reset X3 turns on, the counter status bit will turn off and the current value will be 0.

*Direct*SOFT



Counting Diagram



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT		
$ STR	→	C 2	ENT		
$ STR	→	D 3	ENT		
SHFT	U ISG	D 3	C 2	→	C 2

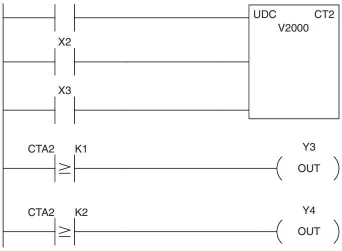
Handheld Programmer Keystrokes (cont)

→	D 3	ENT					
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2	ENT
GX OUT	→	B 1	A 0	ENT			

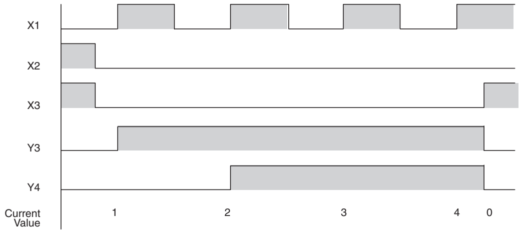
Up / Down Counter Example Using Comparative Contacts

In the following example, when X1 makes an off-to-on transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3 and Y4 at different counts. When the reset (X3) turns on, the counter status bit will turn off, the current value will be 0, and the comparative contacts will turn off.

*Direct*SOFT



Counting Diagram



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
$ STR	→	C 2	ENT			
$ STR	→	D 3	ENT			
SHFT	U ISG	D 3	C 2	→	C 2	→
SHFT	V AND	C 2	A 0	A 0	A 0	ENT
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2

Handheld Programmer Keystrokes (cont)

→	B 1	ENT				
GX OUT	→	D 3	ENT			
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	C 2	ENT				
GX OUT	→	E 4	ENT			

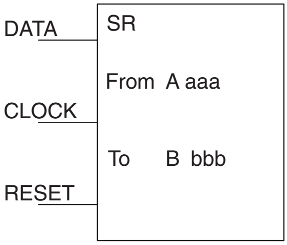
Shift Register (SR)

DS	Used
HPP	Used

The Shift Register instruction shifts data through a predefined number of control relays. The control ranges in the shift register block must start at the beginning of an 8 bit boundary and must use 8-bit blocks.

The Shift Register has three contacts.

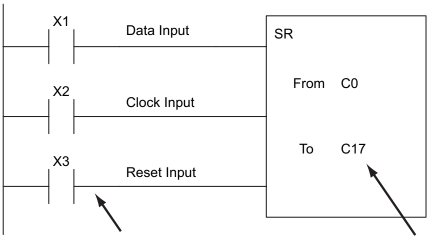
- Data — determines the value (1 or 0) that will enter the register
- Clock — shifts the bits one position on each low to high transition
- Reset —resets the Shift Register to all zeros.



With each off-to-on transition of the clock input, the bits which make up the shift register block are shifted by one bit position and the status of the data input is placed into the starting bit position in the shift register. The direction of the shift depends on the entry in the From and To fields. From C0 to C17 would define a block of sixteen bits to be shifted from left to right. From C17 to C0 would define a block of sixteen bits to be shifted from right to left. The maximum size of the shift register block depends on the number of available control relays. The minimum block size is 8 control relays.

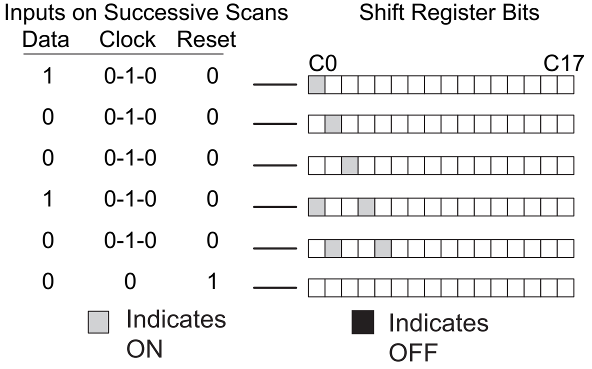
Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Control Relay	C	0–1777	0–1777

*Direct*SOFT



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
$ STR	→	C 2	ENT			
$ STR	→	D 3	ENT			
SHFT	S RST	SHFT	R ORN	SHFT	→	A 0
→	B 1	H 7	ENT			



Data	Clock	Reset
1	0-1-0	0
0	0-1-0	0
0	0-1-0	0
1	0-1-0	0
0	0-1-0	0
0	0	1

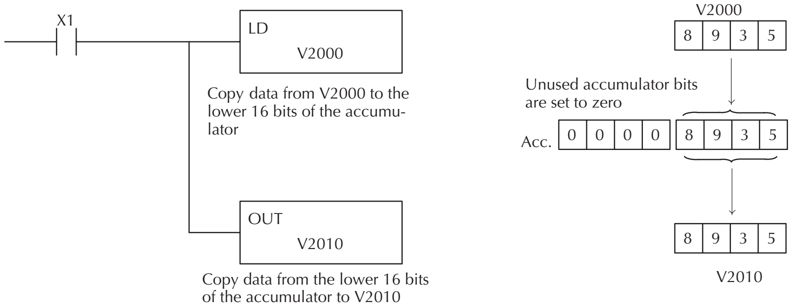
Accumulator/Stack Load and Output Data Instructions

Using the Accumulator

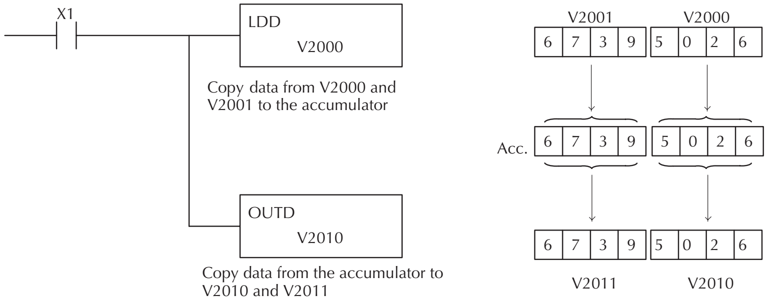
The accumulator in the DL06 internal CPUs is a 32-bit register which is used as a temporary storage location for data that is being copied or manipulated in some manner. For example, you have to use the accumulator to perform math operations such as add, subtract, multiply, etc. Since there are 32 bits, you can use up to an 8-digit BCD number. The accumulator is reset to 0 at the end of every CPU scan.

Copying Data to the Accumulator

The Load and Out instructions and their variations are used to copy data from a V-memory location to the accumulator, or to copy data from the accumulator to V-memory. The following example copies data from V-memory location V2000 to V-memory location V2010.

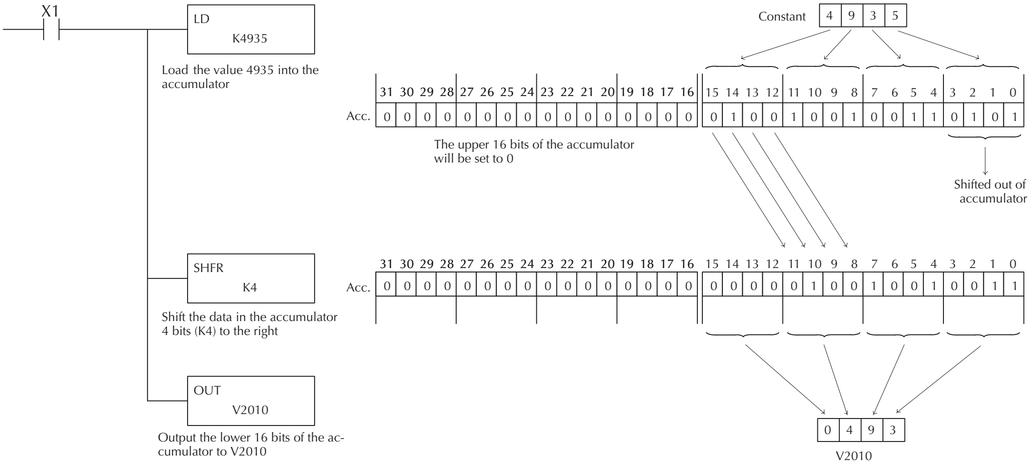

X1
LD
V2000
Copy data from V2000 to the lower 16 bits of the accumulator
OUT
V2010
Copy data from the lower 16 bits of the accumulator to V2010
V2000
8 9 3 5
Unused accumulator bits are set to zero
Acc.
0 0 0 0 8 9 3 5
8 9 3 5
V2010

Since the accumulator is 32 bits and V-memory locations are 16 bits, the Load Double and Out Double (or variations thereof) use two consecutive V-memory locations or 8 digit BCD constants to copy data either to the accumulator from a V-memory address or from a V-memory address to the accumulator. For example, if you wanted to copy data from V2000 and V2001 to V2010 and V2011 the most efficient way to perform this function would be as follows:


X1
LDD
V2000
Copy data from V2000 and V2001 to the accumulator
OUTD
V2010
Copy data from the accumulator to V2010 and V2011
V2001
V2000
6 7 3 9 5 0 2 6
Acc.
6 7 3 9 5 0 2 6
6 7 3 9 5 0 2 6
V2011
V2010

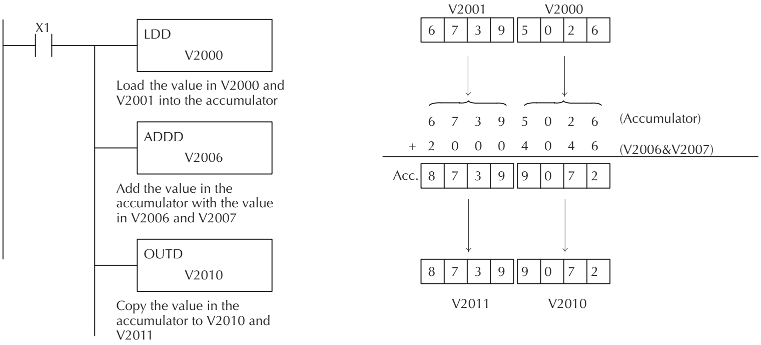
Changing the Accumulator Data

Instructions that manipulate data also use the accumulator. The result of the manipulated data resides in the accumulator. The data that was being manipulated is cleared from the accumulator. The following example loads the constant value 4935 into the accumulator, shifts the data right 4 bits, and outputs the result to V2010.


X1
LD
K4935
Load the value 4935 into the accumulator
Constant 4 9 3 5
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 1 0 1 0 1
The upper 16 bits of the accumulator will be set to 0
Shifted out of accumulator
SHFR
K4
Shift the data in the accumulator 4 bits (K4) to the right
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 1
OUT
V2010
Output the lower 16 bits of the accumulator to V2010
0 4 9 3
V2010

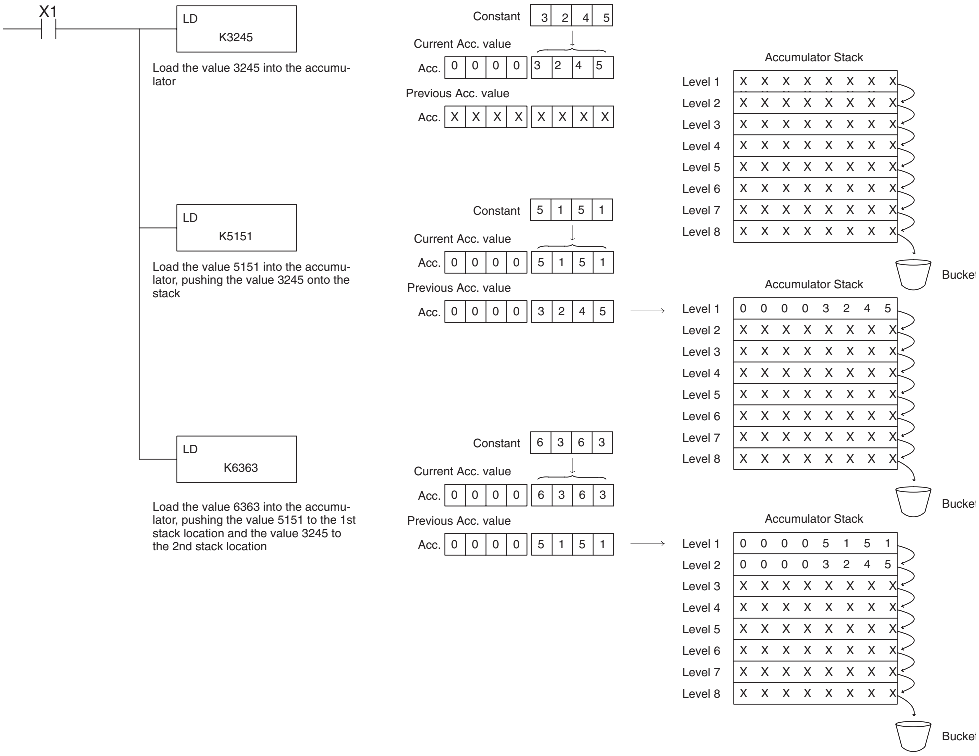
Some of the data manipulation instructions use 32 bits. They use two consecutive V-memory locations or an 8 digit BCD constant to manipulate data in the accumulator.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is added with the value in V2006 and V2007 using the Add Double instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.


X1
LDD
V2000
Load the value in V2000 and V2001 into the accumulator
ADDD
V2006
Add the value in the accumulator with the value in V2006 and V2007
OUTD
V2010
Copy the value in the accumulator to V2010 and V2011
V2001
V2000
6 7 3 9 5 0 2 6
6 7 3 9 5 0 2 6 (Accumulator)
+ 2 0 0 0 4 0 4 6 (V2006&V2007)
Acc. 8 7 3 9 9 0 7 2
8 7 3 9 9 0 7 2
V2011
V2010

Using the Accumulator Stack

The accumulator stack is used for instructions that require more than one parameter to execute a function or for user-defined functionality. The accumulator stack is used when more than one Load instruction is executed without the use of an Out instruction. The first load instruction in the scan places a value into the accumulator. Every Load instruction thereafter without the use of an Out instruction places a value into the accumulator and the value that was in the accumulator is placed onto the accumulator stack. The Out instruction nullifies the previous load instruction and does not place the value that was in the accumulator onto the accumulator stack when the next load instruction is executed. Every time a value is placed onto the accumulator stack the other values in the stack are pushed down one location. The accumulator is eight levels deep (eight 32-bit registers). If there is a value in the eighth location when a new value is placed onto the stack, the value in the eighth location is pushed off the stack and cannot be recovered.

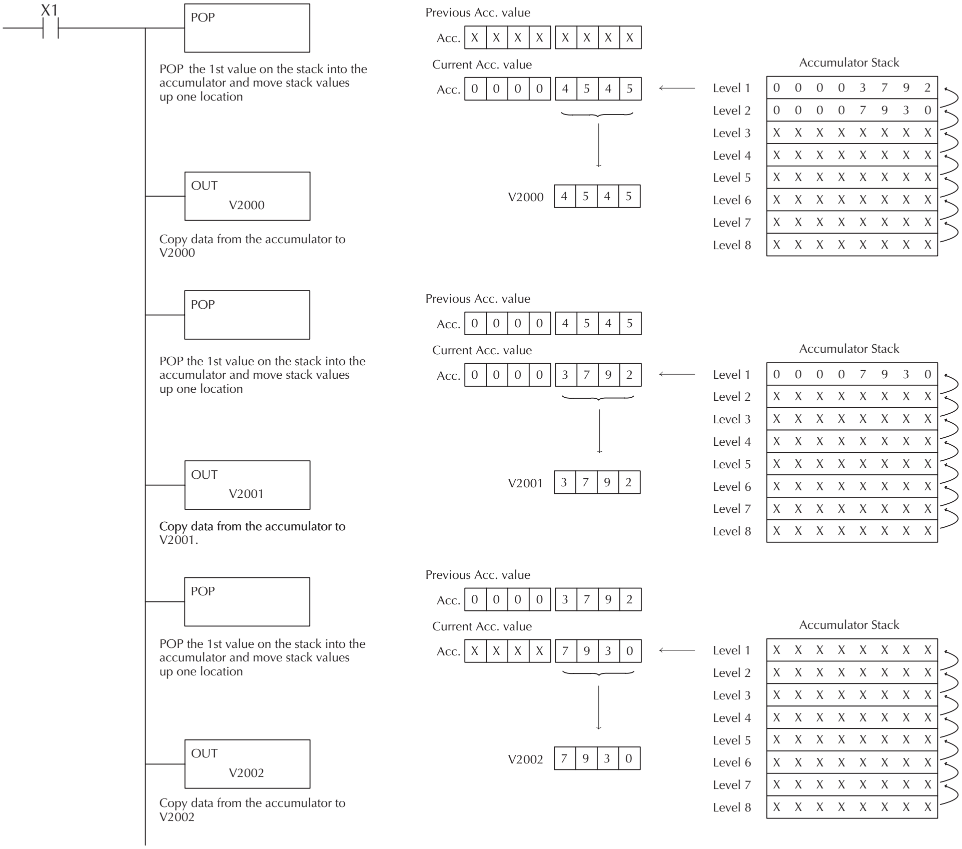


Load the value 3245 into the accumulator

Load the value 5151 into the accumulator, pushing the value 3245 onto the stack

Load the value 6363 into the accumulator, pushing the value 5151 to the 1st stack location and the value 3245 to the 2nd stack location

The POP instruction rotates values upward through the stack into the accumulator. When a POP is executed, the value which was in the accumulator is cleared and the value that was on top of the stack is in the accumulator. The values in the stack are shifted up one position in the stack.

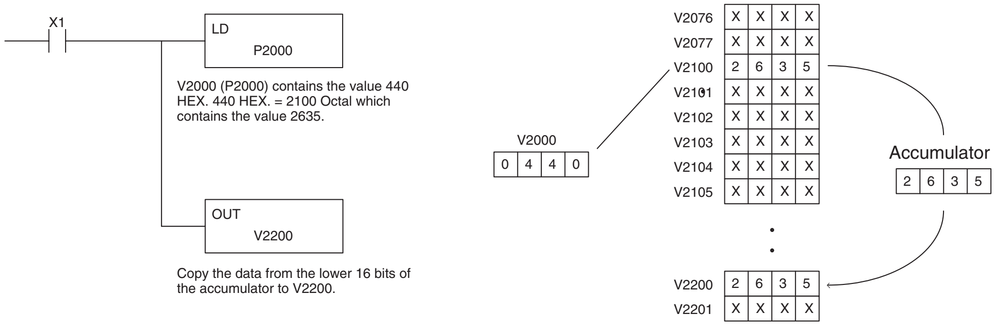


Using Pointers

Many of the DL06 series instructions will allow V-memory pointers as operands (commonly known as indirect addressing). Pointers allow instructions to obtain data from V-memory locations referenced by the pointer value.

***NOTE**: DL06 V-memory addressing is in octal. However, the pointers reference a V-memory location with values viewed as HEX. Use the Load Address (LDA) instruction to move an address into the pointer location. This instruction performs the Octal to Hexadecimal conversion automatically.*

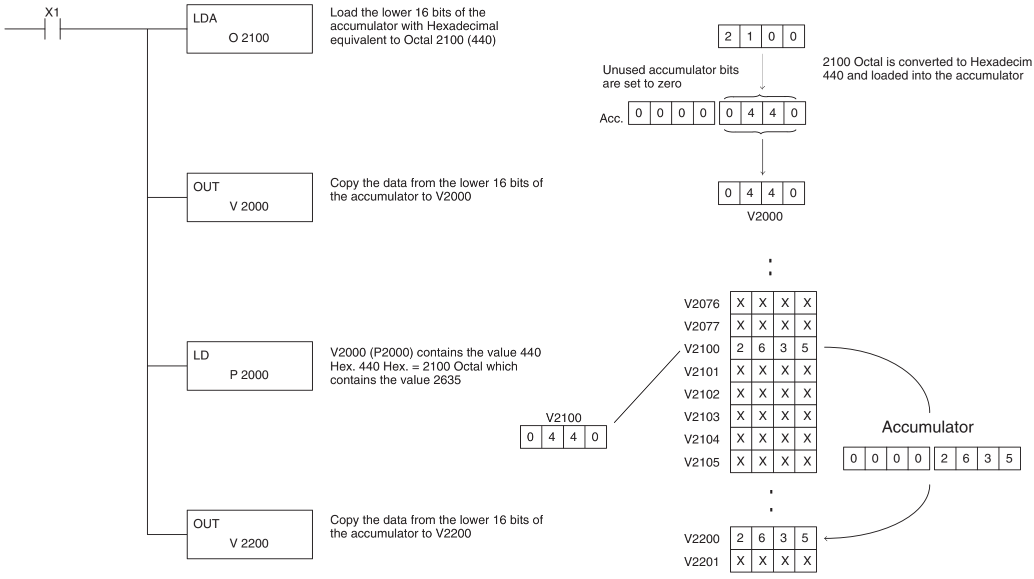
In the following example we are using a pointer operand in a Load instruction. V-memory location 2000 is being used as the pointer location. V2000 contains the value 440 which the CPU views as the Hex equivalent of the Octal address V-memory location V2100. The CPU will copy the data from V2100, which (in this example) contains the value 2635, into the lower word of the accumulator.



V2000 (P2000) contains the value 440 HEX. 440 HEX. = 2100 Octal which contains the value 2635.

Copy the data from the lower 16 bits of the accumulator to V2200.

The following example is identical to the one above, with one exception. The LDA (Load Address) instruction automatically converts the Octal address to Hex.



Load the lower 16 bits of the accumulator with Hexadecimal equivalent to Octal 2100 (440)

Unused accumulator bits are set to zero

2100 Octal is converted to Hexadecim 440 and loaded into the accumulator

Copy the data from the lower 16 bits of the accumulator to V2000

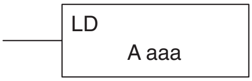
V2000 (P2000) contains the value 440 Hex. 440 Hex. = 2100 Octal which contains the value 2635

Copy the data from the lower 16 bits of the accumulator to V2200

Load (LD)

DS	Used
HPP	Used

The Load instruction is a 16 bit instruction that loads the value (Aaaa), which is either a V-memory location or a 4 digit constant, into the lower 16 bits of the accumulator. The upper 16 bits of the accumulator are set to 0.

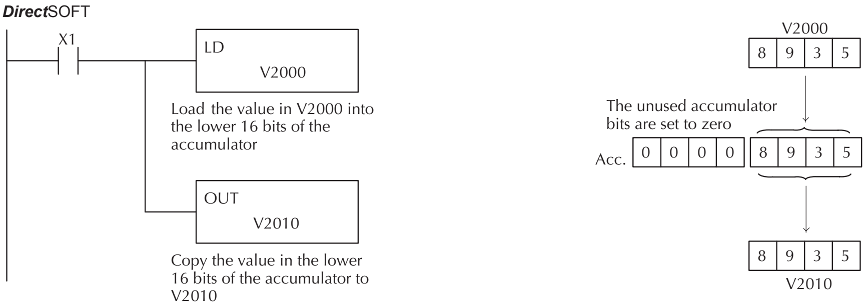


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFF

Discrete Bit Flags	Description
SP53	On when the pointer is outside of the available range.
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator and output to V2010.


*Direct*SOFT
X1
LD
V2000
Load the value in V2000 into the lower 16 bits of the accumulator
OUT
V2010
Copy the value in the lower 16 bits of the accumulator to V2010
V2000
8 9 3 5
The unused accumulator bits are set to zero
Acc.
0 0 0 0 8 9 3 5
8 9 3 5
V2010

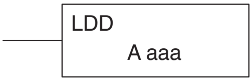
Handheld Programmer Keystrokes

$ STR	→	B 1	X SET					
SHFT	L ANDST	D 3	→					
C 2	A 0	A 0	A 0	ENT				
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

Load Double (LDD)

DS	Used
HPP	Used

The Load Double instruction is a 32-bit instruction that loads the value (Aaaa), which is either two consecutive V-memory locations or an 8 digit constant value, into the accumulator.

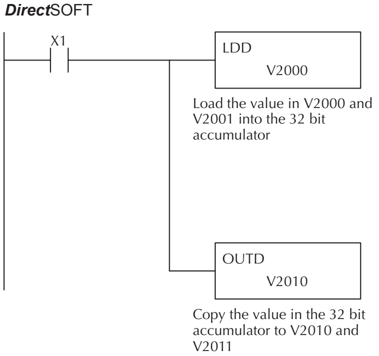


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFFFFFF

Discrete Bit Flags	Description
SP53	On when the pointer is outside of the available range.
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

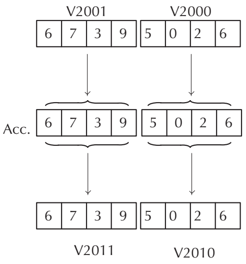
***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

In the following example, when X1 is on, the 32-bit value in V2000 and V2001 will be loaded into the accumulator and output to V2010 and V2011.

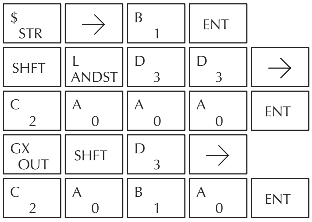


Load the value in V2000 and V2001 into the 32 bit accumulator

Copy the value in the 32 bit accumulator to V2010 and V2011



Handheld Programmer Keystrokes

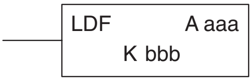


$ STR	→	B 1	ENT	
SHFT	L ANDST	D 3	D 3	→
C 2	A 0	A 0	A 0	ENT
GX OUT	SHFT	D 3	→	
C 2	A 0	B 1	A 0	ENT

Load Formatted (LDF)

DS	Used
HPP	Used

The Load Formatted instruction loads 1–32 consecutive bits from discrete memory locations into the accumulator. The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be loaded. Unused accumulator bit locations are set to zero.

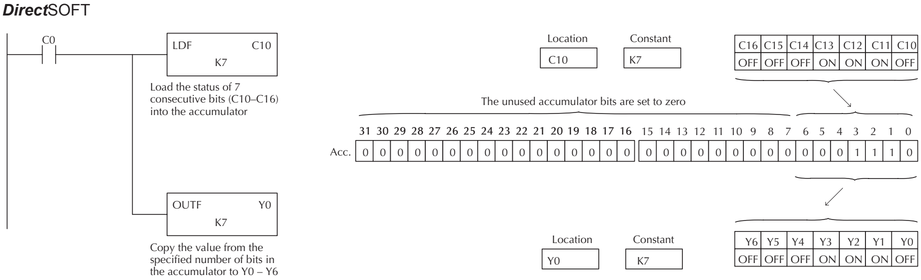


Operand Data Type		DL06 Range	
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0–777	—
Constant	K	—	1–32

Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

In the following example, when C0 is on, the binary pattern of C10–C16 (7 bits) will be loaded into the accumulator using the Load Formatted instruction. The lower 7 bits of the accumulator are output to Y0–Y6 using the Out Formatted instruction.



Handheld Programmer Keystrokes

$ STR | → | SHFT | C 2 | A 0 | ENT

SHFT | L ANDST | D 3 | F 5 | →

SHFT | C 2 | B 1 | A 0 | → | H 7 | ENT

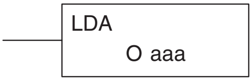
GX OUT | SHFT | F 5 | →

A 0 | → | H 7 | ENT

Load Address (LDA)

DS	Used
HPP	Used

The Load Address instruction is a 16-bit instruction. It converts any octal value or address to the HEX equivalent value and loads the HEX value into the accumulator. This instruction is useful when an address parameter is required, since all addresses for the DL06 system are in octal.



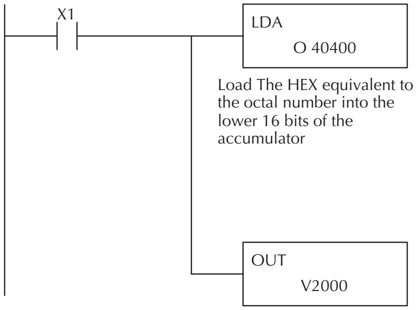
Operand Data Type		DL06 Range
		aaa
Octal Address	O	See memory map

Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

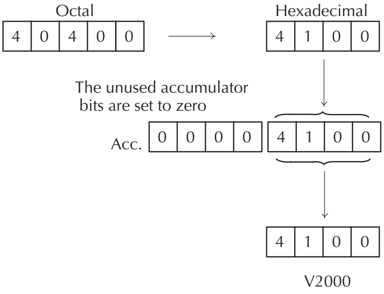
In the following example, when X1 is on, the octal number 40400 will be converted to a HEX 4100 and loaded into the accumulator using the Load Address instruction. The value in the lower 16 bits of the accumulator is copied to V2000 using the Out instruction.

***Direct**SOFT*



Load The HEX equivalent to the octal number into the lower 16 bits of the accumulator

Copy the value in lower 16 bits of the accumulator to V2000



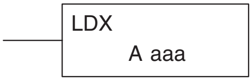
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	A 0	→				
E 4	A 0	E 4	A 0	A 0	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT

Load Accumulator Indexed (LDX)

DS	Used
HPP	Used

Load Accumulator Indexed is a 16-bit instruction that specifies a source address (V-memory) which will be offset by the value in the first stack location. This instruction interprets the value in the first stack location as HEX. The value in the offset address (source address + offset) is loaded into the lower 16 bits of the accumulator. The upper 16 bits of the accumulator are set to 0.



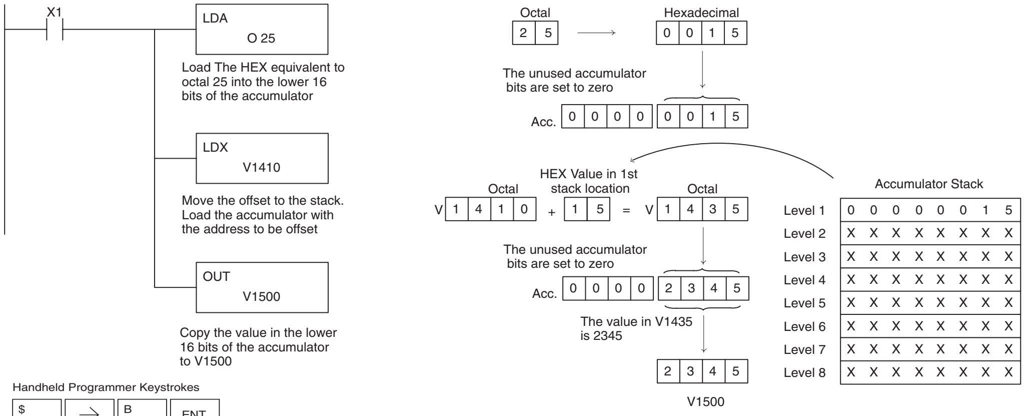
Helpful Hint: — The Load Address instruction can be used to convert an octal address to a HEX address and load the value into the accumulator.

Operand Data Type		DL06 Range	
A		aaa	aaa
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map

Discrete Bit Flags	Description
SP53	On when the pointer is outside of the available range.
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

In the following example, when X1 is on, the HEX equivalent for octal 25 will be loaded into the accumulator (this value will be placed on the stack when the Load Accumulator Indexed instruction is executed). V-memory location V1410 will be added to the value in the first level of the stack and the value in this location (V1435 = 2345) is loaded into the lower 16 bits of the accumulator using the Load Accumulator Indexed instruction. The value in the lower 16 bits of the accumulator is output to V1500 using the Out instruction.


X1
LDA
O 25
Load The HEX equivalent to octal 25 into the lower 16 bits of the accumulator
LDX
V1410
Move the offset to the stack. Load the accumulator with the address to be offset
OUT
V1500
Copy the value in the lower 16 bits of the accumulator to V1500
Octal
2 5
Hexadecimal
0 0 1 5
The unused accumulator bits are set to zero
Acc. 0 0 0 0 0 0 1 5
HEX Value in 1st stack location
Octal
V 1 4 1 0 + 1 5 = V 1 4 3 5
Octal
The unused accumulator bits are set to zero
Acc. 0 0 0 0 2 3 4 5
The value in V1435 is 2345
2 3 4 5
V1500
Accumulator Stack
Level 1 0 0 0 0 0 0 1 5
Level 2 X X X X X X X X
Level 3 X X X X X X X X
Level 4 X X X X X X X X
Level 5 X X X X X X X X
Level 6 X X X X X X X X
Level 7 X X X X X X X X
Level 8 X X X X X X X X

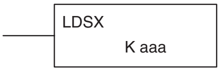
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	A 0	→	C 2	F 5	ENT		
SHFT	L ANDST	D 3	X SET	→	B 1	E 4	B 1	A 0	ENT
GX OUT	→	PREV	PREV	PREV	B 1	F 5	A 0	A 0	ENT

Load Accumulator Indexed from Data Constants (LDSX)

DS	Used
HPP	Used

The Load Accumulator Indexed from Data Constants is a 16-bit instruction. The instruction specifies a Data Label Area (DLBL) where numerical or ASCII constants are stored. This value will be loaded into the lower 16 bits.



The LDSX instruction uses the value in the first level of the accumulator stack as an offset to determine which numerical or ASCII constant within the Data Label Area will be loaded into the accumulator. The LDSX instruction interprets the value in the first level of the accumulator stack as a HEX value.

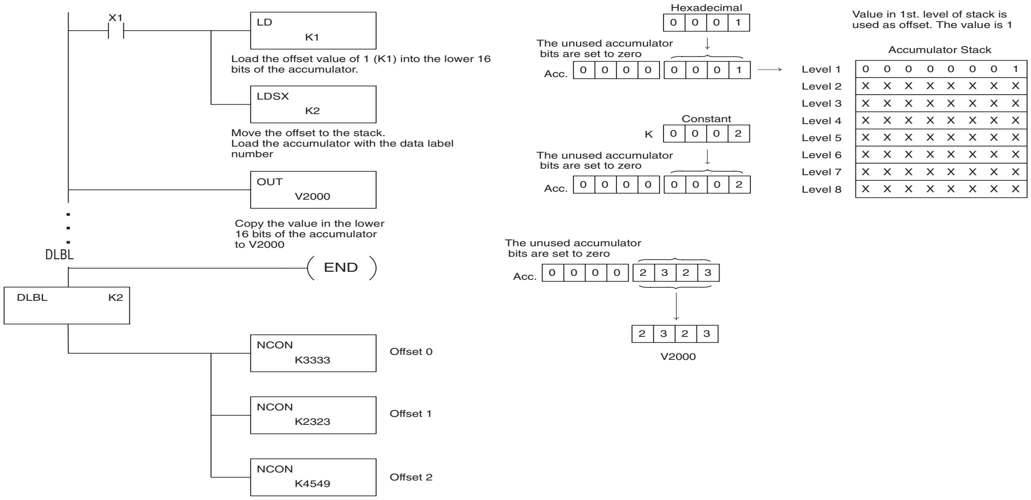
Helpful Hint: — The Load Address instruction can be used to convert octal to HEX and load the value into the accumulator.

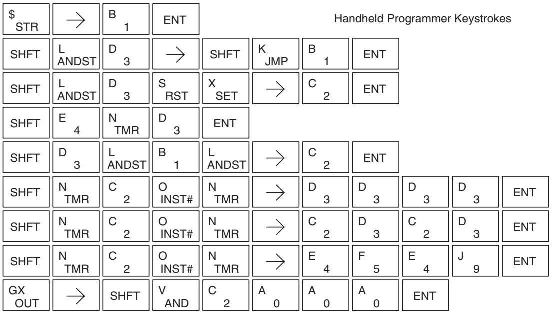
Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

Discrete Bit Flags	Description
SP53	On when the pointer is outside of the available range.
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

In the following example when X1 is on, the offset of 1 is loaded into the accumulator. This value will be placed into the first level of the accumulator stack when the LDSX instruction is executed. The LDSX instruction specifies the Data Label (DLBL K2) where the numerical constant(s) are located in the program and loads the constant value, indicated by the offset in the stack, into the lower 16 bits of the accumulator.

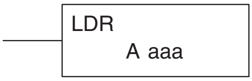

X1
LD
K1
Load the offset value of 1 (K1) into the lower 16 bits of the accumulator.
LDSX
K2
Move the offset to the stack.
Load the accumulator with the data label number
OUT
V2000
Copy the value in the lower 16 bits of the accumulator to V2000
DLBL
END
DLBL K2
NCON
K3333
Offset 0
NCON
K2323
Offset 1
NCON
K4549
Offset 2
Hexadecimal
0 0 0 1
The unused accumulator bits are set to zero
Acc. 0 0 0 0 0 0 0 1
Constant
K 0 0 0 2
The unused accumulator bits are set to zero
Acc. 0 0 0 0 0 0 0 2
The unused accumulator bits are set to zero
Acc. 0 0 0 0 2 3 2 3
2 3 2 3
V2000
Value in 1st. level of stack is used as offset. The value is 1
Accumulator Stack
Level 1 0 0 0 0 0 0 0 1
Level 2 X X X X X X X X
Level 3 X X X X X X X X
Level 4 X X X X X X X X
Level 5 X X X X X X X X
Level 6 X X X X X X X X
Level 7 X X X X X X X X
Level 8 X X X X X X X X


Handheld Programmer Keystrokes
$ STR → B 1 ENT
SHFT L ANDST D 3 → SHFT K JMP B 1 ENT
SHFT L ANDST D 3 S RST X SET → C 2 ENT
SHFT E 4 N TMR D 3 ENT
SHFT D 3 L ANDST B 1 L ANDST → C 2 ENT
SHFT N TMR C 2 O INST# N TMR → D 3 D 3 D 3 D 3 ENT
SHFT N TMR C 2 O INST# N TMR → C 2 D 3 C 2 D 3 ENT
SHFT N TMR C 2 O INST# N TMR → E 4 F 5 E 4 J 9 ENT
GX OUT → SHFT V AND C 2 A 0 A 0 A 0 ENT

Load Real Number (LDR)

DS	Used
HPP	N/A

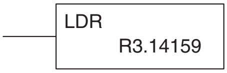
The Load Real Number instruction loads a real number contained in two consecutive V-memory locations, or an 8-digit constant into the accumulator.



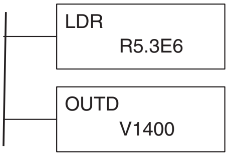
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Real Constant	R	-3.402823E^{+38} to + -3.402823E^{+38}

Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

*Direct*SOFT allows you to enter real numbers directly, by using the leading "R" to indicate a real number entry. You can enter a constant such as Pi, shown in the example to the right. To enter negative numbers, use a minus (–) after the "R".

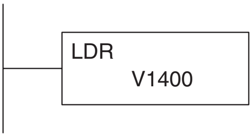


For very large numbers or very small numbers, you can use exponential notation. The number to the right is 5.3 million. The OUTD instruction stores it in V1400 and V1401.



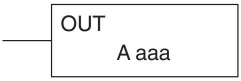
These real numbers are in the IEEE 32-bit floating point format, so they occupy two V-memory locations, regardless of how big or small the number may be! If you view a stored real number in hex, binary, or even BCD, the number shown will be very difficult to decipher. Just like all other number types, you must keep track of real number locations in memory, so they can be read with the proper instructions later.

The previous example above stored a real number in V1400 and V1401. Suppose that now we want to retrieve that number. Just use the Load Real with the V data type, as shown to the right. Next we could perform real math on it, or convert it to a binary number.



Out (OUT)

DS	Used
HPP	Used

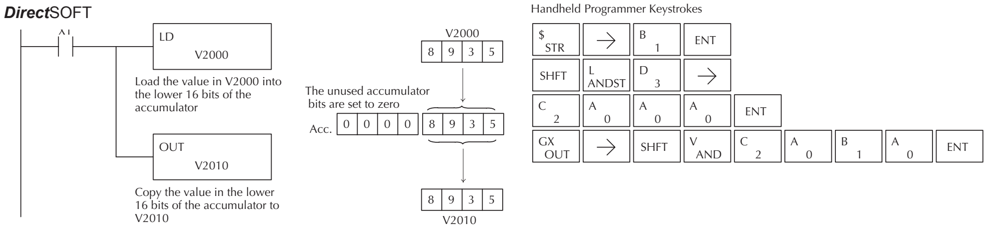


The Out instruction is a 16-bit instruction that copies the value in the lower 16 bits of the accumulator to a specified V-memory location (Aaaa).

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

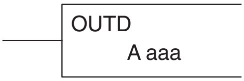
Discrete Bit Flags	Description
SP53	On if CPU cannot solve the logic.

In the following example, when X1 is on, the value in V2000 will be loaded into the lower 16 bits of the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator are copied to V2010 using the OUT instruction.


DirectSOFT
X1
LD
V2000
Load the value in V2000 into the lower 16 bits of the accumulator
OUT
V2010
Copy the value in the lower 16 bits of the accumulator to V2010
V2000
8 9 3 5
The unused accumulator bits are set to zero
Acc.
0 0 0 0 8 9 3 5
8 9 3 5
V2010
Handheld Programmer Keystrokes
$ STR
→
B 1
ENT
SHFT
L ANDST
D 3
→
C 2
A 0
A 0
A 0
ENT
GX OUT
→
SHFT
V AND
C 2
A 0
B 1
A 0
ENT

Out Double (OUTD)

DS	Used
HPP	Used

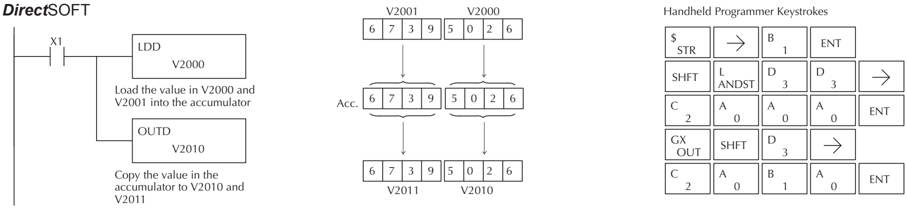


The Out Double instruction is a 32 bit instruction that copies the value in the accumulator to two consecutive V-memory locations at specified starting location (Aaaa).

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP53	On if CPU cannot solve the logic.

In the following example, when X1 is on, the 32-bit value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is output to V2010 and V2011 using the OUTD instruction.



Out Formatted (OUTF)

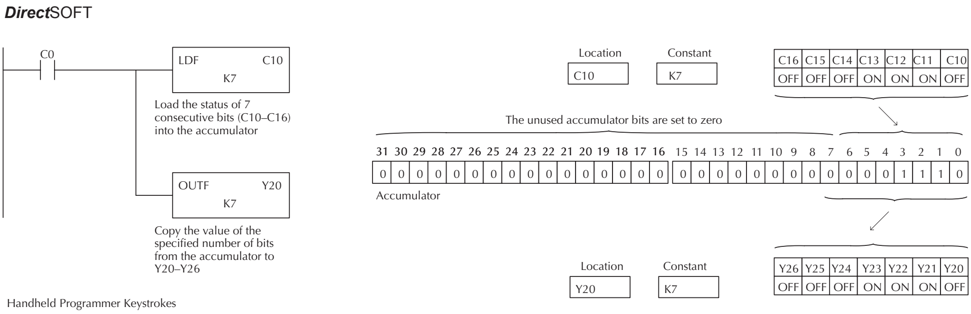
DS	Used
HPP	Used

The Out Formatted instruction outputs 1–32 bits from the accumulator to the specified discrete memory locations. The instruction requires a starting location (Aaaa) for the destination and the number of bits (Kbbb) to be output.

```
OUTF      A aaa
     K bbb
```

Operand Data Type		DL06 Range	
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Constant	K	—	1–32

In the following example, when C0 is on, the binary pattern of C10–C16 (7 bits) will be loaded into the accumulator using the Load Formatted instruction. The lower 7 bits of the accumulator are output to Y0–Y6 using the OUTF instruction.



Load the status of 7 consecutive bits (C10–C16) into the accumulator

Copy the value of the specified number of bits from the accumulator to Y20–Y26

Handheld Programmer Keystrokes

$ STR	→	SHFT	C 2	A 0	ENT	
SHFT	L ANDST	D 3	F 5	→		
SHFT	C 2	B 1	A 0	→	H 7	ENT
GX OUT	SHFT	F 5	→			
A 0	→	H 7	ENT			

Pop (POP)

DS	Used
HPP	Used

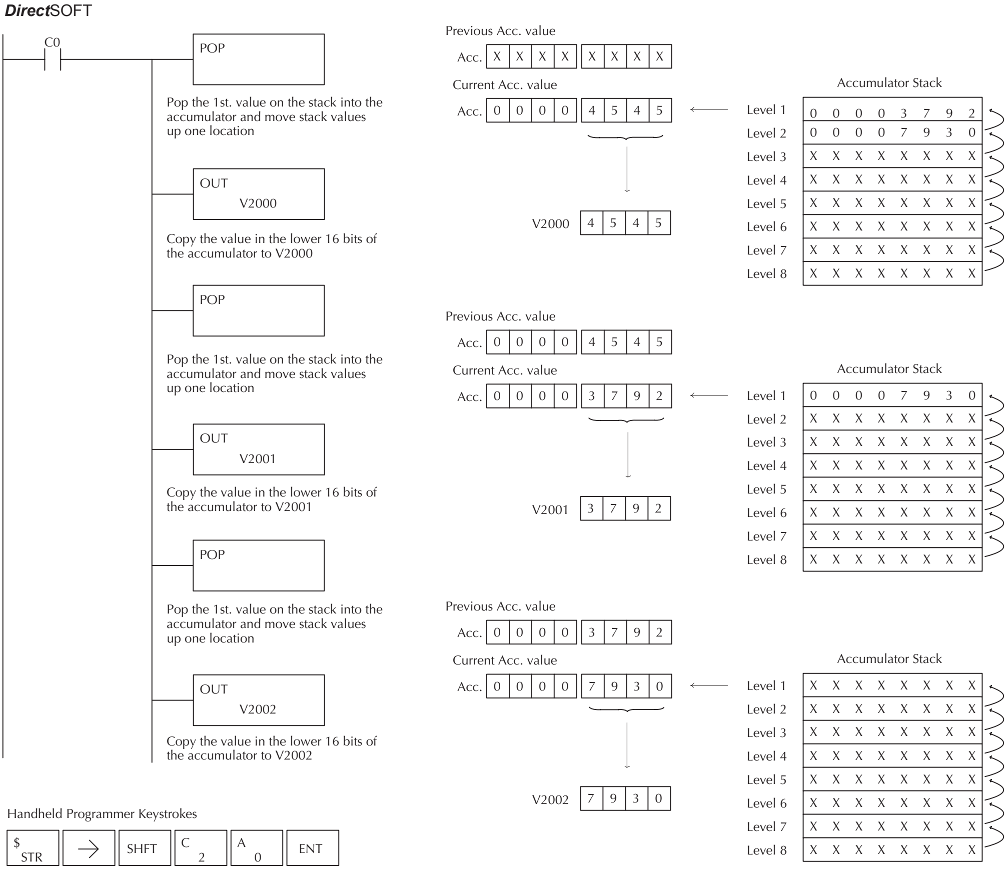
The Pop instruction moves the value from the first level of the accumulator stack (32 bits) to the accumulator and shifts each value in the stack up one level.

```
POP
```

Discrete Bit Flags	Description
SP63	ON when the result of the instruction causes the value in the accumulator to be zero.

Pop Instruction (cont'd)

In the example below, when C0 is on, the value 4545 that was on top of the stack is moved into the accumulator using the Pop instruction The value is output to V2000 using the OUT instruction. The next Pop moves the value 3792 into the accumulator and outputs the value to V2001. The last Pop moves the value 7930 into the accumulator and outputs the value to V2002. Please note if the value in the stack were greater than 16 bits (4 digits) the OUTD instruction would be used and 2 V-memory locations for each OUTD must be allocated.



Pop the 1st. value on the stack into the accumulator and move stack values up one location

Copy the value in the lower 16 bits of the accumulator to V2000

Pop the 1st. value on the stack into the accumulator and move stack values up one location

Copy the value in the lower 16 bits of the accumulator to V2001

Pop the 1st. value on the stack into the accumulator and move stack values up one location

Copy the value in the lower 16 bits of the accumulator to V2002

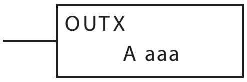
Handheld Programmer Keystrokes

$ STR	→	SHFT	C 2	A 0	ENT			
SHFT	P CV	SHFT	O INST#	P CV	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT
SHFT	P CV	SHFT	O INST#	P CV	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	B 1	ENT
SHFT	P CV	SHFT	O INST#	P CV	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	C 2	ENT

Out Indexed (OUTX)

DS	Used
HPP	Used

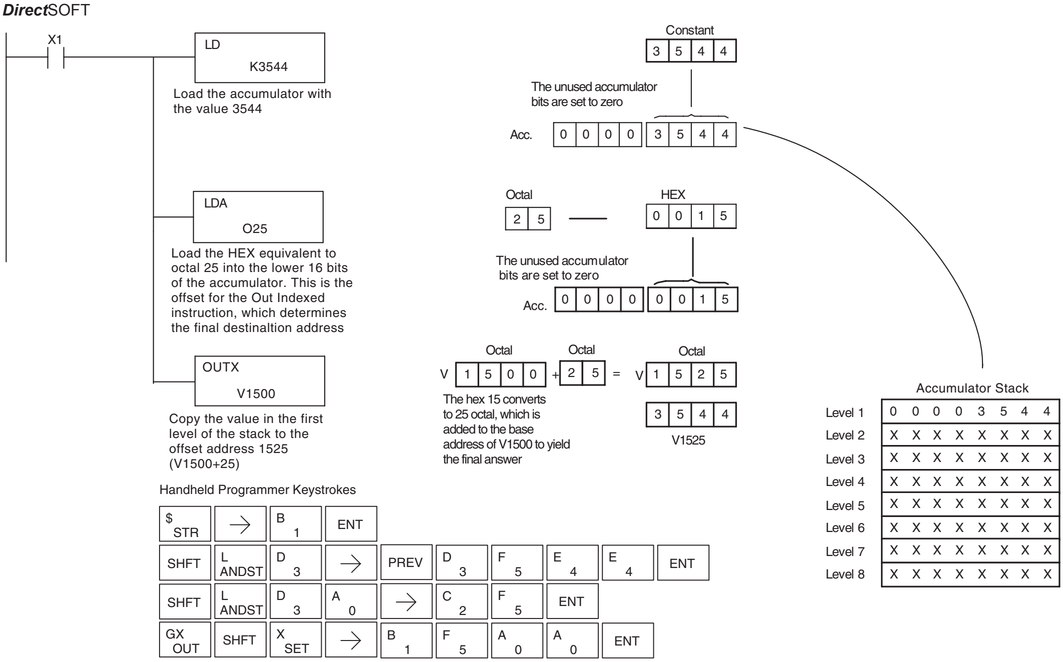
The OUTX instruction is a 16 bit instruction. It copies a 16 bit or 4 digit value from the first level of the accumulator stack to a source address offset by the value in the accumulator(V-memory + offset).This instruction interprets the offset value as a HEX number. The upper 16 bits of the accumulator are set to zero.



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP53	On if CPU cannot solve the logic.

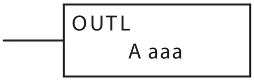
In the following example, when X1 is on, the constant value 3544 is loaded into the accumulator. This is the value that will be output to the specified offset V-memory location (V1525). The value 3544 will be placed onto the stack when the LDA instruction is executed. Remember, two consecutive LD instructions places the value of the first load instruction onto the stack. The LDA instruction converts octal 25 to HEX 15 and places the value in the accumulator. The OUTX instruction outputs the value 3544 which resides in the first level of the accumulator stack to V1525.


DirectSOFT
X1
LD
K3544
Load the accumulator with the value 3544
Constant
3 5 4 4
The unused accumulator bits are set to zero
Acc. 0 0 0 0 3 5 4 4
LDA
O25
Load the HEX equivalent to octal 25 into the lower 16 bits of the accumulator. This is the offset for the Out Indexed instruction, which determines the final destinaltion address
Octal
2 5
HEX
0 0 1 5
The unused accumulator bits are set to zero
Acc. 0 0 0 0 0 0 1 5
OUTX
V1500
Copy the value in the first level of the stack to the offset address 1525 (V1500+25)
Octal
V 1 5 0 0
Octal
+ 2 5
=
Octal
V 1 5 2 5
The hex 15 converts to 25 octal, which is added to the base address of V1500 to yield the final answer
3 5 4 4
V1525
Accumulator Stack
Level 1 0 0 0 0 3 5 4 4
Level 2 X X X X X X X X
Level 3 X X X X X X X X
Level 4 X X X X X X X X
Level 5 X X X X X X X X
Level 6 X X X X X X X X
Level 7 X X X X X X X X
Level 8 X X X X X X X X
Handheld Programmer Keystrokes
$ STR → B 1 ENT
SHFT L ANDST D 3 → PREV D 3 F 5 E 4 E 4 ENT
SHFT L ANDST D 3 A 0 → C 2 F 5 ENT
GX OUT SHFT X SET → B 1 F 5 A 0 A 0 ENT

Out Least (OUTL)

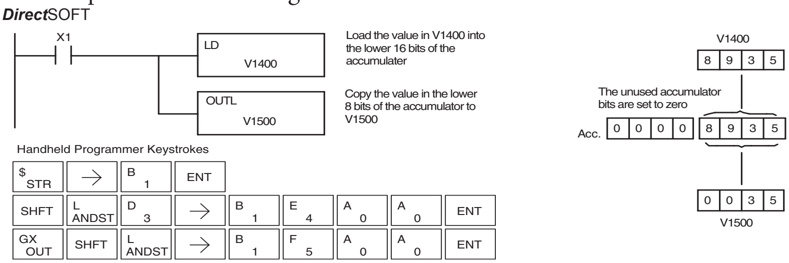
DS	Used
HPP	Used

The OUTL instruction copies the value in the lower eight bits of the accumulator to the lower eight bits of the specified V-memory location (i.e., it copies the low byte of the low word of the accumulator).



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

In the following example, when X1 is on, the value in V1400 will be loaded into the lower 16 bits of the accumulator using the LD instruction. The value in the lower 8 bits of the accumulator is copied to V1500 using the OUTL instruction.


*Direct*SOFT
X1
LD
V1400
OUTL
V1500
Load the value in V1400 into the lower 16 bits of the accumulater
Copy the value in the lower 8 bits of the accumulator to V1500
V1400
8 9 3 5
The unused accumulator bits are set to zero
Acc. 0 0 0 0 8 9 3 5
0 0 3 5
V1500

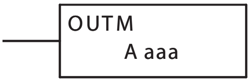
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	B 1	E 4	A 0	A 0	ENT
GX OUT	SHFT	L ANDST	→	B 1	F 5	A 0	A 0	ENT

Out Most (OUTM)

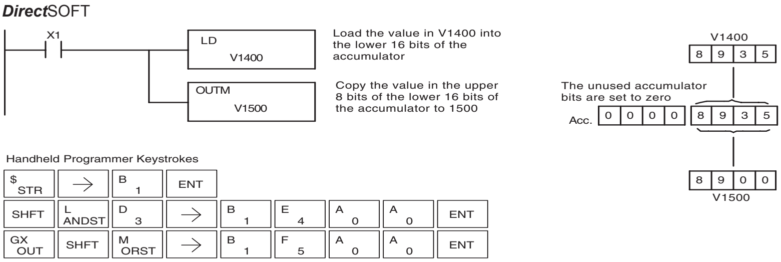
DS	Used
HPP	Used

The OUTM instruction copies the value in the upper eight bits of the lower sixteen bits of the accumulator to the upper eight bits of the specified V-memory location (i.e., it copies the high byte of the low word of the accumulator).



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

In the following example, when X1 is on, the value in V1400 will be loaded into the lower 16 bits of the accumulator using the LD instruction. The value in the upper 8 bits of the lower 16 bits of the accumulator is copied to V1500 using the OUTM instruction.


*Direct*SOFT
X1
LD
V1400
OUTM
V1500
Load the value in V1400 into the lower 16 bits of the accumulator
Copy the value in the upper 8 bits of the lower 16 bits of the accumulator to 1500
V1400
8 9 3 5
The unused accumulator bits are set to zero
Acc. 0 0 0 0 8 9 3 5
8 9 0 0
V1500

Handheld Programmer Keystrokes

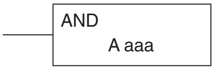
$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	B 1	E 4	A 0	A 0	ENT
GX OUT	SHFT	M ORST	→	B 1	F 5	A 0	A 0	ENT

Logical Instructions (Accumulator)

And (AND logical)

DS	Used
HPP	Used

The AND instruction is a 16-bit instruction that logically ANDs the value in the lower 16 bits of the accumulator with a specified V-memory location (Aaaa). The result resides in the accumulator. The discrete status flag indicates if the result of the AND is zero.



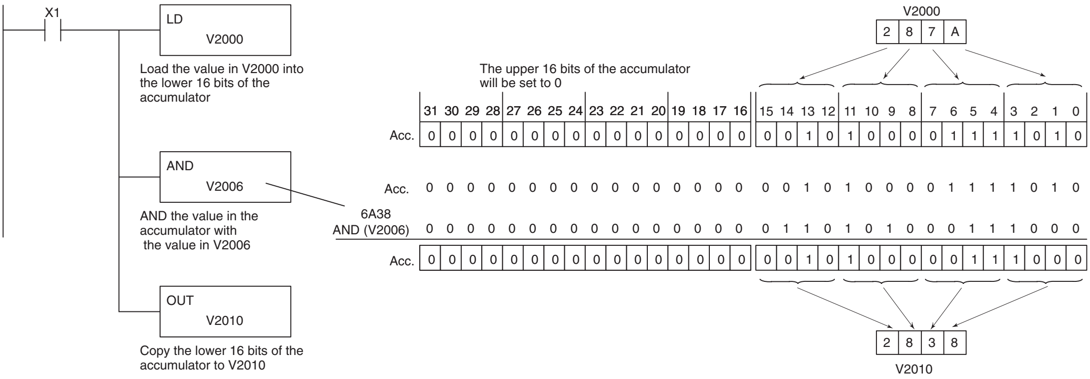
Operand Data Type	A	DL06 Range aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON when the value loaded into the accumulator is zero.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the LD instruction. The value in the accumulator is ANDed with the value in V2006 using the AND instruction. The value in the lower 16 bits of the accumulator is output to V2010 using the OUT instruction.

*Direct*SOFT



Load the value in V2000 into the lower 16 bits of the accumulator

AND the value in the accumulator with the value in V2006

Copy the lower 16 bits of the accumulator to V2010

The upper 16 bits of the accumulator will be set to 0

```
             31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
Acc.          0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0  1  1  1  1  0  1  0

Acc.          0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0  1  1  1  1  0  1  0
6A38
AND (V2006)   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  1  0  1  0  0  0  1  1  1  0  0  0
Acc.          0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0  0  1  1  1  0  0  0
```

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
V AND	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

And Double (ANDD)

DS	Used
HPP	Used

ANDD is a 32-bit instruction that logically ANDs the value in the accumulator with two consecutive V-memory locations or an 8 digit (max.) constant value (Aaaa). The result resides in the accumulator. Discrete status flags indicate if the result of the ANDD is zero or a negative number (the most significant bit is on).

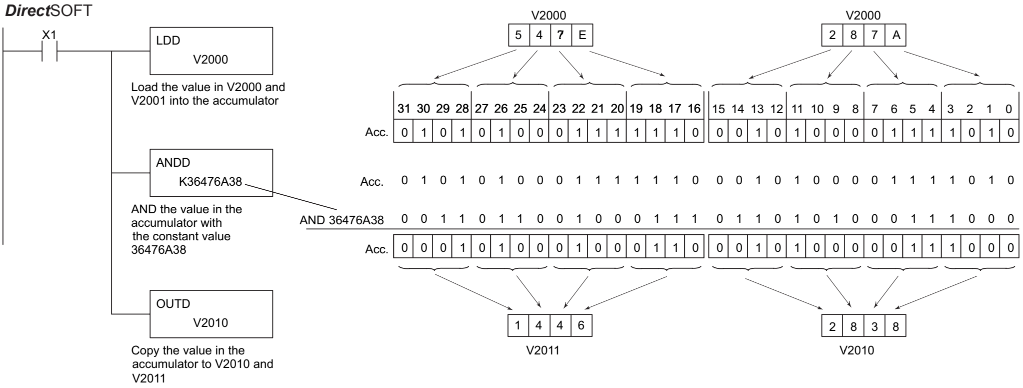
```
ANDD
    K aaa
```

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFFFFFF

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the LDD instruction. The value in the accumulator is ANDed with 36476A38 using the ANDD instruction. The value in the accumulator is output to V2010 and V2011 using the OUTD instruction.


*Direct*SOFT
X1
LDD V2000
Load the value in V2000 and V2001 into the accumulator
ANDD K36476A38
AND the value in the accumulator with the constant value 36476A38
OUTD V2010
Copy the value in the accumulator to V2010 and V2011
V2000
5 4 7 E
V2000
2 8 7 A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
Acc. 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
AND 36476A38 0 0 1 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0
Acc. 0 0 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0
1 4 4 6
V2011
2 8 3 8
V2010

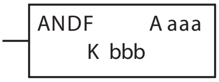
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT													
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT							
V AND	SHFT	D 3	→	SHFT	K JMP	D 3	G 6	E 4	H 7	G 6	SHFT	A 0	SHFT	D 3	I 8	ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT								

And Formatted (ANDF)

DS	Used
HPP	Used

The ANDF instruction logically ANDs the binary value in the accumulator with a specified range of discrete memory bits (1–32). The instruction requires a starting location (Aaaa) and number of bits (Kbbb) to be ANDed. Discrete status flags indicate if the result is zero or a negative number (the most significant bit =1).

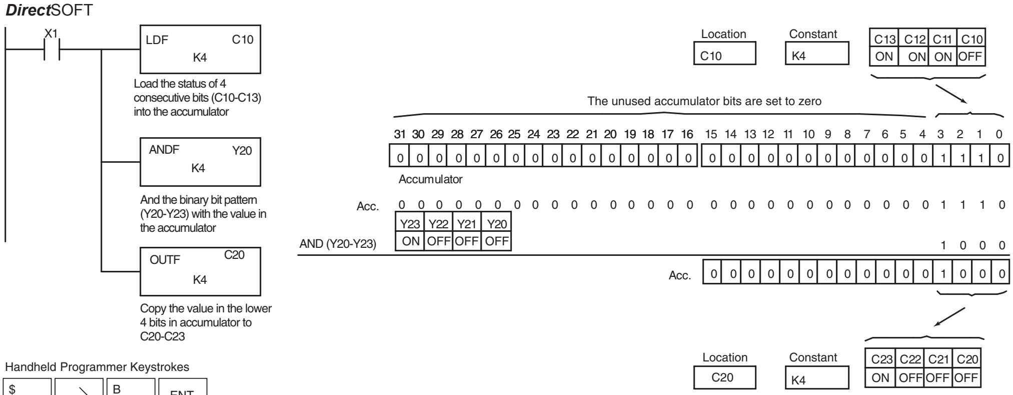


Operand Data Type		DL06 Range	
	B	aaa	bbb
Inputs	X	0-777	-
Outputs	Y	0-777	-
Control Relays	C	0-1777	-
Stage Bits	S	0-1777	-
Timer Bits	T	0-377	-
Counter Bits	CT	177	-
Special Relays	SP	0-777	-
Constant	K	-	1-32

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the LDF instruction loads C10–C13 (4 binary bits) into the accumulator. The accumulator contents is logically ANDed with the bit pattern from Y20–Y23 using the ANDF instruction. The OUTF instruction outputs the accumulator's lower four bits to C20–C23.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT										
SHFT	L ANDST	D 3	F 5	→	NEXT	NEXT	NEXT	NEXT	B 1	A 0	→	E 4	ENT
V AND	SHFT	F 5	→	NEXT	C 2	A 0	→	E 4	ENT				
GX OUT	SHFT	F 5	→	PREV	PREV	C 2	A 0	→	E 4	ENT			

And with Stack (ANDS)

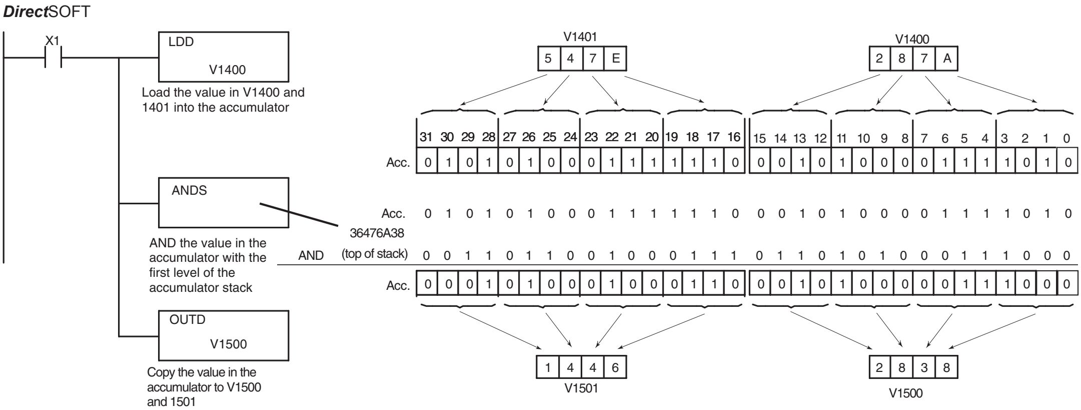
DS	Used
HPP	Used

The ANDS instruction is a 32-bit instruction that logically ANDs the value in the accumulator with the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed from the stack and all values are moved up one level. Discrete status flags indicate if the result of the ANDS is zero or a negative number (the most significant bit is on).

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

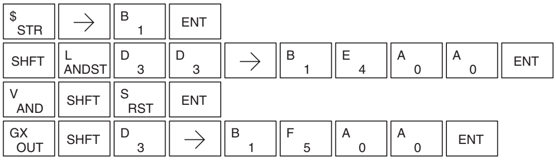
***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the binary value in the accumulator will be ANDed with the binary value in the first level or the accumulator stack. The result resides in the accumulator. The 32-bit value is then output to V1500 and V1501.


*Direct*SOFT
X1
LDD
V1400
Load the value in V1400 and 1401 into the accumulator
ANDS
AND the value in the accumulator with the first level of the accumulator stack
OUTD
V1500
Copy the value in the accumulator to V1500 and 1501
V1401
5 4 7 E
V1400
2 8 7 A
36476A38
AND
1 4 4 6
V1501
2 8 3 8
V1500

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Acc.	0	1	0	1	0	1	0	0	0	1	1	1	1	1	1	0	0	0	1	0	1	0	0	0	0	1	1	1	1	0	1	0
Acc.	0	1	0	1	0	1	0	0	0	1	1	1	1	1	1	0	0	0	1	0	1	0	0	0	0	1	1	1	1	0	1	0
(top of stack)	0	0	1	1	0	1	1	0	0	1	0	0	0	1	1	1	0	1	1	0	1	0	1	0	0	0	1	1	1	0	0	0
Acc.	0	0	0	1	0	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	1	0	0	0	0	0	1	1	1	0	0	0

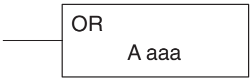
Handheld Programmer Keystrokes



Or (OR)

DS	Used
HPP	Used

The Or instruction is a 16-bit instruction that logically ORs the value in the lower 16 bits of the accumulator with a specified V-memory location (Aaaa). The result resides in the accumulator. The discrete status flag indicates if the result of the OR is zero.

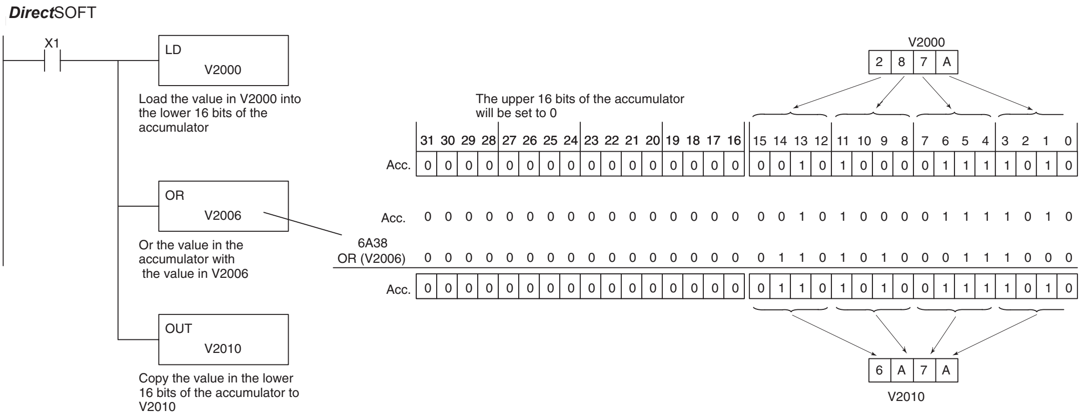


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator is ORed with V2006 using the OR instruction. The value in the lower 16 bits of the accumulator is output to V2010 using the Out instruction.


*Direct*SOFT
X1
LD
V2000
Load the value in V2000 into the lower 16 bits of the accumulator
OR
V2006
Or the value in the accumulator with the value in V2006
OUT
V2010
Copy the value in the lower 16 bits of the accumulator to V2010
V2000
2 8 7 A
The upper 16 bits of the accumulator will be set to 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
6A38
OR (V2006) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 1 1 1 0 1 0
6 A 7 A
V2010

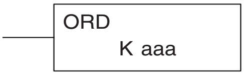
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
Q OR	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

Or Double (ORD)

DS	Used
HPP	Used

ORD is a 32-bit instruction that logically ORs the value in the accumulator with the value (Aaaa), which is either two consecutive V-memory locations or an 8-digit (max.) constant value. The result resides in the accumulator. Discrete status flags indicate if the result of the ORD is zero or a negative number (the most significant bit is on).

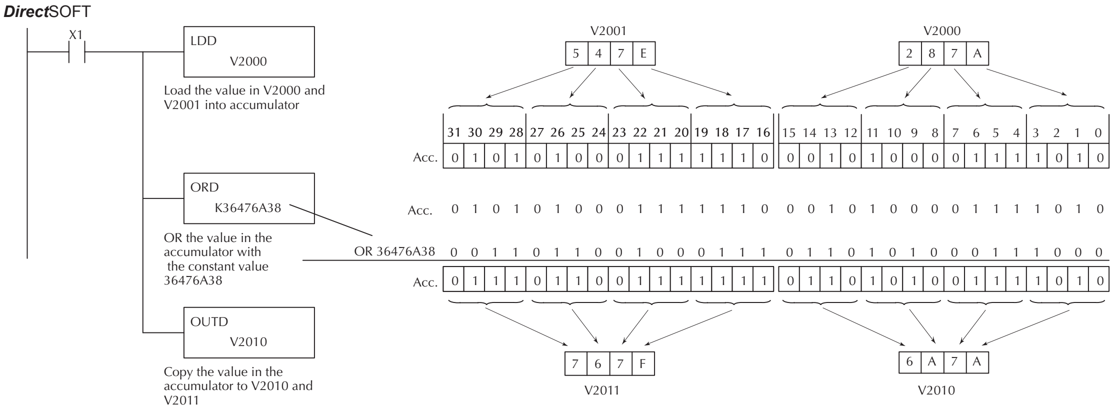


Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFFFFFF

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the LDD instruction. The value in the accumulator is ORed with 36476A38 using the ORD instruction. The value in the accumulator is output to V2010 and V2011 using the OUTD instruction.


DirectSOFT
X1
LDD
V2000
Load the value in V2000 and V2001 into accumulator
ORD
K36476A38
OR the value in the accumulator with the constant value 36476A38
OUTD
V2010
Copy the value in the accumulator to V2010 and V2011
V2001
5 4 7 E
V2000
2 8 7 A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
Acc. 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
OR 36476A38 0 0 1 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0
Acc. 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 1 1 1 0 1 0
7 6 7 F
V2011
6 A 7 A
V2010

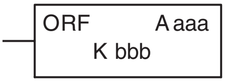
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT													
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT							
Q OR	SHFT	D 3	→	SHFT	K JMP	D 3	G 6	E 4	H 7	G 6	SHFT	A 0	SHFT	D 3	I 8	ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT								

Or Formatted (ORF)

DS	Used
HPP	Used

The ORF instruction logically ORs the binary value in the accumulator and a specified range of discrete bits (1–32). The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be ORed. Discrete status flags indicate if the result is zero or negative (the most significant bit =1).

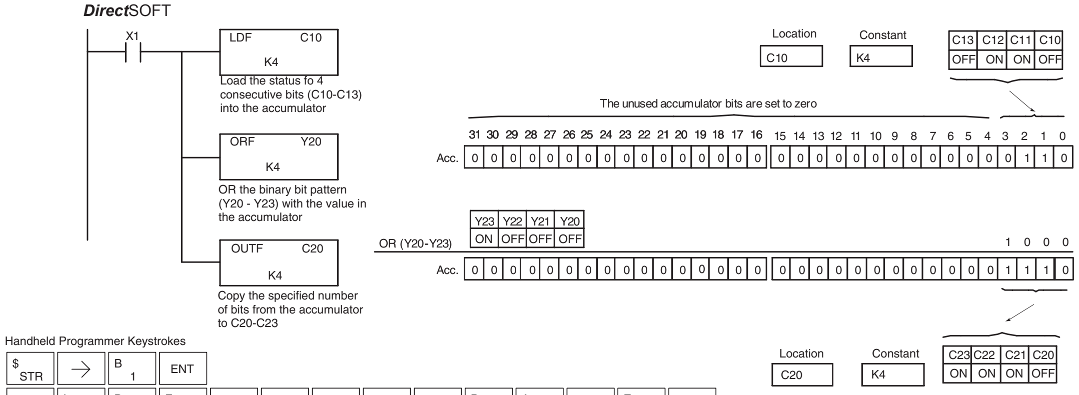


Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Inputs	X	0-777	- -
Outputs	Y	0-777	- -
Control Relays	C	0-1777	- -
Stage Bits	S	0-1777	- -
Timer Bits	T	0-377	- -
Counter Bits	CT	0-177	- -
Special Relays	SP	0-777	- -
Constant	K	-	1-32

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the LDF instruction loads C10–C13 (4 binary bits) into the accumulator. The ORF instruction logically ORs the accumulator contents with Y20–Y23 bit pattern. The ORF instruction outputs the accumulator's lower four bits to C20–C23.


*Direct*SOFT
X1
LDF C10
K4
Load the status fo 4 consecutive bits (C10-C13) into the accumulator
ORF Y20
K4
OR the binary bit pattern (Y20 - Y23) with the value in the accumulator
OUTF C20
K4
Copy the specified number of bits from the accumulator to C20-C23
Location
C10
Constant
K4
C13 C12 C11 C10
OFF ON ON OFF
The unused accumulator bits are set to zero
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
Y23 Y22 Y21 Y20
ON OFF OFF OFF
OR (Y20-Y23)
1 0 0 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0
Location
C20
Constant
K4
C23 C22 C21 C20
ON ON ON OFF

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT										
SHFT	L ANDST	D 3	F 5	→	NEXT	NEXT	NEXT	NEXT	B 1	A 0	→	E 4	ENT
Q OR	SHFT	F 5	→	NEXT	C 2	A 0	→	E 4	ENT				
GX OUT	SHFT	F 5	→	PREV	PREV	C 2	A 0	→	E 4	ENT			

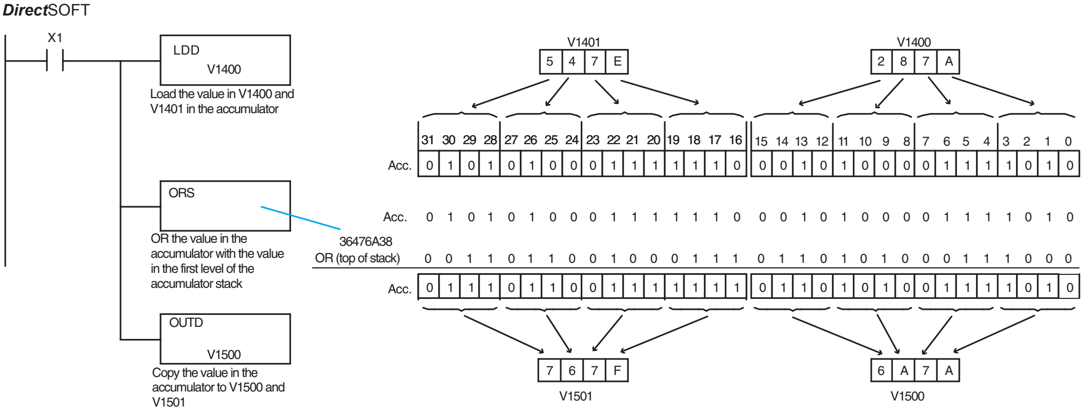
Or with Stack (ORS)

DS	Used
HPP	Used

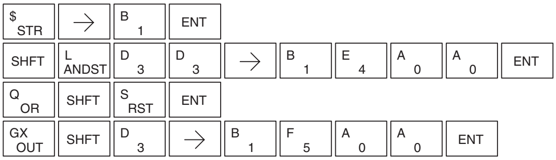
The ORS instruction is a 32-bit instruction that logically ORs the value in the accumulator with the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed from the stack and all values are moved up one level. Discrete status flags indicate if the result of the ORS is zero or a negative number (the most significant bit is on).

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative.

In the following example when X1 is on, the binary value in the accumulator will be ORed with the binary value in the first level of the stack. The result resides in the accumulator.


*Direct*SOFT
X1
LDD
V1400
Load the value in V1400 and V1401 in the accumulator
ORS
OR the value in the accumulator with the value in the first level of the accumulator stack
OUTD
V1500
Copy the value in the accumulator to V1500 and V1501
V1401
5 4 7 E
V1400
2 8 7 A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
Acc. 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
36476A38
OR (top of stack) 0 0 1 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0
Acc. 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 1 1 1 0 1 0
7 6 7 F
V1501
6 A 7 A
V1500

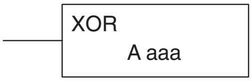
Handheld Programmer Keystrokes


$ STR | → | B 1 | ENT
SHFT | L ANDST | D 3 | D 3 | → | B 1 | E 4 | A 0 | A 0 | ENT
Q OR | SHFT | S RST | ENT
GX OUT | SHFT | D 3 | → | B 1 | F 5 | A 0 | A 0 | ENT

Exclusive Or (XOR)

DS	Used
HPP	Used

The XOR instruction is a 16-bit instruction that performs an exclusive OR of the value in the lower 16 bits of the accumulator and a specified V-memory location (Aaaa). The result resides in the in the accumulator. The discrete status flag indicates if the result of the XOR is zero.

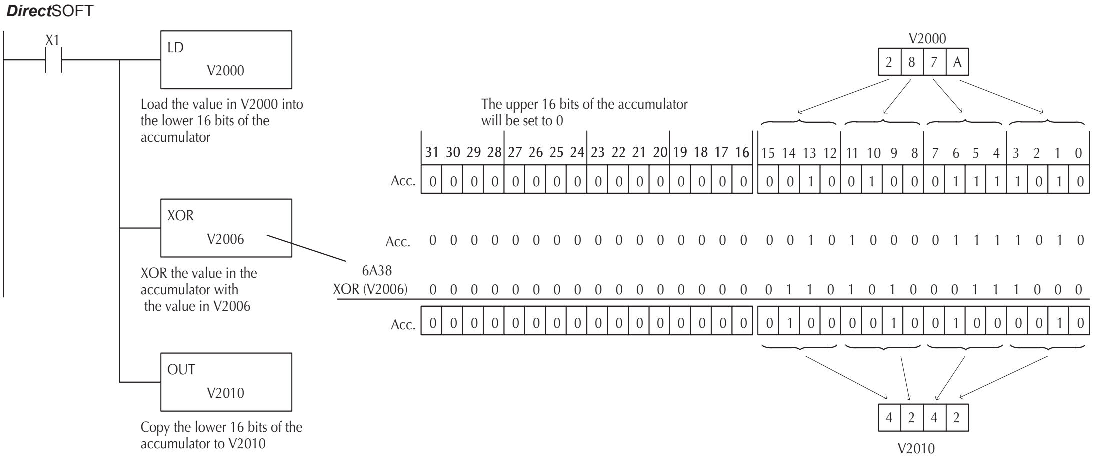


Operand Data Type	A	DL06 Range aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the LD instruction. The value in the accumulator is exclusive ORed with V2006 using the XOR instruction. The value in the lower 16 bits of the accumulator is output to V2010 using the OUT instruction.


DirectSOFT
X1
LD
V2000
Load the value in V2000 into the lower 16 bits of the accumulator
XOR
V2006
XOR the value in the accumulator with the value in V2006
OUT
V2010
Copy the lower 16 bits of the accumulator to V2010
V2000
2 8 7 A
The upper 16 bits of the accumulator will be set to 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 1 1 0 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 1 1 0 1 0
6A38
XOR (V2006) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0
4 2 4 2
V2010

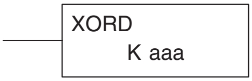
Handheld Programmer Keystrokes

\$ STR	→	SHFT	X SET	B 1	ENT						
SHFT	L ANDST	D 3	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT	
SHFT	X SET	SHFT	Q OR	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT			

Exclusive Or Double (XORD)

DS	Used
HPP	Used

The XORD is a 32-bit instruction that performs an exclusive OR of the value in the accumulator and the value (Aaaa), which is either two consecutive V-memory locations or an 8 digit (max.) constant. The result resides in the accumulator. Discrete status flags indicate if the result of the XORD is zero or a negative number (the most significant bit is on).

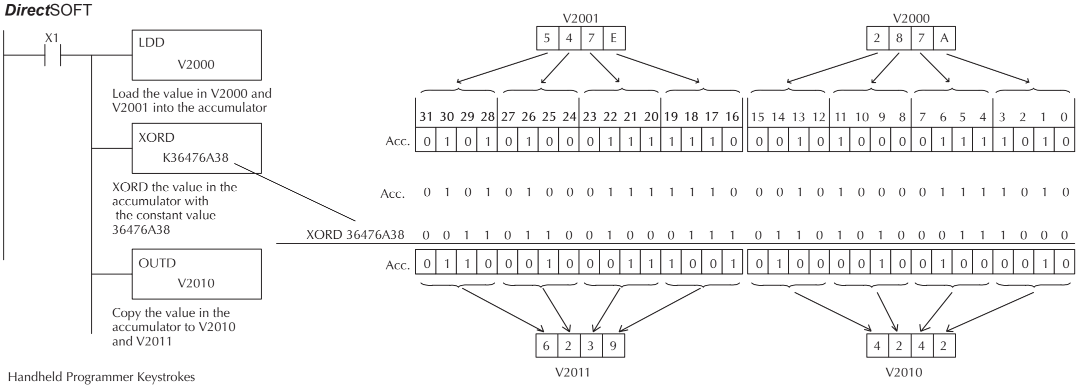


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFFFFFF

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the LDD instruction. The value in the accumulator is exclusively ORed with 36476A38 using the XORD instruction. The value in the accumulator is output to V2010 and V2011 using the OUTD instruction.


*Direct*SOFT
X1
LDD
V2000
Load the value in V2000 and V2001 into the accumulator
XORD
K36476A38
XORD the value in the accumulator with the constant value 36476A38
OUTD
V2010
Copy the value in the accumulator to V2010 and V2011
V2001
5 4 7 E
V2000
2 8 7 A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
Acc. 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
XORD 36476A38 0 0 1 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0
Acc. 0 1 1 0 0 0 1 0 0 0 1 1 1 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0
6 2 3 9
V2011
4 2 4 2
V2010

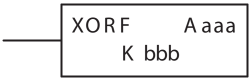
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT	
SHFT	X SET	Q OR	SHFT	D 3	→	SHFT	K JMP			
D 3	G 6	E 4	H 7	G 6	SHFT	A 0	SHFT	D 3	I 8	ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT		

Exclusive Or Formatted (XORF)

DS	Used
HPP	Used

The XORF instruction performs an exclusive OR of the binary value in the accumulator and a specified range of discrete memory bits (1–32).



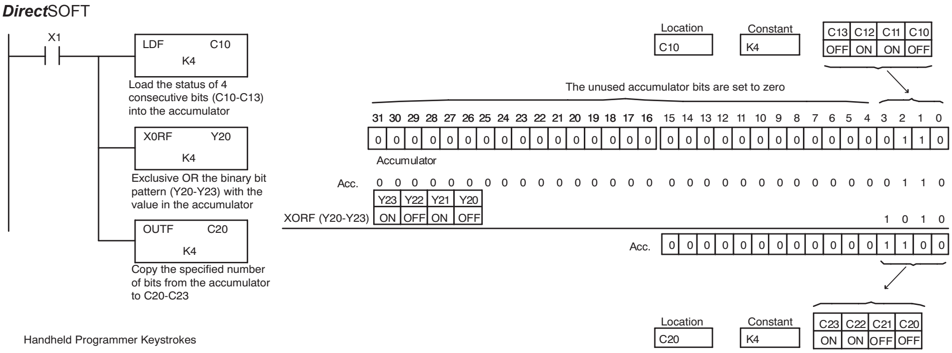
The instruction requires a starting location (Aaaa) and the number of bits (Bbbb) to be exclusive OR'd. Discrete status flags indicate if the result of the XORF is zero or negative (the most significant bit =1).

Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Inputs	X	0-777	-
Outputs	Y	0-777	-
Control Relays	C	0-1777	-
Stage Bits	S	0-1777	-
Timer Bits	T	0-377	-
Counter Bits	CT	177	-
Special Relays	SP	0-777	-
Constant	K	-	1-32

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the binary pattern of C10–C13 (4 bits) will be loaded into the accumulator using the LDF instruction. The value in the accumulator will be logically exclusive ORed with the bit pattern from Y20–Y23 using the XORF instruction. The value in the lower 4 bits of the accumulator is output to C20–C23 using the OUTF instruction.


DirectSOFT
X1
LDF C10
K4
Load the status of 4 consecutive bits (C10-C13) into the accumulator
X0RF Y20
K4
Exclusive OR the binary bit pattern (Y20-Y23) with the value in the accumulator
OUTF C20
K4
Copy the specified number of bits from the accumulator to C20-C23
Location
C10
Constant
K4
C13 C12 C11 C10
OFF ON ON OFF
The unused accumulator bits are set to zero
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
Accumulator
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
Y23 Y22 Y21 Y20
XORF (Y20-Y23) ON OFF ON OFF
1 0 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
Location
C20
Constant
K4
C23 C22 C21 C20
ON ON OFF OFF

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT										
SHFT	L ANDST	D 3	F 5	→	NEXT	NEXT	NEXT	NEXT	B 1	A 0	→	E 4	ENT
SHFT	X SET	Q OR	SHFT	F 5	→	NEXT	C 2	A 0	→	E 4	ENT		
GX OUT	SHFT	F 5	→	PREV	PREV	C 2	A 0	→	E 4	ENT			

Exclusive Or with Stack (XORS)

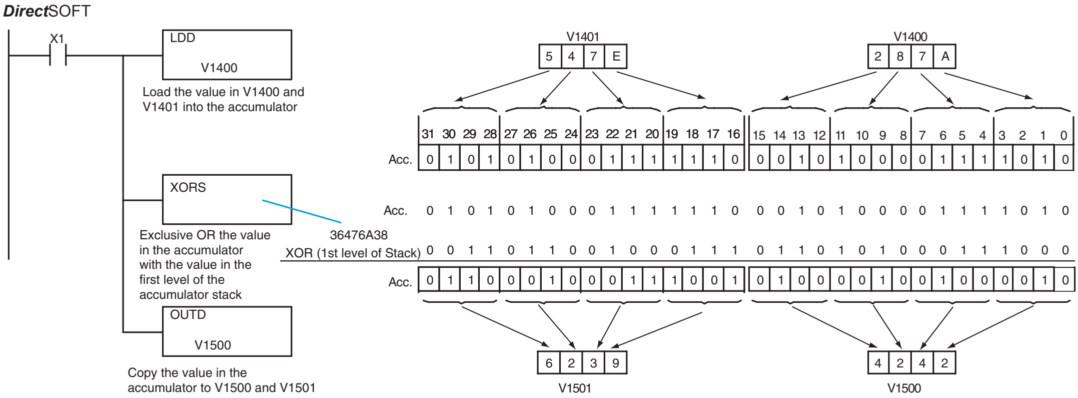
DS	Used
HPP	Used

The XORS instruction is a 32-bit instruction that performs an Exclusive Or of the value in the accumulator with the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed from the stack and all values are moved up one level. Discrete status flags indicate if the result of the XORS is zero or a negative number (the most significant bit is on).

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the LDD instruction. The binary value in the accumulator will be exclusively ORed with 36476A38 using the XORS instruction. The value in the accumulator is output to V1500 and V1501 using the OUTD instruction.


DirectSOFT
X1
LDD
V1400
Load the value in V1400 and V1401 into the accumulator
XORS
Exclusive OR the value in the accumulator with the value in the first level of the accumulator stack
OUTD
V1500
Copy the value in the accumulator to V1500 and V1501
V1401
5 4 7 E
V1400
2 8 7 A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
Acc. 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0
36476A38
XOR (1st level of Stack) 0 0 1 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0
Acc. 0 1 1 0 0 0 1 0 0 0 1 1 1 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0
6 2 3 9
V1501
4 2 4 2
V1500

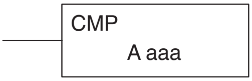
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	X SET	Q OR	SHFT	S RST	ENT				
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Compare (CMP)

DS	Used
HPP	Used

The CMP instruction is a 16-bit instruction that compares the value in the lower 16 bits of the accumulator with the value in a specified V-memory location (Aaaa). The corresponding status flag will be turned on indicating the result of the comparison. The data format for this instruction is BCD/Hex, Decimal and Binary.

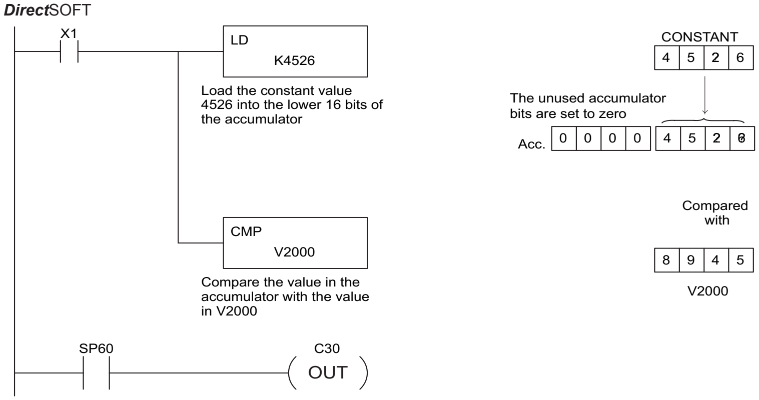


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example when X1 is on, the constant 4526 will be loaded into the lower 16 bits of the accumulator using the LD instruction. The value in the accumulator is compared with the value in V2000 using the CMP instruction. The corresponding discrete status flag will be turned on indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the CMP instruction, SP60 will turn on, energizing C30.


*Direct*SOFT
X1
LD
K4526
Load the constant value 4526 into the lower 16 bits of the accumulator
CONSTANT
4 5 2 6
The unused accumulator bits are set to zero
Acc. 0 0 0 0 4 5 2 6
CMP
V2000
Compare the value in the accumulator with the value in V2000
Compared with
8 9 4 5
V2000
SP60
C30
OUT

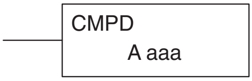
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	F 5	C 2	G 6	ENT
SHFT	C 2	SHFT	M ORST	P CV	→	C 2	A 0	A 0	A 0	ENT
$ STR	→	SHFT	SP STRN	G 6	A 0	ENT				
GX OUT	→	SHFT	C 2	D 3	A 0	ENT				

Compare Double (CMPD)

DS	Used
HPP	Used

The Compare Double instruction is a 32–bit instruction that compares the value in the accumulator with the value (Aaaa), which is either two consecutive V-memory locations or an 8–digit (max.) constant. The corresponding status flag will be turned on indicating the result of the comparison. The data format for this instruction is BCD/Hex, Decimal and Binary.

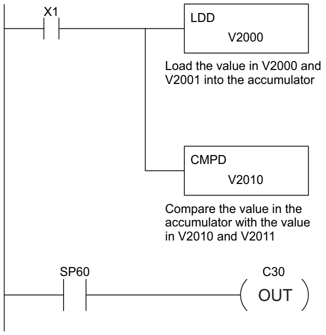


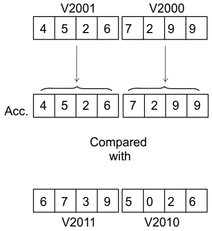
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFFFFFF

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is compared with the value in V2010 and V2011 using the CMPD instruction. The corresponding discrete status flag will be turned on, indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the Compare instruction, SP60 will turn on energizing C30.


X1
LDD
V2000
Load the value in V2000 and V2001 into the accumulator
CMPD
V2010
Compare the value in the accumulator with the value in V2010 and V2011
SP60
C30
OUT



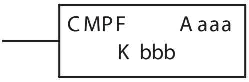
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT								
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT		
SHFT	C 2	SHFT	M ORST	P CV	D 3	→	C 2	A 0	B 1	A 0	ENT
$ STR	→	SHFT	SP STRN	G 6	A 0	ENT					
GX OUT	→	SHFT	C 2	D 3	A 0	ENT					

Compare Formatted (CMPF)

DS	Used
HPP	Used

The Compare Formatted instruction compares the value in the accumulator with a specified number of discrete locations (1–32). The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be compared. The corresponding status flag will be turned on, indicating the result of the comparison. The data format for this instruction is BCD/Hex, Decimal and Binary.



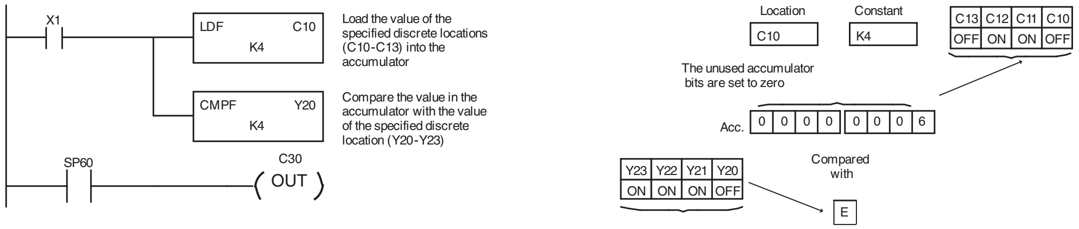
Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Inputs	X	0-777	-
Outputs	Y	0-777	-
Control Relays	C	0-1777	-
Stage Bits	S	0-1777	-
Timer Bits	T	0-377	-
Counter Bits	CT	0-177	-
Special Relays	SP	0-777	-
Constant	K	-	1-32

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the Load Formatted instruction loads the binary value (6) from C10–C13 into the accumulator. The CMPF instruction compares the value in the accumulator to the value in Y20–Y23 (E hex). The corresponding discrete status flag will be turned on, indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the Compare instruction, SP60 will turn on energizing C30.

***Direct**SOFT*


X1
LDF C10
K4
Load the value of the specified discrete locations (C10-C13) into the accumulator
CMPF Y20
K4
Compare the value in the accumulator with the value of the specified discrete location (Y20-Y23)
SP60
C30
OUT
Location
C10
Constant
K4
C13 C12 C11 C10
OFF ON ON OFF
The unused accumulator bits are set to zero
Acc.
0 0 0 0 0 0 0 6
Y23 Y22 Y21 Y20
ON ON ON OFF
Compared with
E

Compare with Stack (CMPS)

DS	Used
HPP	Used

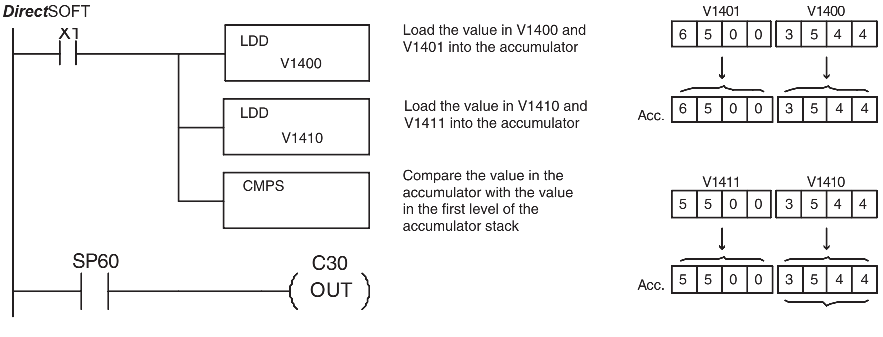
The Compare with Stack instruction is a 32-bit instruction that compares the value in the accumulator with the value in the first level of the accumulator stack. The data format for this instruction is BCD/Hex, Decimal and Binary.

The corresponding status flag will be turned on, indicating the result of the comparison. This does not affect the value in the accumulator.

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The value in V1410 and V1411 is loaded into the accumulator using the Load Double instruction. The value that was loaded into the accumulator from V1400 and V1401 is placed on top of the stack when the second Load instruction is executed. The value in the accumulator is compared with the value in the first level of the accumulator stack using the CMPS instruction. The corresponding discrete status flag will be turned on indicating the result of the comparison. In this example, if the value in the accumulator is less than the value in the stack, SP60 will turn on, energizing C30.


DirectSOFT
X1
LDD
V1400
LDD
V1410
CMPS
SP60
C30
OUT
Load the value in V1400 and V1401 into the accumulator
Load the value in V1410 and V1411 into the accumulator
Compare the value in the accumulator with the value in the first level of the accumulator stack
V1401
V1400
6 5 0 0
3 5 4 4
Acc.
6 5 0 0
3 5 4 4
V1411
V1410
5 5 0 0
3 5 4 4
Acc.
5 5 0 0
3 5 4 4
Compared with Top of Stack

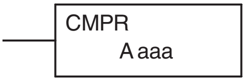
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	B 1	A 0	ENT
SHFT	C 2	SHFT	M ORST	P CV	S RST	ENT			
$ STR	PREV	G 6	A 0	ENT					
GX OUT	→	NEXT	NEXT	NEXT	SHFT	C 2	D 3	A 0	ENT

Compare Real Number (CMPR)

DS	Used
HPP	Used

The Compare Real Number instruction compares a real number value in the accumulator with two consecutive V-memory locations containing a real number. The corresponding status flag will be turned on, indicating the result of the comparison. Both numbers being compared are 32 bits long.

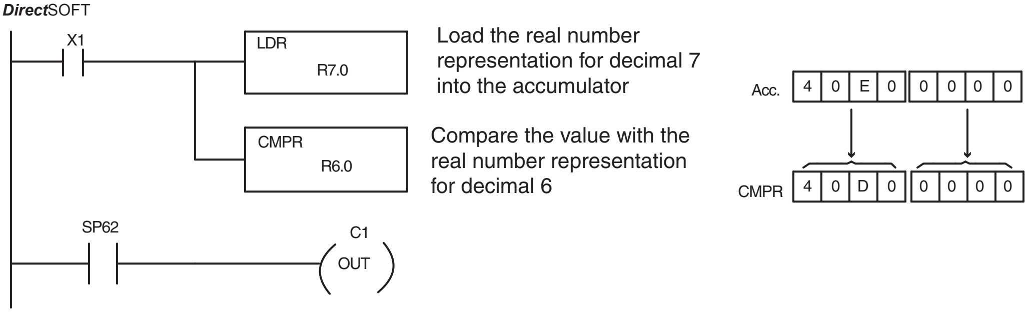


Operand Data Type	A	DL06 Range aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	R	-3.402823E+ 038 to + -3.402823E+ 038

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.
SP71	On anytime the V-memory specified by a pointer (P) is not valid

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the LDR instruction loads the real number representation for 7 decimal into the accumulator. The CMPR instruction compares the accumulator contents with the real representation for decimal 6. Since 7 > 6, the corresponding discrete status flag is turned on (special relay SP62), turning on control relay C1.

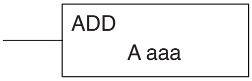

DirectSOFT
X1
LDR
R7.0
Load the real number representation for decimal 7 into the accumulator
Acc.
4 0 E 0 0 0 0 0
CMPR
R6.0
Compare the value with the real number representation for decimal 6
CMPR
4 0 D 0 0 0 0 0
SP62
C1
OUT

Math Instructions

Add (ADD)

DS	Used
HPP	Used

Add is a 16-bit instruction that adds a BCD value in the accumulator with a BCD value in a V-memory location (Aaaa). (You cannot use a constant as the parameter in the box.) The result resides in the accumulator.

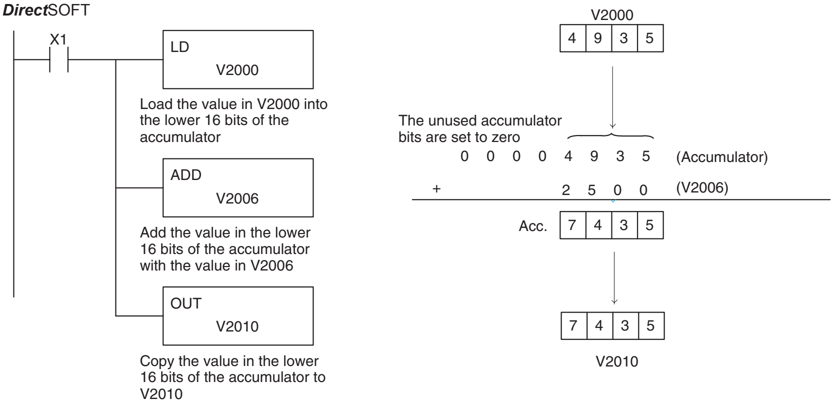


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32-bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator is added to the value in V2006 using the Add instruction. The value in the accumulator is copied to V2010 using the Out instruction.


*Direct*SOFT
X1
LD
V2000
Load the value in V2000 into the lower 16 bits of the accumulator
ADD
V2006
Add the value in the lower 16 bits of the accumulator with the value in V2006
OUT
V2010
Copy the value in the lower 16 bits of the accumulator to V2010
V2000
4 9 3 5
The unused accumulator bits are set to zero
0 0 0 0 4 9 3 5 (Accumulator)
+ 2 5 0 0 (V2006)
Acc. 7 4 3 5
7 4 3 5
V2010

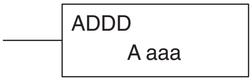
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT	
SHFT	A 0	D 3	D 3	→	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT	

Add Double (ADDD)

DS	Used
HPP	Used

Add Double is a 32-bit instruction that adds the BCD value in the accumulator with a BCD value (Aaaa), which is either two consecutive V-memory locations or an 8–digit (max.) BCD constant. The result resides in the accumulator.

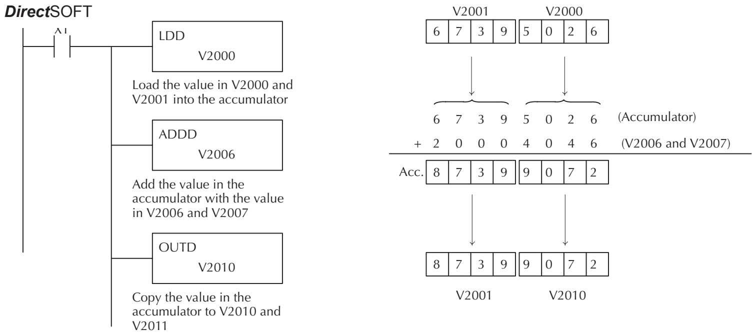


Operand Data Type		DL06Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–99999999

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32-bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is added with the value in V2006 and V2007 using the Add Double instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.


DirectSOFT
X1
LDD
V2000
Load the value in V2000 and V2001 into the accumulator
ADDD
V2006
Add the value in the accumulator with the value in V2006 and V2007
OUTD
V2010
Copy the value in the accumulator to V2010 and V2011
V2001
V2000
6 7 3 9 5 0 2 6
6 7 3 9 5 0 2 6
(Accumulator)
+ 2 0 0 0 4 0 4 6
(V2006 and V2007)
Acc. 8 7 3 9 9 0 7 2
8 7 3 9 9 0 7 2
V2001
V2010

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT	
SHFT	A 0	D 3	D 3	D 3	→	C 2	A 0	A 0	G 6	ENT
GX OUT	SHFT	D 3	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

Add Real (ADDR)

DS	Used
HPP	Used

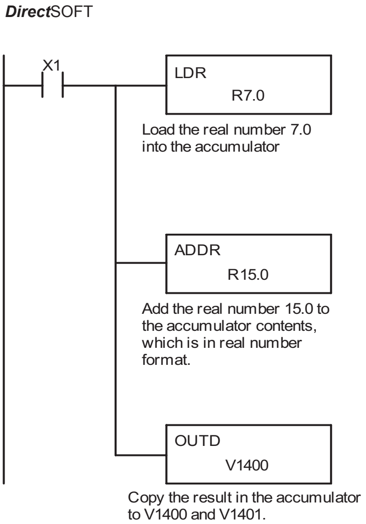
The Add Real instruction adds a real number in the accumulator with either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must be Real data type (IEEE floating point format).

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	R	-3.402823E+ 38 to + -3.402823E+ 38

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction results in a incorrect sign bit.
SP74	On anytime a floating point math operation results in an underflow error.

NOTE: *Status flags are valid only until another instruction uses the same flag.*

***NOTE**: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **Direct**SOFT for this feature.*



Load the real number 7.0 into the accumulator

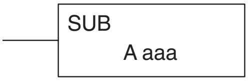
Add the real number 15.0 to the accumulator contents, which is in real number format.

Copy the result in the accumulator to V1400 and V1401.

Subtract (SUB)

DS	Used
HPP	Used

Subtract is a 16-bit instruction that subtracts the BCD value (Aaaa) in a V-memory location from the BCD value in the lower 16 bits of the accumulator. The result resides in the accumulator.

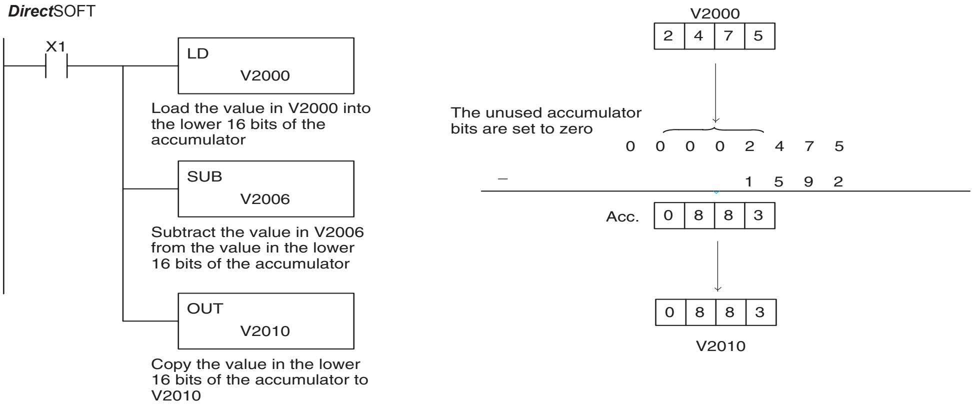


Operand Data Type		DL06Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow
SP65	On when the 32-bit subtraction instruction results in a borrow
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in V2006 is subtracted from the value in the accumulator using the Subtract instruction. The value in the accumulator is copied to V2010 using the Out instruction.


*Direct*SOFT
X1
LD
V2000
Load the value in V2000 into the lower 16 bits of the accumulator
SUB
V2006
Subtract the value in V2006 from the value in the lower 16 bits of the accumulator
OUT
V2010
Copy the value in the lower 16 bits of the accumulator to V2010
V2000
2 4 7 5
The unused accumulator bits are set to zero
0 0 0 0 2 4 7 5
– 1 5 9 2
Acc. 0 8 8 3
0 8 8 3
V2010

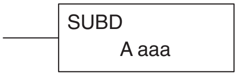
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT								
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT			
SHFT	S RST	U ISG	B 1	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT			

Subtract Double (SUBD)

DS	Used
HPP	Used

Subtract Double is a 32-bit instruction that subtracts the BCD value (Aaaa), which is either two consecutive V-memory locations or an 8-digit (max.) constant, from the BCD value in the accumulator.

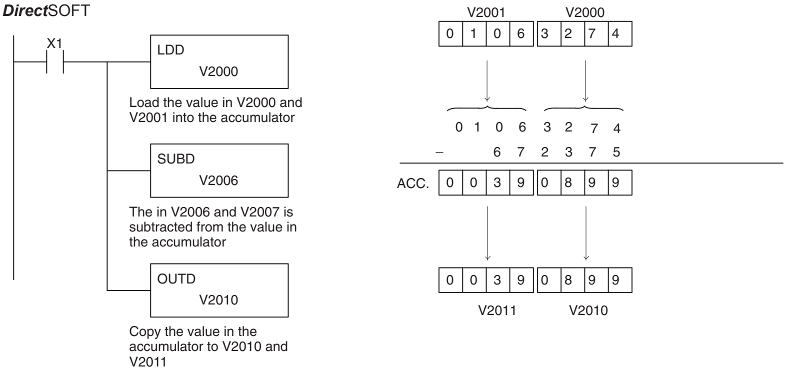


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–99999999

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16- bit subtraction instruction results in a borrow
SP65	On when the 32-bit subtraction instruction results in a borrow
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in V2006 and V2007 is subtracted from the value in the accumulator. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

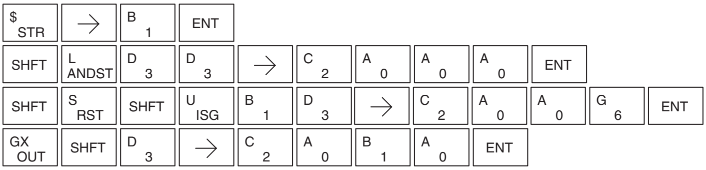


Load the value in V2000 and V2001 into the accumulator

The in V2006 and V2007 is subtracted from the value in the accumulator

Copy the value in the accumulator to V2010 and V2011

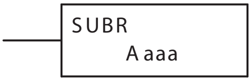
Handheld Programmer Keystrokes



Subtract Real (SUBR)

DS	Used
HPP	N/A

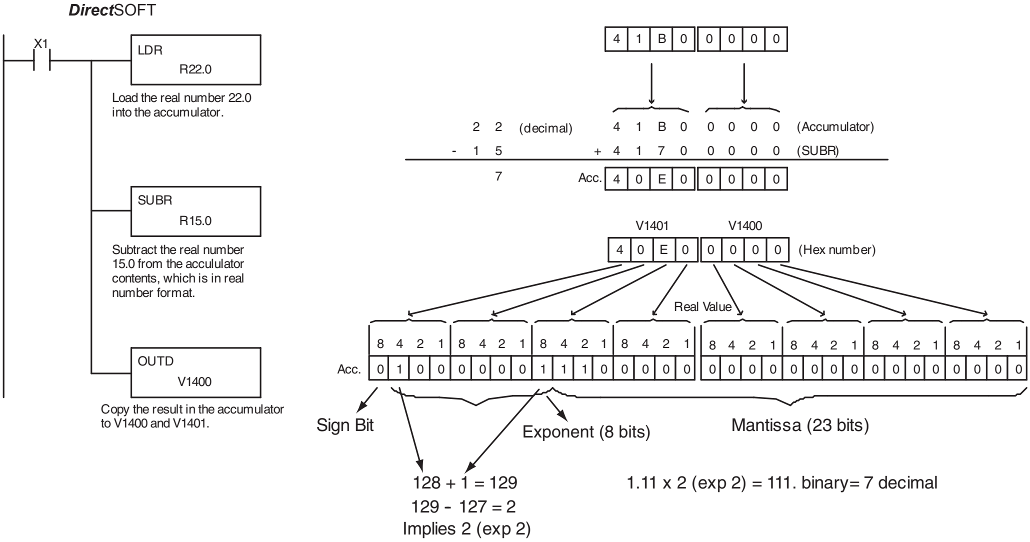
The Subtract Real is a 32-bit instruction that subtracts a real number, which is either two consecutive V-memory locations or a 32-bit constant, from a real number in the accumulator. The result is a 32-bit real number that resides in the accumulator. Both numbers must be Real data type (IEEE floating point format).



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	R	-3.402823E + 38 to+-3.402823E + 38

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction results in a incorrect sign bit.
SP74	On anytime a floating point math operation results in an underflow error.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*



Load the real number 22.0 into the accumulator.

Subtract the real number 15.0 from the accululator contents, which is in real number format.

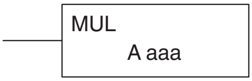
Copy the result in the accumulator to V1400 and V1401.

***NOTE**: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **Direct**SOFT for this feature*

Multiply (MUL)

DS	Used
HPP	Used

Multiply is a 16-bit instruction that multiplies the BCD value (Aaaa), which is either a V-memory location or a 4–digit (max.) constant, by the BCD value in the lower 16 bits of the accumulator The result can be up to 8 digits and resides in the accumulator.

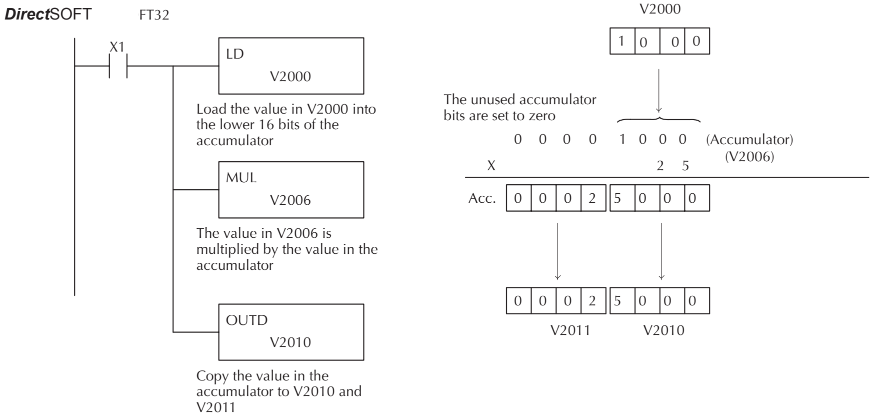


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–9999

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in V2006 is multiplied by the value in the accumulator. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



Load the value in V2000 into the lower 16 bits of the accumulator

The value in V2006 is multiplied by the value in the accumulator

Copy the value in the accumulator to V2010 and V2011

The unused accumulator bits are set to zero

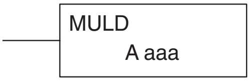
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	M ORST	U ISG	L ANDST	→	C 2	A 0	A 0	G 6	ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT

Multiply Double (MULD)

DS	Used
HPP	Used

Multiply Double is a 32-bit instruction that multiplies the 8-digit BCD value in the accumulator by the 8-digit BCD value in the two consecutive V-memory locations specified in the instruction. The lower 8 digits of the results reside in the accumulator. Upper digits of the result reside in the accumulator stack.

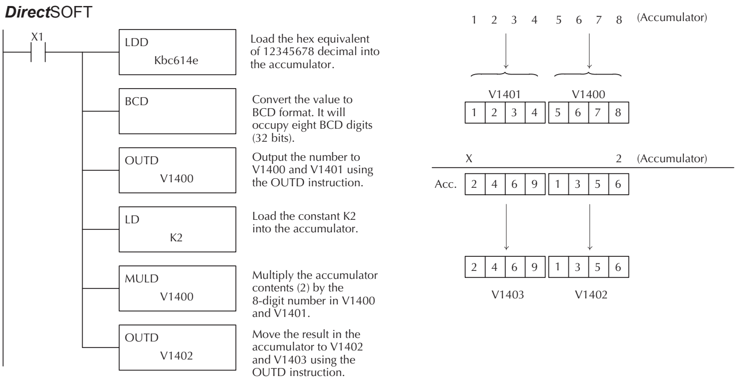


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the constant Kbc614e hex will be loaded into the accumulator. When converted to BCD the number is ”12345678”. That number is stored in V1400 and V1401. After loading the constant K2 into the accumulator, we multiply it times 12345678, which gives us 24691356.



Load the hex equivalent of 12345678 decimal into the accumulator.

Convert the value to BCD format. It will occupy eight BCD digits (32 bits).

Output the number to V1400 and V1401 using the OUTD instruction.

Load the constant K2 into the accumulator.

Multiply the accumulator contents (2) by the 8-digit number in V1400 and V1401.

Move the result in the accumulator to V1402 and V1403 using the OUTD instruction.

Handheld Programmer Keystrokes

$ STR | → | B 1 | ENT

SHFT | L ANDST | D 3 | D 3 | → | PREV | SHFT | B 1 | C 2 | SHFT | G 6 | B 1 | E 4 | SHFT | E 4 | ENT

SHFT | B 1 | C 2 | D 3 | ENT

GX OUT | SHFT | D 3 | → | B 1 | E 4 | A 0 | A 0 | ENT

SHFT | L ANDST | D 3 | → | PREV | C 2 | ENT

SHFT | M ORST | U ISG | L ANDST | D 3 | → | B 1 | E 4 | A 0 | A 0 | ENT

GX OUT | SHFT | D 3 | → | B 1 | E 4 | A 0 | C 2 | ENT

Multiply Real (MULR)

DS	Used
HPP	Used

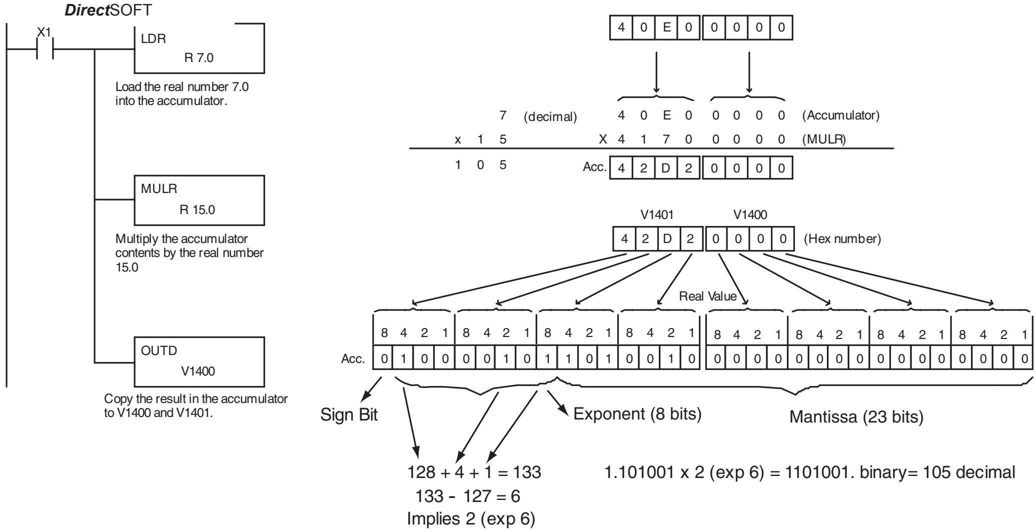
The Multiply Real instruction multiplies a real number in the accumulator with either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must be Real data type (IEEE floating point format).

```
MULR
    A aaa
```

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Real Constant	R	-3.402823E +38 to + -3.402823E +38

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP74	On anytime a floating point math operation results in an underflow error.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

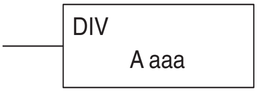

DirectSOFT
X1
LDR
R 7.0
Load the real number 7.0 into the accumulator.
MULR
R 15.0
Multiply the accumulator contents by the real number 15.0
OUTD
V1400
Copy the result in the accumulator to V1400 and V1401.
4 0 E 0 0 0 0 0
7 (decimal)
x 1 5
1 0 5
4 0 E 0 0 0 0 0 (Accumulator)
X 4 1 7 0 0 0 0 0 (MULR)
Acc. 4 2 D 2 0 0 0 0
V1401
V1400
4 2 D 2 0 0 0 0 (Hex number)
Real Value
8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1
Acc. 0 1 0 0 0 0 1 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Sign Bit
Exponent (8 bits)
Mantissa (23 bits)
128 + 4 + 1 = 133
133 - 127 = 6
Implies 2 (exp 6)
1.101001 x 2 (exp 6) = 1101001. binary= 105 decimal

***NOTE**: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **Direct**SOFT for this feature.*

Divide (DIV)

DS	Used
HPP	Used

Divide is a 16-bit instruction that divides the BCD value in the accumulator by a BCD value (Aaaa), which is either a V-memory location or a 4-digit (max.) constant. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

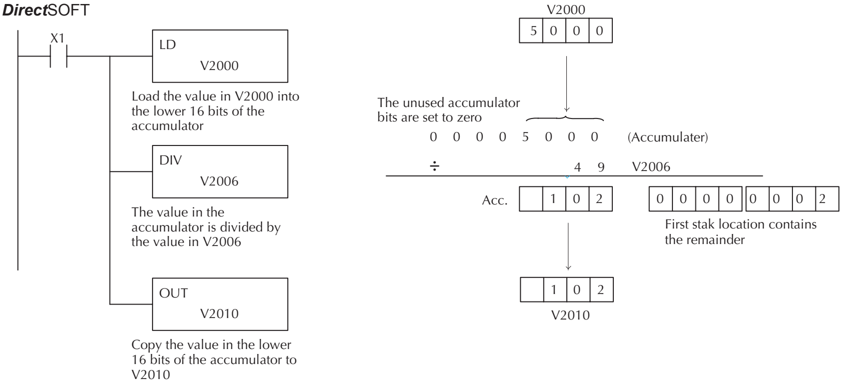


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–9999

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator will be divided by the value in V2006 using the Divide instruction. The value in the accumulator is copied to V2010 using the Out instruction.


DirectSOFT
X1
LD
V2000
Load the value in V2000 into the lower 16 bits of the accumulator
DIV
V2006
The value in the accumulator is divided by the value in V2006
OUT
V2010
Copy the value in the lower 16 bits of the accumulator to V2010
V2000
5 0 0 0
The unused accumulator bits are set to zero
0 0 0 0 5 0 0 0
(Accumulater)
÷
4 9
V2006
Acc.
1 0 2
0 0 0 0 0 0 0 2
First stak location contains the remainder
1 0 2
V2010

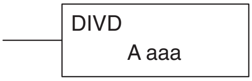
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT	
SHFT	D 3	I 8	V AND	→	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT	

Divide Double (DIVD)

DS	Used
HPP	Used

Divide Double is a 32-bit instruction that divides the BCD value in the accumulator by a BCD value (Aaaa), which must be obtained from two consecutive V-memory locations. (You cannot use a constant as the parameter in the box.) The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

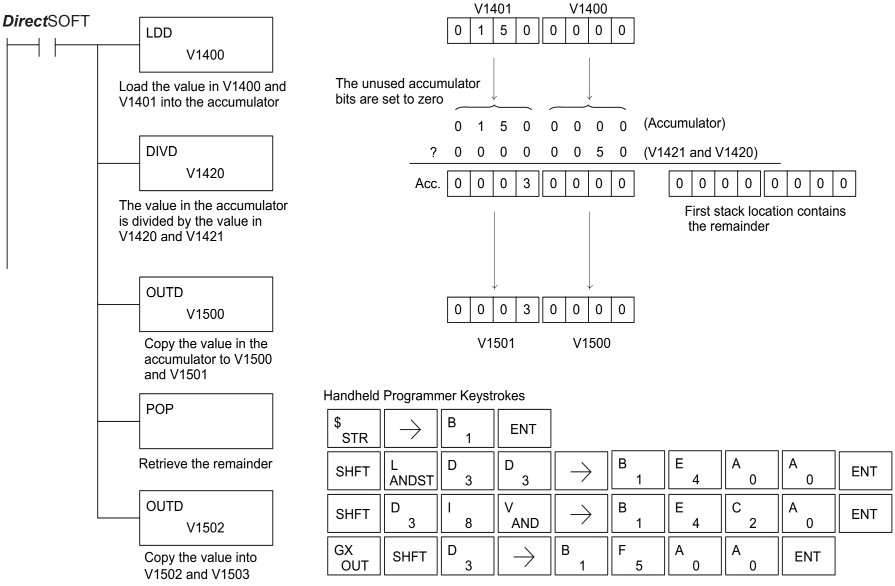


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is divided by the value in V1420 and V1421 using the Divide Double instruction. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.


DirectSOFT
LDD
V1400
Load the value in V1400 and V1401 into the accumulator
DIVD
V1420
The value in the accumulator is divided by the value in V1420 and V1421
OUTD
V1500
Copy the value in the accumulator to V1500 and V1501
POP
Retrieve the remainder
OUTD
V1502
Copy the value into V1502 and V1503
V1401
V1400
0 1 5 0 0 0 0 0
The unused accumulator bits are set to zero
0 1 5 0 0 0 0 0 (Accumulator)
? 0 0 0 0 0 0 5 0 (V1421 and V1420)
Acc. 0 0 0 3 0 0 0 0
0 0 0 0 0 0 0 0
First stack location contains the remainder
0 0 0 3 0 0 0 0
V1501
V1500

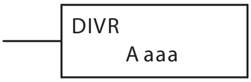
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	D 3	I 8	V AND	→	B 1	E 4	C 2	A 0	ENT
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Divide Real (DIVR)

DS	Used
HPP	N/A

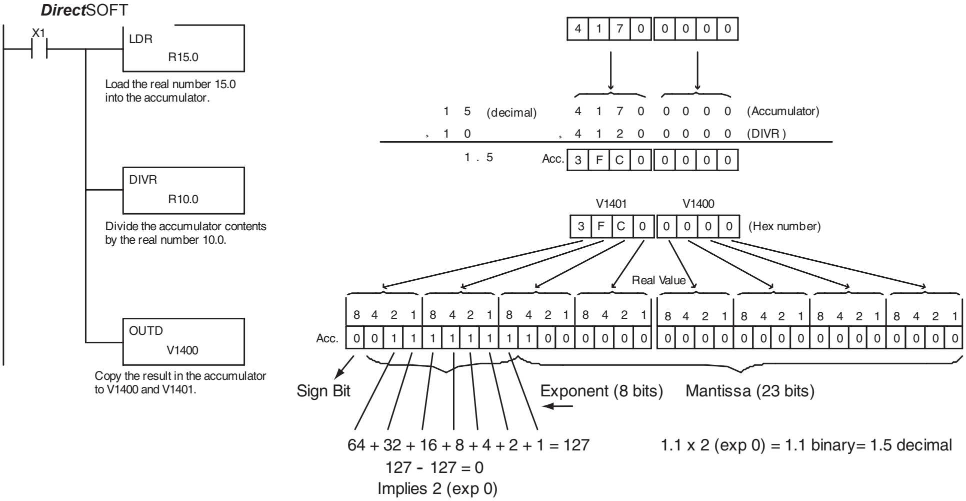
The Divide Real instruction divides a real number in the accumulator by either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must be Real data type (IEEE floating point format).



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Real Constant	R	-3.402823E + 38 to + -3.402823E + 38

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP74	On anytime a floating point math operation results in an underflow error.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

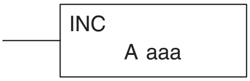

DirectSOFT
X1
LDR
R15.0
Load the real number 15.0 into the accumulator.
DIVR
R10.0
Divide the accumulator contents by the real number 10.0.
OUTD
V1400
Copy the result in the accumulator to V1400 and V1401.
4 1 7 0 0 0 0 0
1 5 (decimal) 4 1 7 0 0 0 0 0 (Accumulator)
÷ 1 0 ÷ 4 1 2 0 0 0 0 0 (DIVR)
1 . 5 Acc. 3 F C 0 0 0 0 0
V1401 V1400
3 F C 0 0 0 0 0 (Hex number)
Real Value
8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1
Acc. 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Sign Bit
Exponent (8 bits)
Mantissa (23 bits)
64 + 32 + 16 + 8 + 4 + 2 + 1 = 127
127 - 127 = 0
Implies 2 (exp 0)
1.1 x 2 (exp 0) = 1.1 binary= 1.5 decimal

***NOTE**: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **Direct**SOFT for this feature.*

Increment (INC)

DS	Used
HPP	Used

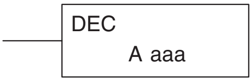
The Increment instruction increments a BCD value in a specified V-memory location by "1" each time the instruction is executed.



Decrement (DEC)

DS	Used
HPP	Used

The Decrement instruction decrements a BCD value in a specified V-memory location by "1" each time the instruction is executed.

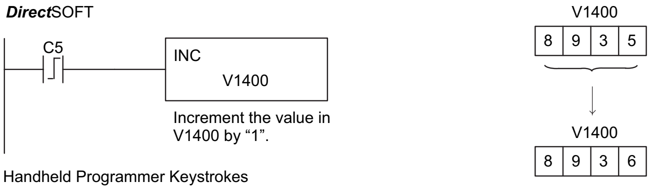


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

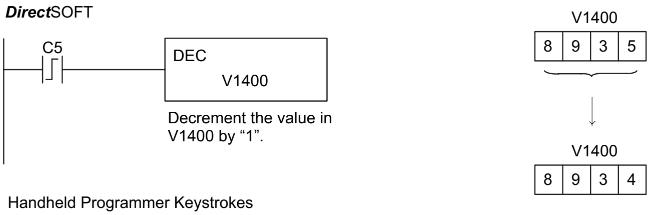
In the following increment example, when C5 makes an Off-to-On transition the value in V1400 increases by one.



Handheld Programmer Keystrokes

$ STR	→	NEXT	NEXT	NEXT	NEXT	F 5	ENT		
SHFT	I 8	N TMR	C 2	→	B 1	E 4	A 0	A 0	ENT

In the following decrement example, when C5 makes an Off-to-On transition the value in V1400 is decreased by one.



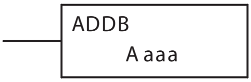
Handheld Programmer Keystrokes

$ STR	→	NEXT	NEXT	NEXT	NEXT	F 5	ENT		
SHFT	D 3	E 4	C 2	→	B 1	E 4	A 0	A 0	ENT

Add Binary (ADDB)

DS	Used
HPP	Used

Add Binary is a 16-bit instruction that adds the binary value in the lower 16 bits of the accumulator with a binary value (Aaaa), which is either a V-memory location or a 16-bit constant. The result can be up to 32 bits and resides in the accumulator.

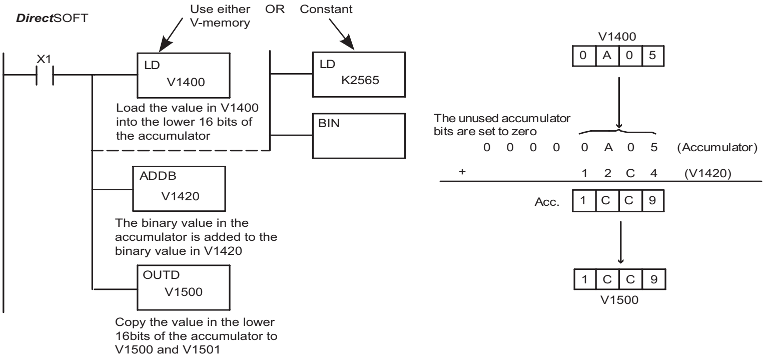


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF, h=65636

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32-bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in the accumulator will be added to the binary value in V1420 using the Add Binary instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Load the value in V1400 into the lower 16 bits of the accumulator

The binary value in the accumulator is added to the binary value in V1420

Copy the value in the lower 16bits of the accumulator to V1500 and V1501

The unused accumulator bits are set to zero

Handheld Programmer Keystrokes

STR → X(IN) 1 ENT
SHFT L D V 1 4 O O ENT
SHFT A D D B → V 1 4 2 O ENT
OUT SHFT D → V 1 5 O O ENT

Add Binary Double (ADDBD)

DS	Used
HPP	Used

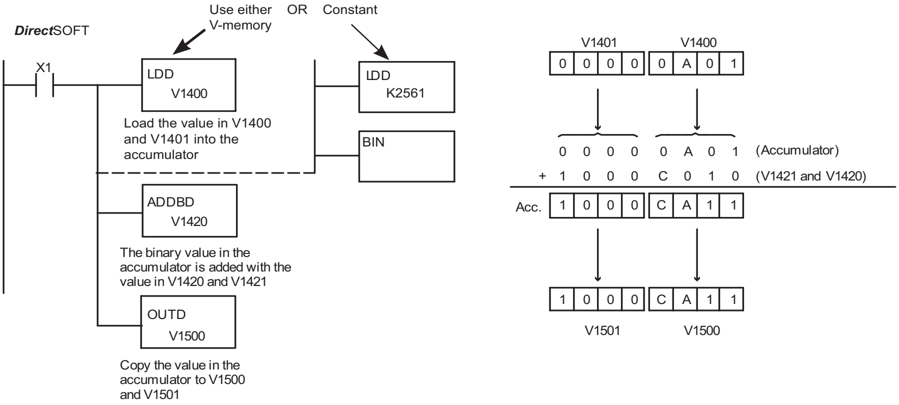
Add Binary Double is a 32-bit instruction that adds the binary value in the accumulator with the value (Aaaa), which is either two consecutive V-memory locations or an 8-digit (max.) binary constant. The result resides in the accumulator.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF FFFF

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32-bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The binary value in the accumulator is added with the binary value in V1420 and V1421 using the Add Binary Double instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.


DirectSOFT
Use either V-memory
OR
Constant
X1
LDD
V1400
Load the value in V1400 and V1401 into the accumulator
LDD
K2561
BIN
ADDBD
V1420
The binary value in the accumulator is added with the value in V1420 and V1421
OUTD
V1500
Copy the value in the accumulator to V1500 and V1501
V1401
V1400
0 0 0 0 0 A 0 1
0 0 0 0 0 A 0 1 (Accumulator)
+ 1 0 0 0 C 0 1 0 (V1421 and V1420)
Acc. 1 0 0 0 C A 1 1
1 0 0 0 C A 1 1
V1501
V1500

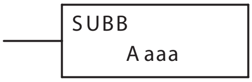
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT								
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT		
SHFT	A 0	D 3	D 3	B 1	D 3	→	B 1	E 4	C 2	A 0	ENT
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT			

Subtract Binary (SUBB)

DS	Used
HPP	Used

Subtract Binary is a 16-bit instruction that subtracts the binary value (Aaaa), which is either a V-memory location or a 4-digit (max.) binary constant, from the binary value in the accumulator. The result resides in the accumulator.

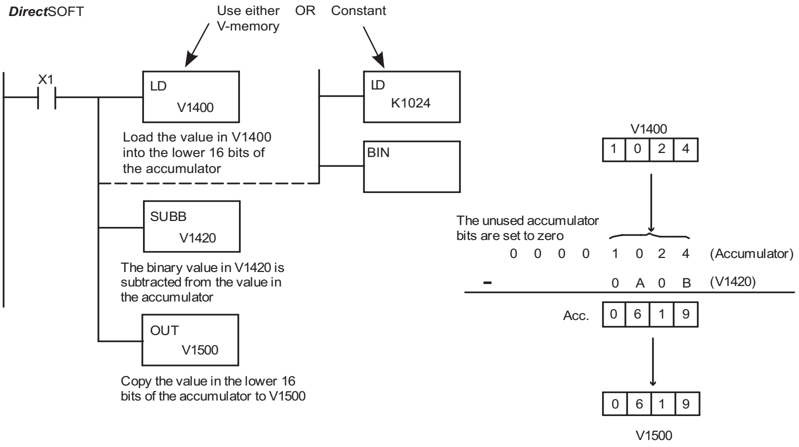


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF, h=65636

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow.
SP65	On when the 32-bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in V1420 is subtracted from the binary value in the accumulator using the Subtract Binary instruction. The value in the accumulator is copied to V1500 using the Out instruction.


***Direct*SOFT**
Use either V-memory
OR
Constant
X1
LD V1400
LD K1024
BIN
Load the value in V1400 into the lower 16 bits of the accumulator
SUBB V1420
The binary value in V1420 is subtracted from the value in the accumulator
OUT V1500
Copy the value in the lower 16 bits of the accumulator to V1500
V1400
1 0 2 4
The unused accumulator bits are set to zero
0 0 0 0 1 0 2 4 (Accumulator)
– 0 A 0 B (V1420)
Acc. 0 6 1 9
0 6 1 9
V1500

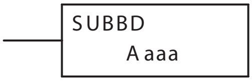
Handheld Programmer Keystrokes

STR	→	X(IN)	1	ENT					
SHFT	L	D	V	1	4	0	0	ENT	
SHFT	S	SHFT	U	B	B	→			
V	1	4	2	0	ENT				
OUT	SHFT	D	→	V	1	5	0	0	ENT

Subtract Binary Double (SUBBD)

DS	Used
HPP	Used

Subtract Binary Double is a 32-bit instruction that subtracts the binary value (Aaaa), which is either two consecutive V-memory locations or an 8-digit (max.) binary constant, from the binary value in the accumulator. The result resides in the accumulator.

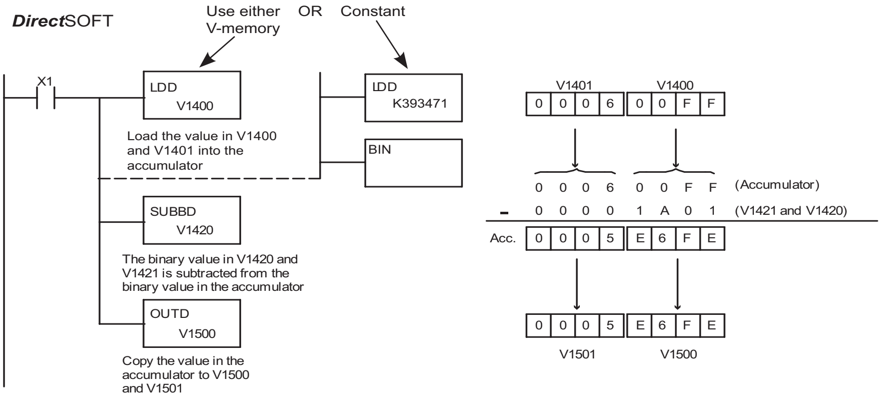


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF FFFF

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow.
SP65	On when the 32-bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The binary value in V1420 and V1421 is subtracted from the binary value in the accumulator using the Subtract Binary Double instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.


DirectSOFT
Use either V-memory
OR
Constant
X1
LDD V1400
Load the value in V1400 and V1401 into the accumulator
LDD K393471
BIN
SUBBD V1420
The binary value in V1420 and V1421 is subtracted from the binary value in the accumulator
OUTD V1500
Copy the value in the accumulator to V1500 and V1501
V1401
V1400
0 0 0 6 0 0 F F
0 0 0 6 0 0 F F (Accumulator)
- 0 0 0 0 1 A 0 1 (V1421 and V1420)
Acc. 0 0 0 5 E 6 F E
0 0 0 5 E 6 F E
V1501
V1500

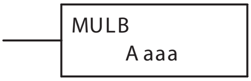
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT									
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT			
SHFT	S RST	SHFT	U ISG	B 1	B 1	D 3	→	B 1	E 4	C 2	A 0	ENT
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT				

Multiply Binary (MULB)

DS	Used
HPP	Used

Multiply Binary is a 16-bit instruction that multiplies the binary value (Aaaa), which is either a V-memory location or a 4-digit (max.) binary constant, by the binary value in the accumulator. The result can be up to 32 bits and resides in the accumulator.

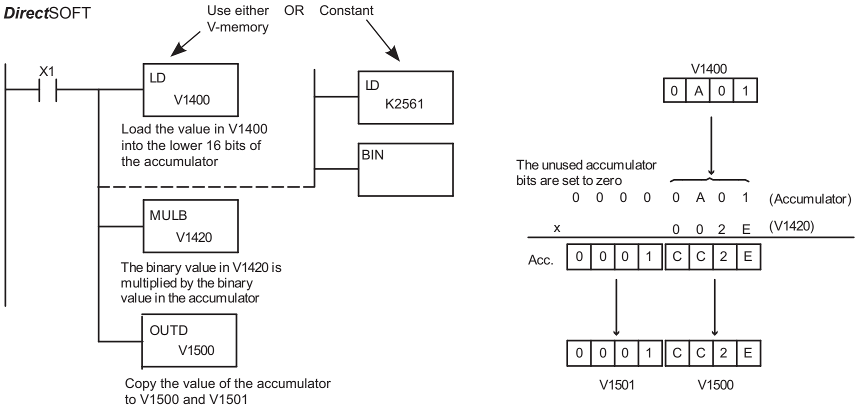


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in V1420 is multiplied by the binary value in the accumulator using the Multiply Binary instruction. The value in the accumulator is copied to V1500 using the Out instruction.


*Direct*SOFT
Use either V-memory
OR
Constant
X1
LD
V1400
Load the value in V1400 into the lower 16 bits of the accumulator
LD
K2561
BIN
MULB
V1420
The binary value in V1420 is multiplied by the binary value in the accumulator
OUTD
V1500
Copy the value of the accumulator to V1500 and V1501
V1400
0 A 0 1
The unused accumulator bits are set to zero
0 0 0 0 0 A 0 1
(Accumulator)
x
0 0 2 E
(V1420)
Acc.
0 0 0 1 C C 2 E
0 0 0 1 C C 2 E
V1501
V1500

Handheld Programmer Keystrokes

STR → X 1 ENT

SHFT L D V 1 4 0 0 ENT

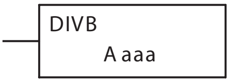
SHFT M U L B → V 1 4 2 0 ENT

OUT SHFT D → V 1 5 0 0 ENT

Divide Binary (DIVB)

DS	Used
HPP	Used

Divide Binary is a 16-bit instruction that divides the binary value in the accumulator by a binary value (Aaaa), which is either a V-memory location or a 16-bit (max.) binary constant. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

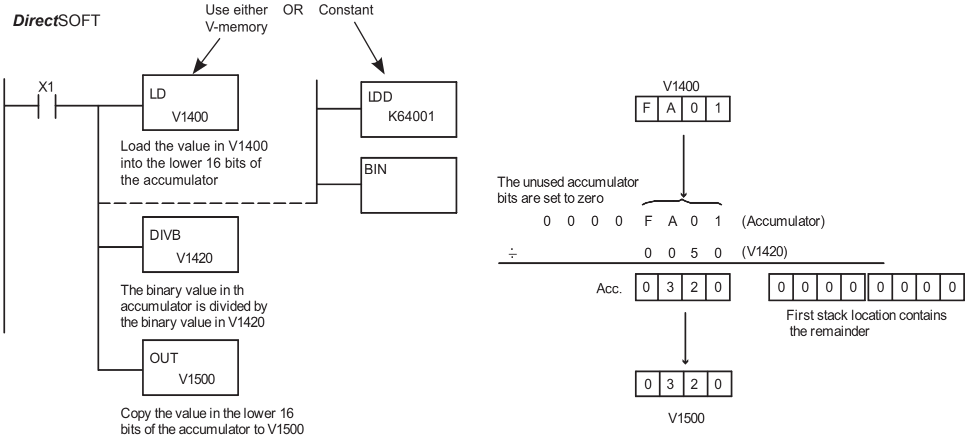


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

NOTE*: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in the accumulator is divided by the binary value in V1420 using the Divide Binary instruction. The value in the accumulator is copied to V1500 using the Out instruction.


*Direct*SOFT
Use either V-memory
OR
Constant
X1
LD V1400
LDD K64001
BIN
Load the value in V1400 into the lower 16 bits of the accumulator
DIVB V1420
The binary value in th accumulator is divided by the binary value in V1420
OUT V1500
Copy the value in the lower 16 bits of the accumulator to V1500
V1400
F A 0 1
The unused accumulator bits are set to zero
0 0 0 0 F A 0 1 (Accumulator)
÷ 0 0 5 0 (V1420)
Acc. 0 3 2 0
0 0 0 0 0 0 0 0
First stack location contains the remainder
0 3 2 0
V1500

Handheld Programmer Keystrokes

STR → X 1 ENT

SHFT L D V 1 4 0 0 ENT

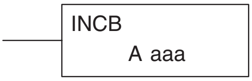
SHFT D I V B → V 1 4 2 0 ENT

OUT SHFT D → V 1 5 0 0 ENT

Increment Binary (INCB)

DS	Used
HPP	Used

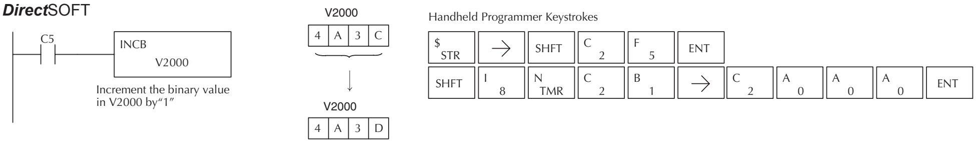
The Increment Binary instruction increments a binary value in a specified V-memory location by "1" each time the instruction is executed.



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.

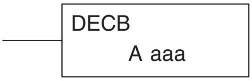
In the following example when C5 is on, the binary value in V2000 is increased by 1.



Decrement Binary (DECB)

DS	Used
HPP	Used

The Decrement Binary instruction decrements a binary value in a specified V-memory location by "1" each time the instruction is executed.

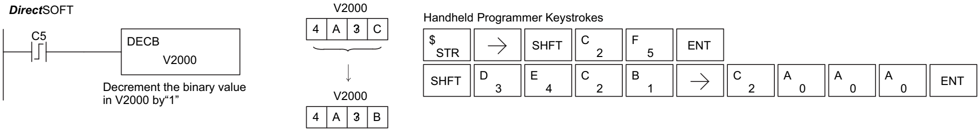


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

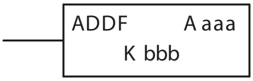
In the following example when C5 is on, the value in V2000 is decreased by 1.



Add Formatted (ADDF)

DS	Used
HPP	Used

Add Formatted is a 32-bit instruction that adds the BCD value in the accumulator with the BCD value (Aaaa) which is a range of discrete bits. The specified range (Kbbb) can be 1 to 32 consecutive bits. The result resides in the accumulator.

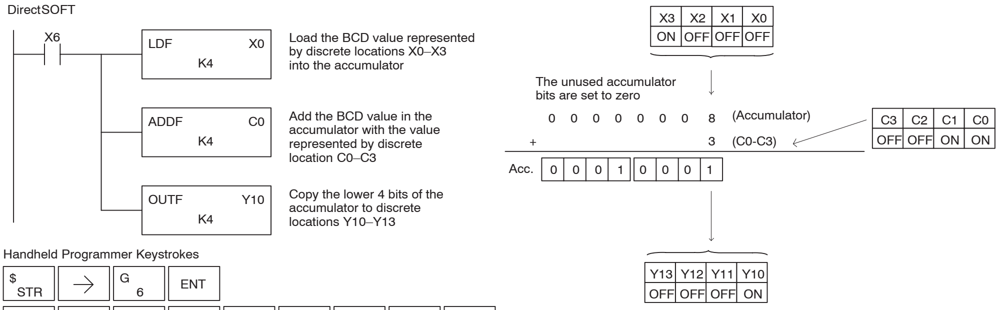


Operand Data Type		DL06 Range	
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0-137 320-717	—
Global I/O	GX	0-3777	—
Constant	K	—	1–32

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X6 is on, the BCD value formed by discrete locations X0–X3 is loaded into the accumulator using the LDF instruction. The BCD value formed by discrete locations C0–C3 is added to the value in the accumulator using the ADDF instruction. The value in the lower four bits of the accumulator is copied to Y10–Y13 using the OUTF instruction.


DirectSOFT
X6
LDF X0
K4
Load the BCD value represented by discrete locations X0–X3 into the accumulator
ADDF C0
K4
Add the BCD value in the accumulator with the value represented by discrete location C0–C3
OUTF Y10
K4
Copy the lower 4 bits of the accumulator to discrete locations Y10–Y13
X3 X2 X1 X0
ON OFF OFF OFF
The unused accumulator bits are set to zero
0 0 0 0 0 0 0 8 (Accumulator)
+ 3 (C0-C3)
C3 C2 C1 C0
OFF OFF ON ON
Acc. 0 0 0 1 0 0 0 1
Y13 Y12 Y11 Y10
OFF OFF OFF ON

Handheld Programmer Keystrokes

$ STR → G 6 ENT

SHFT L ANDST D 3 F 5 → A 0 → E 4 ENT

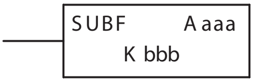
SHFT A 0 D 3 D 3 F 5 → NEXT NEXT NEXT NEXT A 0 → E 4 ENT

GX OUT SHFT F 5 → B 1 A 0 → E 4 ENT

Subtract Formatted (SUBF)

DS	Used
HPP	Used

Subtract Formatted is a 32-bit instruction that subtracts the BCD value (Aaaa), which is a range of discrete bits, from the BCD value in the accumulator. The specified range (Kbbb) can be 1 to 32 consecutive bits. The result resides in the accumulator.

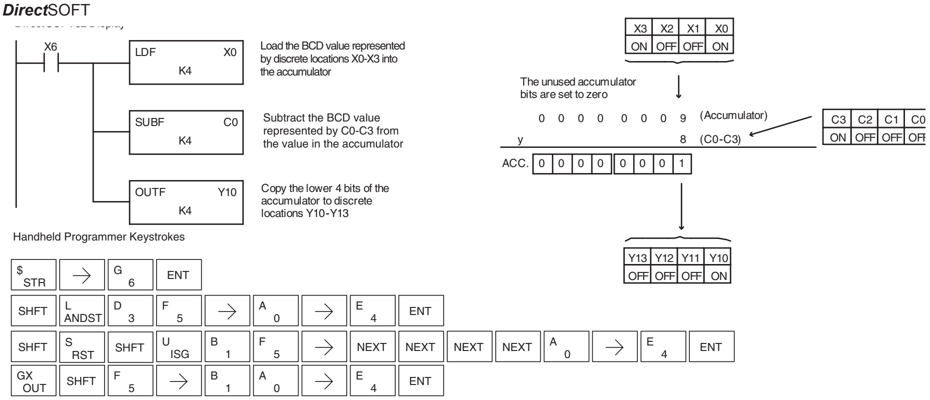


Operand Data Type		DL06 Range	
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0-137 320-717	—
Global I/O	GX	0-3777	—
Constant	K	—	1–32

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow
SP70	On any time the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X6 is on, the BCD value formed by discrete locations X0–X3 is loaded into the accumulator using the LDF instruction. The BCD value formed by discrete location C0–C3 is subtracted from the BCD value in the accumulator using the SUBF instruction. The value in the lower four bits of the accumulator is copied to Y10–Y13 using the OUTF instruction.


DirectSOFT
X6
LDF X0
K4
Load the BCD value represented by discrete locations X0-X3 into the accumulator
SUBF C0
K4
Subtract the BCD value represented by C0-C3 from the value in the accumulator
OUTF Y10
K4
Copy the lower 4 bits of the accumulator to discrete locations Y10-Y13
X3 X2 X1 X0
ON OFF OFF ON
The unused accumulator bits are set to zero
0 0 0 0 0 0 0 9 (Accumulator)
y 8 (C0-C3)
C3 C2 C1 C0
ON OFF OFF OFF
ACC. 0 0 0 0 0 0 0 1
Y13 Y12 Y11 Y10
OFF OFF OFF ON
Handheld Programmer Keystrokes

$ STR → G 6 ENT

SHFT L ANDST D 3 F 5 → A 0 → E 4 ENT

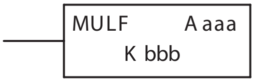
SHFT S RST SHFT U ISG B 1 F 5 → NEXT NEXT NEXT NEXT A 0 → E 4 ENT

GX OUT SHFT F 5 → B 1 A 0 → E 4 ENT

Multiply Formatted (MULF)

DS	Used
HPP	Used

Multiply Formatted is a 16-bit instruction that multiplies the BCD value in the accumulator by the BCD value (Aaaa) which is a range of discrete bits. The specified range (Kbbb) can be 1 to 16 consecutive bits. The result resides in the accumulator.

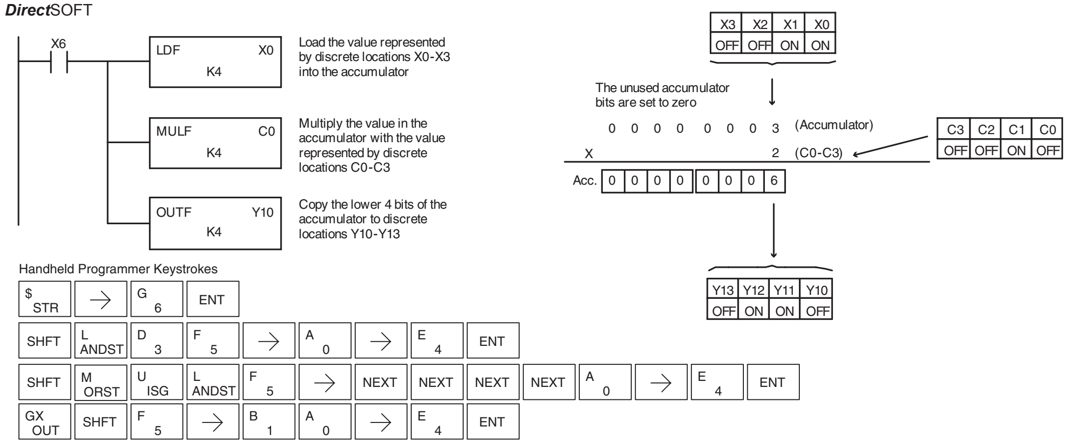


Operand Data Type		DL06 Range	
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0-137 320-717	—
Global I/O	GX	0-3777	—
Constant	K	—	1–16

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X6 is on, the value formed by discrete locations X0–X3 is loaded into the accumulator using the Load Formatted instruction. The value formed by discrete locations C0–C3 is multiplied by the value in the accumulator using the Multiply Formatted instruction. The value in the lower four bits of the accumulator is copied to Y10–Y13 using the Out Formatted instruction.



Load the value represented by discrete locations X0-X3 into the accumulator

Multiply the value in the accumulator with the value represented by discrete locations C0-C3

Copy the lower 4 bits of the accumulator to discrete locations Y10-Y13

The unused accumulator bits are set to zero

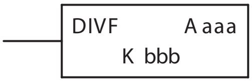
Handheld Programmer Keystrokes

$ STR	→	G 6	ENT										
SHFT	L ANDST	D 3	F 5	→	A 0	→	E 4	ENT					
SHFT	M ORST	U ISG	L ANDST	F 5	→	NEXT	NEXT	NEXT	NEXT	A 0	→	E 4	ENT
GX OUT	SHFT	F 5	→	B 1	A 0	→	E 4	ENT					

Divide Formatted (DIVF)

DS	Used
HPP	Used

Divide Formatted is a 16-bit instruction that divides the BCD value in the accumulator by the BCD value (Aaaa), a range of discrete bits. The specified range (Kbbb) can be 1 to 16 consecutive bits. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location..

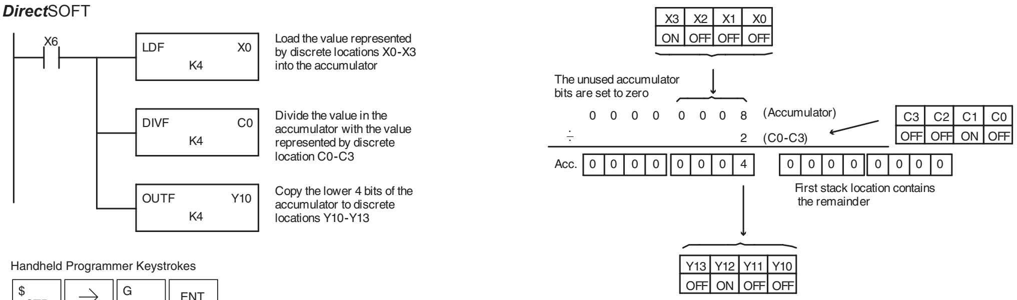


Operand Data Type		DL06 Range	
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	P	0-137 320-717	—
Global I/O	X	0-3777	—
Constant	K	—	1–16

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X6 is on, the value formed by discrete locations X0–X3 is loaded into the accumulator using the Load Formatted instruction. The value in the accumulator is divided by the value formed by discrete location C0–C3 using the Divide Formatted instruction. The value in the lower four bits of the accumulator is copied to Y10–Y13 using the Out Formatted instruction.


*Direct*SOFT
X6
LDF X0
K4
Load the value represented by discrete locations X0-X3 into the accumulator
DIVF C0
K4
Divide the value in the accumulator with the value represented by discrete location C0-C3
OUTF Y10
K4
Copy the lower 4 bits of the accumulator to discrete locations Y10-Y13
X3 X2 X1 X0
ON OFF OFF OFF
The unused accumulator bits are set to zero
0 0 0 0 0 0 0 8 (Accumulator)
÷ 2 (C0-C3)
C3 C2 C1 C0
OFF OFF ON OFF
Acc. 0 0 0 0 0 0 0 4
0 0 0 0 0 0 0 0
First stack location contains the remainder
Y13 Y12 Y11 Y10
OFF ON OFF OFF

Handheld Programmer Keystrokes

$ STR | → | G 6 | ENT

SHFT | L ANDST | D 3 | F 5 | → | A 0 | → | E 4 | ENT

SHFT | D 3 | I 8 | V AND | F 5 | → | NEXT | NEXT | NEXT | NEXT | A 0 | → | E 4 | ENT

GX OUT | SHFT | F 5 | → | B 1 | A 0 | → | E 4 | ENT

Add Top of Stack (ADDS)

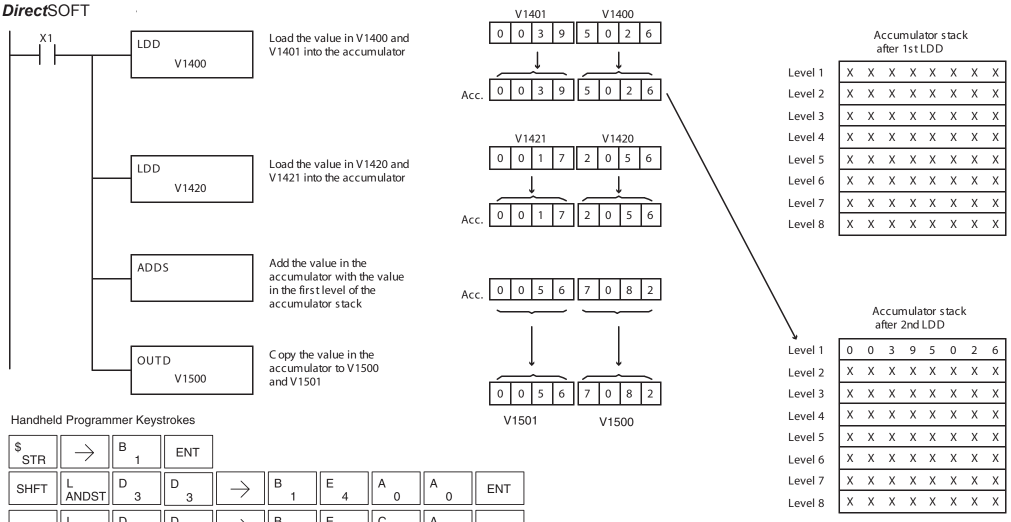
DS	Used
HPP	Used

Add Top of Stack is a 32-bit instruction that adds the BCD value in the accumulator with the BCD value in the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The value in the first level of the accumulator stack is added with the value in the accumulator using the Add Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Handheld Programmer Keystrokes

$ STR | → | B 1 | ENT

SHFT | L ANDST | D 3 | D 3 | → | B 1 | E 4 | A 0 | A 0 | ENT

SHFT | L ANDST | D 3 | D 3 | → | B 1 | E 4 | C 2 | A 0 | ENT

SHFT | A 0 | D 3 | D 3 | S RST | ENT

GX OUT | SHFT | D 3 | → | B 1 | F 5 | A 0 | A 0 | ENT

Subtract Top of Stack (SUBS)

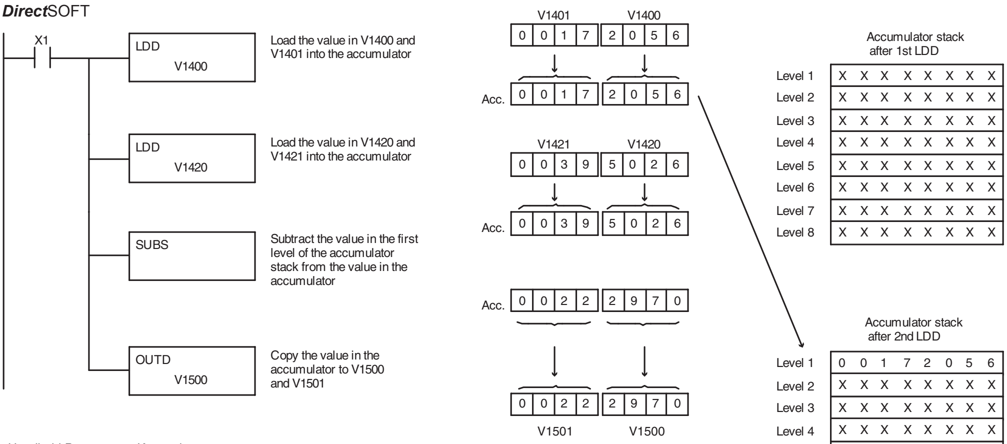
DS	Used
HPP	Used

Subtract Top of Stack is a 32-bit instruction that subtracts the BCD value in the first level of the accumulator stack from the BCD value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded into the accumulator onto the accumulator stack. The BCD value in the first level of the accumulator stack is subtracted from the BCD value in the accumulator using the Subtract Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.


*Direct*SOFT
X1
LDD V1400
Load the value in V1400 and V1401 into the accumulator
LDD V1420
Load the value in V1420 and V1421 into the accumulator
SUBS
Subtract the value in the first level of the accumulator stack from the value in the accumulator
OUTD V1500
Copy the value in the accumulator to V1500 and V1501
V1401
V1400
0 0 1 7 2 0 5 6
Acc.
0 0 1 7 2 0 5 6
V1421
V1420
0 0 3 9 5 0 2 6
Acc.
0 0 3 9 5 0 2 6
Acc.
0 0 2 2 2 9 7 0
0 0 2 2 2 9 7 0
V1501
V1500

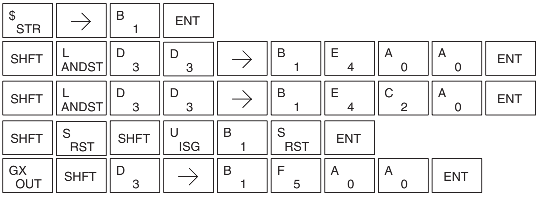
Accumulator stack after 1st LDD

Level 1	X	X	X	X	X	X	X	X
Level 2	X	X	X	X	X	X	X	X
Level 3	X	X	X	X	X	X	X	X
Level 4	X	X	X	X	X	X	X	X
Level 5	X	X	X	X	X	X	X	X
Level 6	X	X	X	X	X	X	X	X
Level 7	X	X	X	X	X	X	X	X
Level 8	X	X	X	X	X	X	X	X

Accumulator stack after 2nd LDD

Level 1	0	0	1	7	2	0	5	6
Level 2	X	X	X	X	X	X	X	X
Level 3	X	X	X	X	X	X	X	X
Level 4	X	X	X	X	X	X	X	X
Level 5	X	X	X	X	X	X	X	X
Level 6	X	X	X	X	X	X	X	X
Level 7	X	X	X	X	X	X	X	X
Level 8	X	X	X	X	X	X	X	X

Handheld Programmer Keystrokes


$ STR | → | B 1 | ENT
SHFT | L ANDST | D 3 | D 3 | → | B 1 | E 4 | A 0 | A 0 | ENT
SHFT | L ANDST | D 3 | D 3 | → | B 1 | E 4 | C 2 | A 0 | ENT
SHFT | S RST | SHFT | U ISG | B 1 | S RST | ENT
GX OUT | SHFT | D 3 | → | B 1 | F 5 | A 0 | A 0 | ENT

Multiply Top of Stack (MULS)

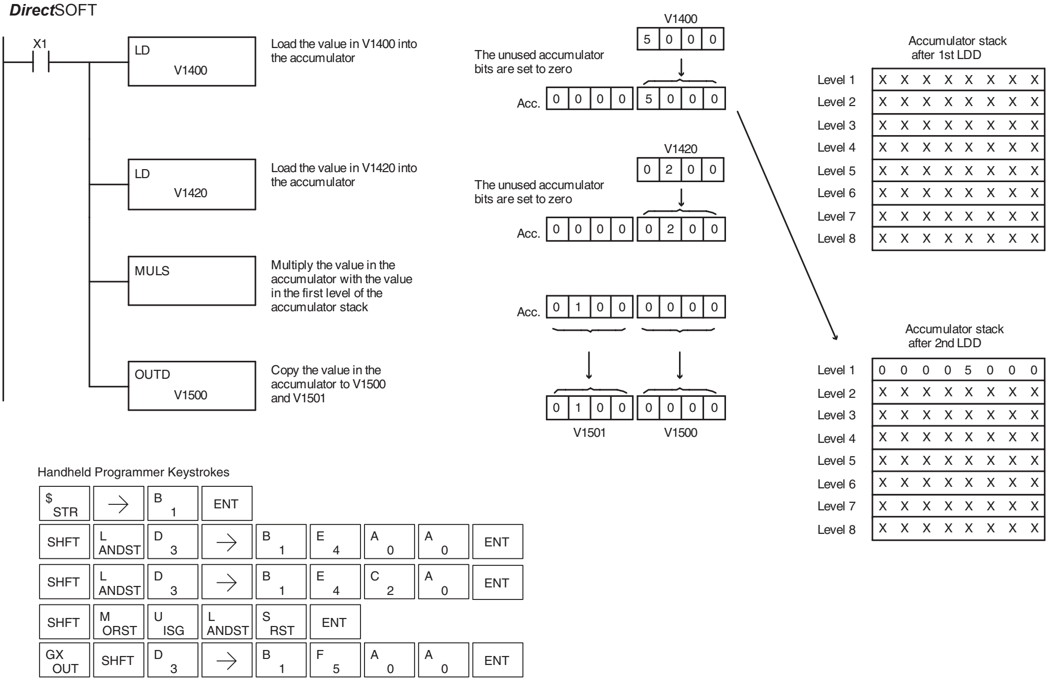
DS	Used
HPP	Used

Multiply Top of Stack is a 16-bit instruction that multiplies a 4-digit BCD value in the first level of the accumulator stack by a 4-digit BCD value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in V1420 is loaded into the accumulator using the Load instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The BCD value in the first level of the accumulator stack is multiplied by the BCD value in the accumulator using the Multiply Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.


*Direct*SOFT
X1
LD
V1400
Load the value in V1400 into the accumulator
LD
V1420
Load the value in V1420 into the accumulator
MULS
Multiply the value in the accumulator with the value in the first level of the accumulator stack
OUTD
V1500
Copy the value in the accumulator to V1500 and V1501
V1400
5 0 0 0
The unused accumulator bits are set to zero
Acc. 0 0 0 0 5 0 0 0
V1420
0 2 0 0
The unused accumulator bits are set to zero
Acc. 0 0 0 0 0 2 0 0
Acc. 0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
V1501
V1500

Accumulator stack after 1st LDD

Level 1	X	X	X	X	X	X	X	X
Level 2	X	X	X	X	X	X	X	X
Level 3	X	X	X	X	X	X	X	X
Level 4	X	X	X	X	X	X	X	X
Level 5	X	X	X	X	X	X	X	X
Level 6	X	X	X	X	X	X	X	X
Level 7	X	X	X	X	X	X	X	X
Level 8	X	X	X	X	X	X	X	X

Accumulator stack after 2nd LDD

Level 1	0	0	0	0	5	0	0	0
Level 2	X	X	X	X	X	X	X	X
Level 3	X	X	X	X	X	X	X	X
Level 4	X	X	X	X	X	X	X	X
Level 5	X	X	X	X	X	X	X	X
Level 6	X	X	X	X	X	X	X	X
Level 7	X	X	X	X	X	X	X	X
Level 8	X	X	X	X	X	X	X	X

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	L ANDST	D 3	→	B 1	E 4	C 2	A 0	ENT
SHFT	M ORST	U ISG	L ANDST	S RST	ENT			
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT

Divide by Top of Stack (DIVS)

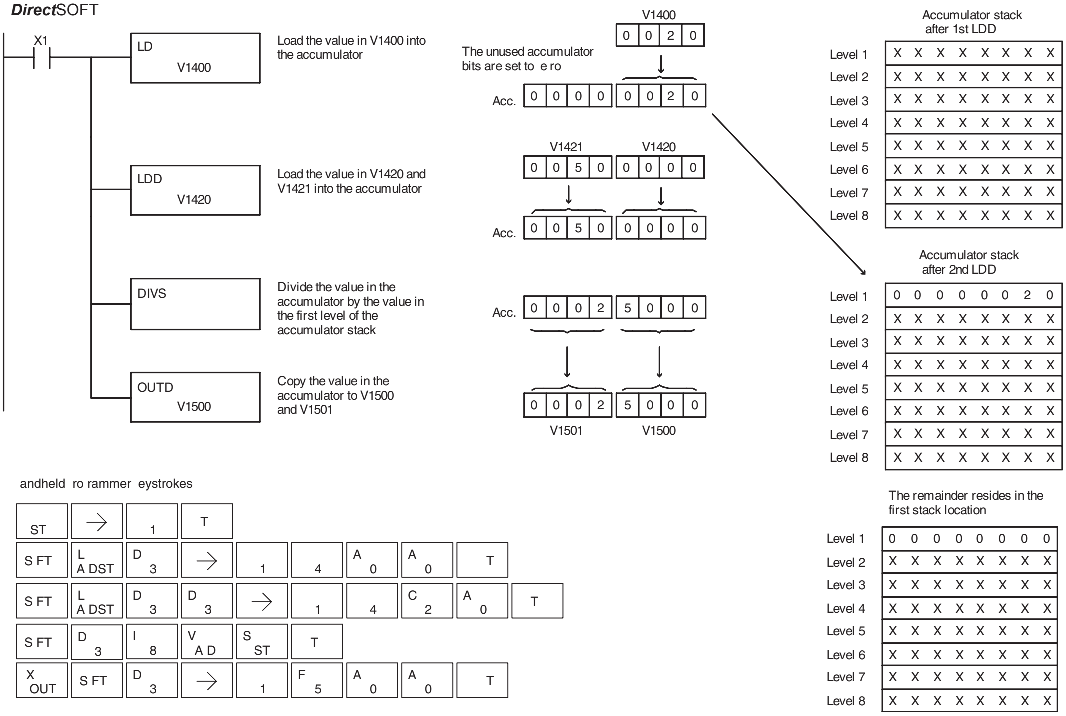
DS	Used
HPP	Used

Divide Top of Stack is a 32-bit instruction that divides the 8-digit BCD value in the accumulator by a 4-digit BCD value in the first level of the accumulator stack. The result resides in the accumulator and the remainder resides in the first level of the accumulator stack.

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the Load instruction loads the value in V1400 into the accumulator. The value in V1420 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The BCD value in the accumulator is divided by the BCD value in the first level of the accumulator stack using the Divide Stack instruction. The Out Double instruction copies the value in the accumulator to V1500 and V1501.



andheld ro rammer eystrokes

ST	→	1	T						
S FT	L A DST	D 3	→	1	4	A 0	A 0	T	
S FT	L A DST	D 3	D 3	→	1	4	C 2	A 0	T
S FT	D 3	I 8	V A D	S ST	T				
X OUT	S FT	D 3	→	1	F 5	A 0	A 0	T	

Add Binary Top of Stack (ADDBS)

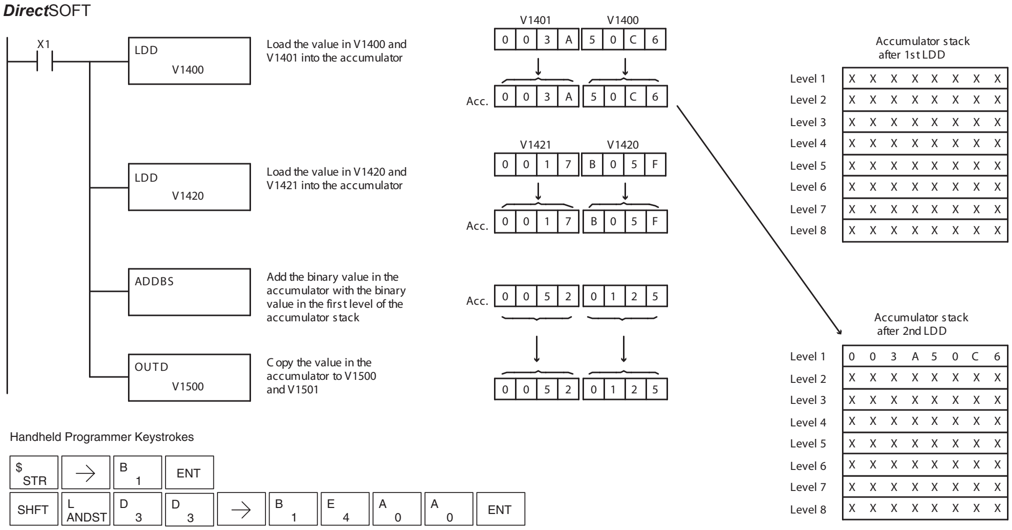
DS	Used
HPP	Used

Add Binary Top of Stack instruction is a 32-bit instruction that adds the binary value in the accumulator with the binary value in the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The binary value in the first level of the accumulator stack is added with the binary value in the accumulator using the Add Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



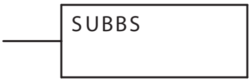
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	C 2	A 0	ENT
SHFT	A 0	D 3	D 3	B 1	S RST	ENT			
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Subtract Binary Top of Stack (SUBBS)

DS	Used
HPP	Used

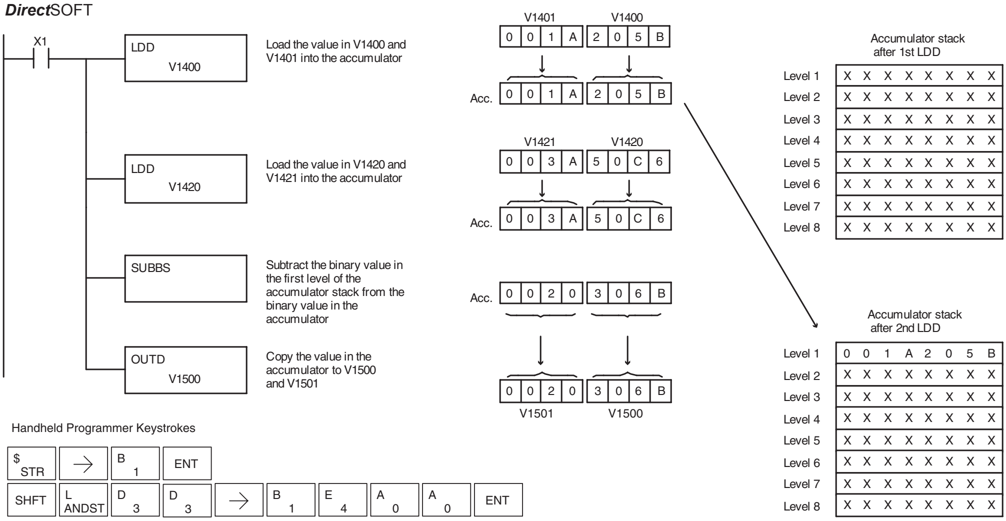
Subtract Binary Top of Stack is a 32-bit instruction that subtracts the binary value in the first level of the accumulator stack from the binary value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack locations are moved up one level



Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow.
SP65	On when the 32-bit subtraction instruction results in a borrow.
SP70	On any time the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The binary value in the first level of the accumulator stack is subtracted from the binary value in the accumulator using the Subtract Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	C 2	A 0	ENT
SHFT	S RST	SHFT	U ISG	B 1	B 1	S RST	ENT		
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Multiply Binary Top of Stack (MULBS)

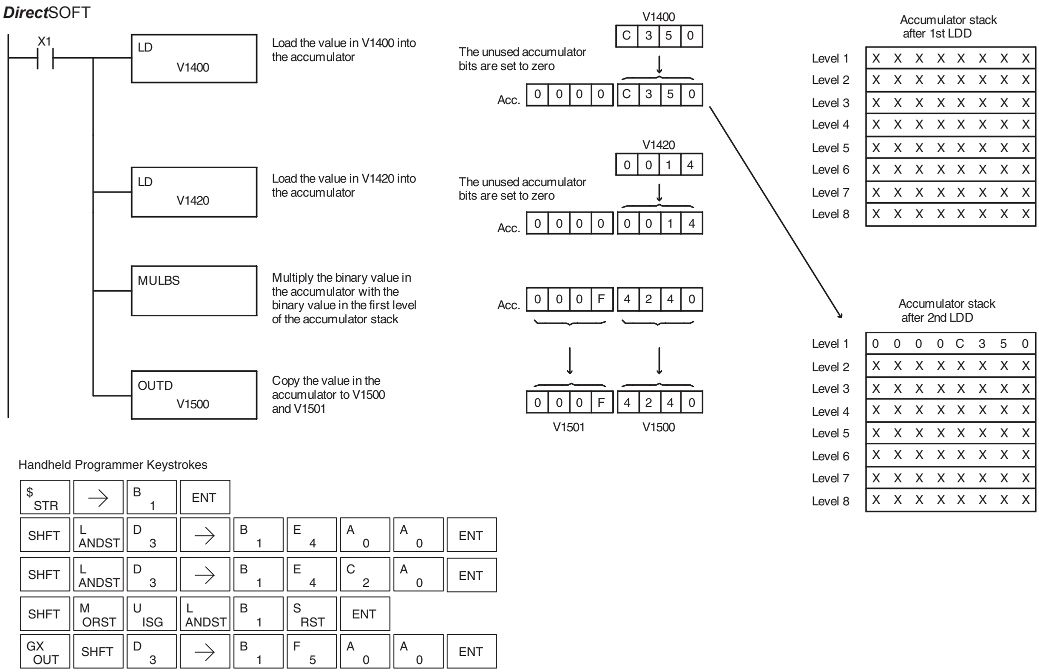
DS	Used
HPP	Used

Multiply Binary Top of Stack is a 16-bit instruction that multiplies the 16-bit binary value in the first level of the accumulator stack by the 16-bit binary value in the accumulator. The result resides in the accumulator and can be 32 bits (8 digits max.). The value in the first level of the accumulator stack is removed and all stack locations are moved up one level

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the Load instruction moves the value in V1400 into the accumulator. The value in V1420 is loaded into the accumulator using the Load instruction, pushing the value previously loaded in the accumulator onto the stack. The binary value in the accumulator stack's first level is multiplied by the binary value in the accumulator using the Multiply Binary Stack instruction. The Out Double instruction copies the value in the accumulator to V1500 and V1501.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	L ANDST	D 3	→	B 1	E 4	C 2	A 0	ENT
SHFT	M ORST	U ISG	L ANDST	B 1	S RST	ENT		
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT

Divide Binary by Top OF Stack (DIVBS)

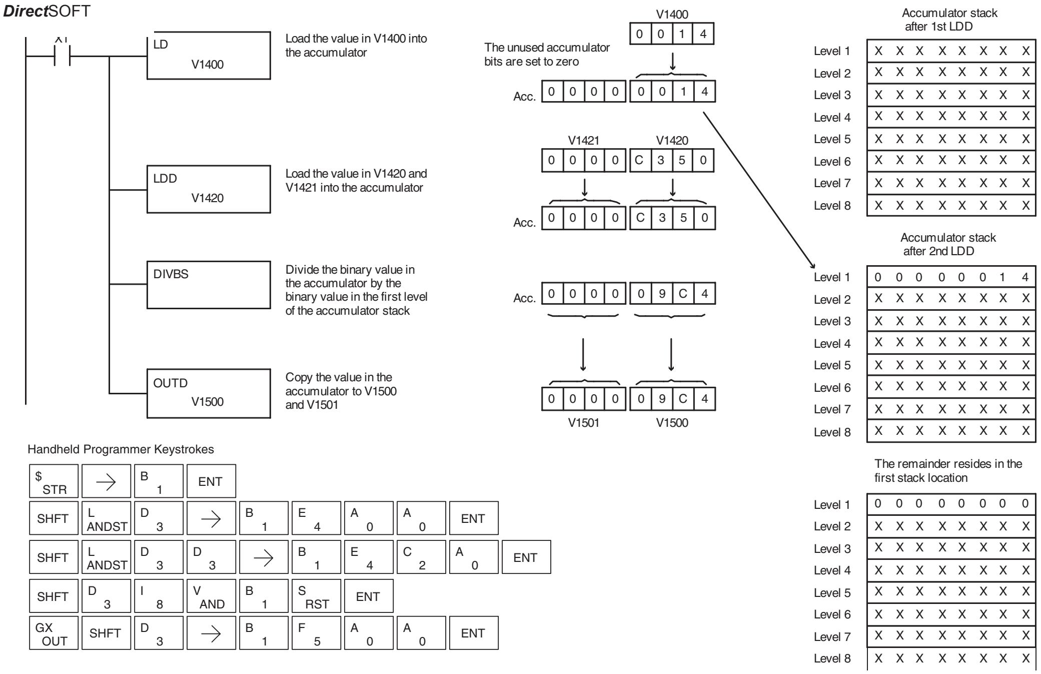
DS	Used
HPP	Used

Divide Binary Top of Stack is a 32-bit instruction that divides the 32-bit binary value in the accumulator by the 16-bit binary value in the first level of the accumulator stack. The result resides in the accumulator and the remainder resides in the first level of the accumulator stack.

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction also, pushing the value previously loaded in the accumulator onto the accumulator stack. The binary value in the accumulator is divided by the binary value in the first level of the accumulator stack using the Divide Binary Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Transcendental Functions

The DL06 CPU features special numerical functions to complement its real number capability. The transcendental functions include the trigonometric sine, cosine, and tangent, and also their inverses (arc sine, arc cosine, and arc tangent). The square root function is also grouped with these other functions.

The transcendental math instructions operate on a real number in the accumulator (it cannot be BCD or binary). The real number result resides in the accumulator. The square root function operates on the full range of positive real numbers. The sine, cosine and tangent functions require numbers expressed in radians. You can work with angles expressed in degrees by first converting them to radians with the Radian (RADR) instruction, then performing the trig function. All transcendental functions utilize the following flag bits.

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP72	On anytime the value in the accumulator is an invalid floating point number
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a real number instruction is executed and a non-real number encountered.

Sine Real (SINR)

DS	Used
HPP	N/A

The Sine Real instruction takes the sine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).

Cosine Real (COSR)

DS	Used
HPP	N/A

The Cosine Real instruction takes the cosine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format)..

Tangent Real (TANR)

DS	Used
HPP	N/A

The Tangent Real instruction takes the tangent of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).

Arc Sine Real (ASINR)

DS	Used
HPP	N/A

The Arc Sine Real instruction takes the inverse sine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).

Arc Cosine Real (ACOSR)

DS	Used
HPP	N/A

The Arc Cosine Real instruction takes the inverse cosine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).

Arc Tangent Real (ATANR)

DS	Used
HPP	N/A

The Arc Tangent Real instruction takes the inverse tangent of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).

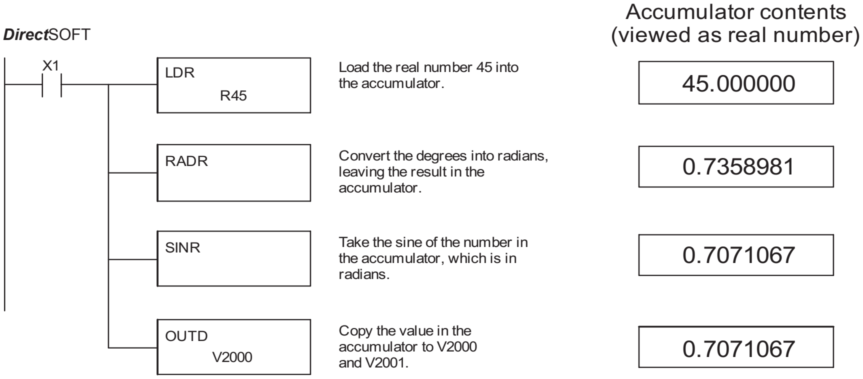
Square Root Real (SQRTR)

DS	Used
HPP	N/A

The Square Root Real instruction takes the square root of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).

***NOTE**: The square root function can be useful in several situations. However, if you are trying to do the square-root extract function for an orifice flow meter measurement, as the PV to a PID loop, note that the PID loop already has the square-root extract function built in.*

The following example takes the **sine** of 45 degrees. Since these transcendental functions operate only on real numbers, we do an LDR (load real) 45. The trig functions operate only in radians, so we must convert the degrees to radians by using the RADR command. After using the SINR (Sine Real) instruction, we use an OUTD (Out Double) instruction to move the result from the accumulator to V-memory. The result is 32-bits wide, requiring the Out Double to move it.



Load the real number 45 into the accumulator.

Convert the degrees into radians, leaving the result in the accumulator.

Take the sine of the number in the accumulator, which is in radians.

Copy the value in the accumulator to V2000 and V2001.

***NOTE**: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **Direct**SOFT for entering real numbers, using the LDR (Load Real) instruction.*

Bit Operation Instructions

Sum (SUM)

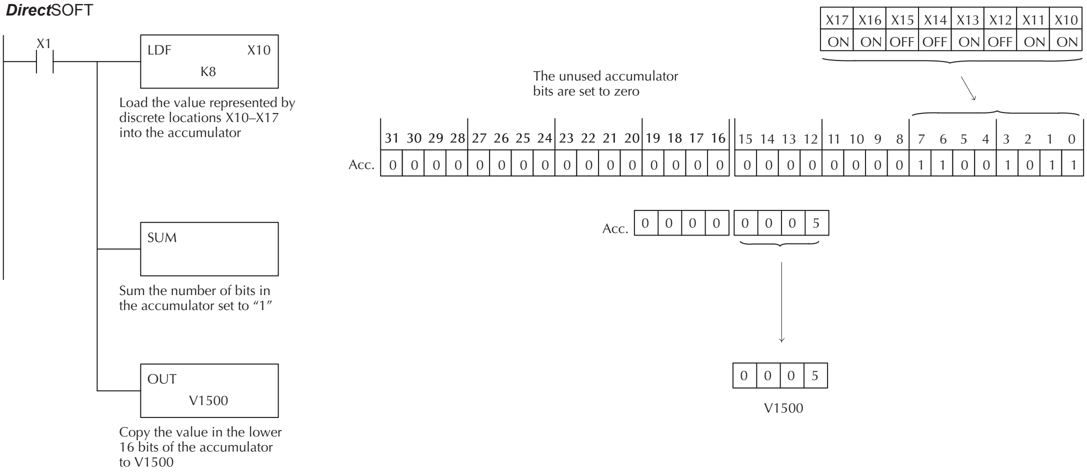
DS	Used
HPP	Used

The Sum instruction counts number of bits that are set to “1” in the accumulator. The HEX result resides in the accumulator.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.

In the following example, when X1 is on, the value formed by discrete locations X10–X17 is loaded into the accumulator using the Load Formatted instruction. The number of bits in the accumulator set to “1” is counted using the Sum instruction. The value in the accumulator is copied to V1500 using the Out instruction.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*


*Direct*SOFT
X1
LDF X10
K8
Load the value represented by discrete locations X10–X17 into the accumulator
SUM
Sum the number of bits in the accumulator set to “1”
OUT
V1500
Copy the value in the lower 16 bits of the accumulator to V1500
X17 X16 X15 X14 X13 X12 X11 X10
ON ON OFF OFF ON OFF ON ON
The unused accumulator bits are set to zero
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 1
Acc. 0 0 0 0 0 0 0 5
0 0 0 5
V1500

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	F 5	→	B 1	A 0	→	I 8	ENT
SHFT	S RST	SHFT	U ISG	M ORST	→	ENT			
GX OUT	→	PREV	PREV	PREV	B 1	F 5	A 0	A 0	ENT

Shift Left (SHFL)

DS	Used
HPP	Used

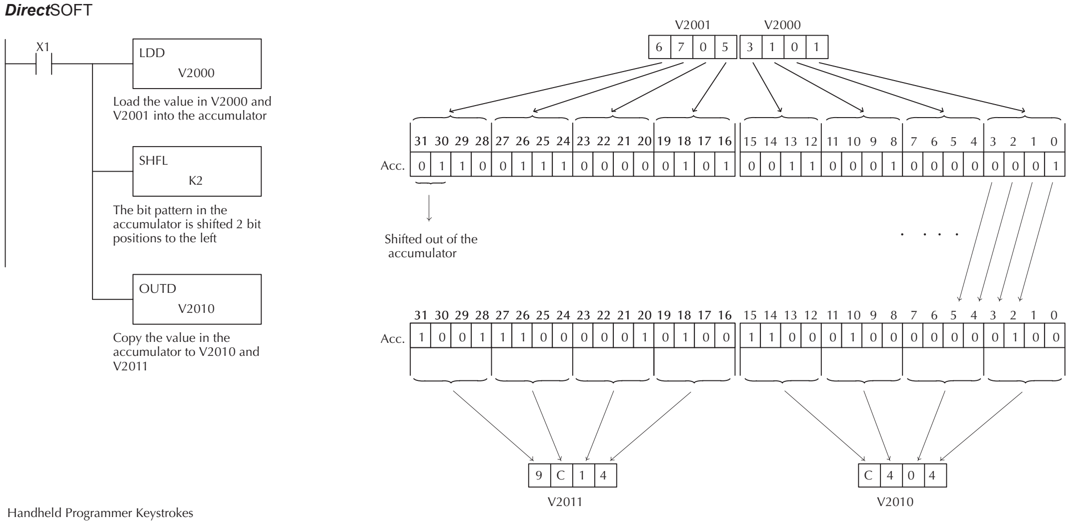
Shift Left is a 32-bit instruction that shifts the bits in the accumulator a specified number (Aaaa) of places to the left. The vacant positions are filled with zeros and the bits shifted out of the accumulator are discarded.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1-32

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 2 bits to the left using the Shift Left instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

NOTE*: Status flags are valid only until another instruction uses the same flag.*


*Direct*SOFT
X1
LDD
V2000
Load the value in V2000 and V2001 into the accumulator
SHFL
K2
The bit pattern in the accumulator is shifted 2 bit positions to the left
OUTD
V2010
Copy the value in the accumulator to V2010 and V2011
V2001
V2000
6 7 0 5
3 1 0 1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 1 1 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1
Shifted out of the accumulator
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 1 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0
9 C 1 4
V2011
C 4 0 4
V2010

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	S RST	SHFT	H 7	F 5	L ANDST	→	C 2	ENT	
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

Shift Right (SHFR)

DS	Used
HPP	Used

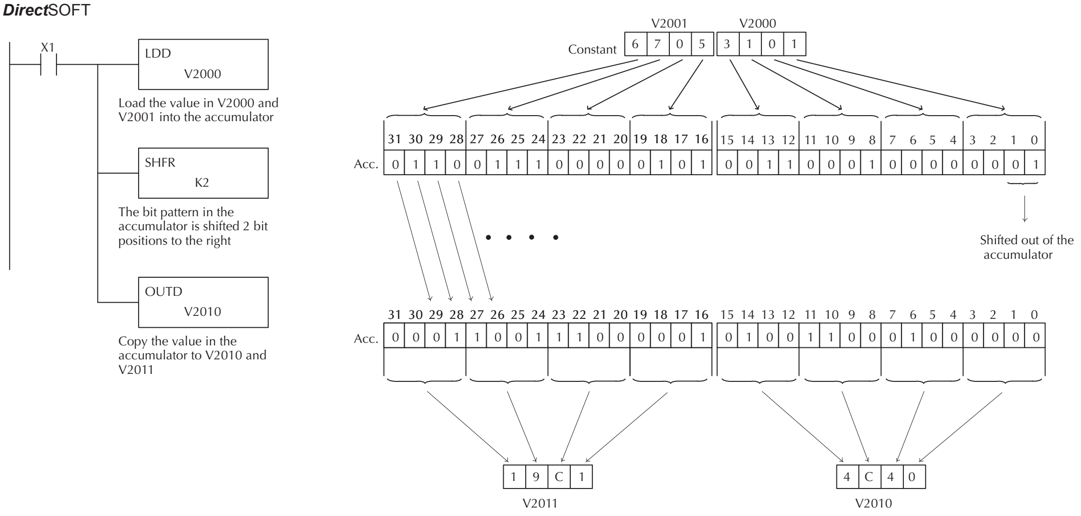
Shift Right is a 32-bit instruction that shifts the bits in the accumulator a specified number (Aaaa) of places to the right. The vacant positions are filled with zeros and the bits shifted out of the accumulator are lost.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1-32

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 2 bits to the right using the Shift Right instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	S RST	SHFT	H 7	F 5	R ORN	→	C 2	ENT	
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

Rotate Left (ROTL)

DS	Used
HPP	Used

Rotate Left is a 32-bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the left.

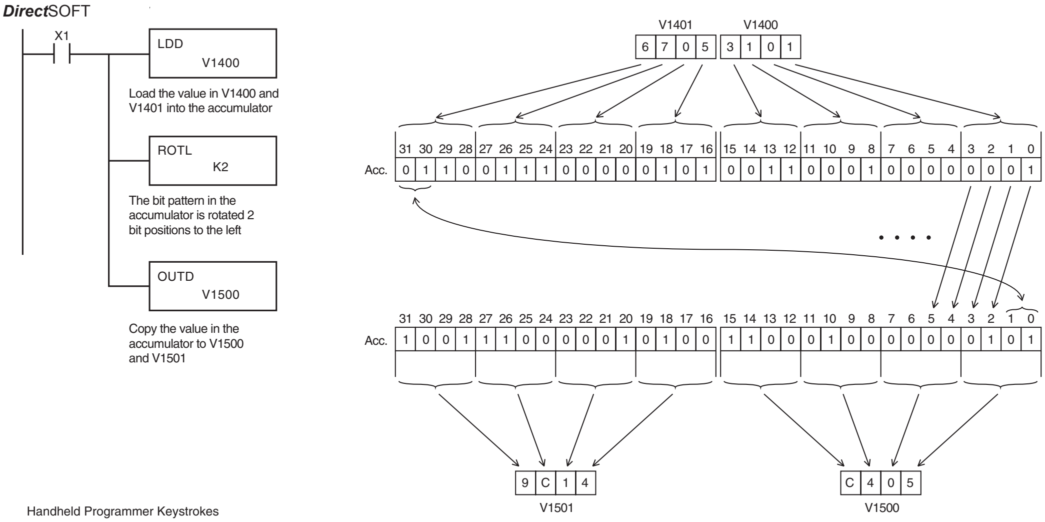
```
ROTL
    A aaa
```

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1-32

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is rotated 2 bit positions to the left using the Rotate Left instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*



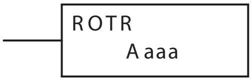
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	R ORN	O INST#	T MLR	L ANDST	→	C 2	ENT		
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Rotate Right (ROTR)

DS	Used
HPP	Used

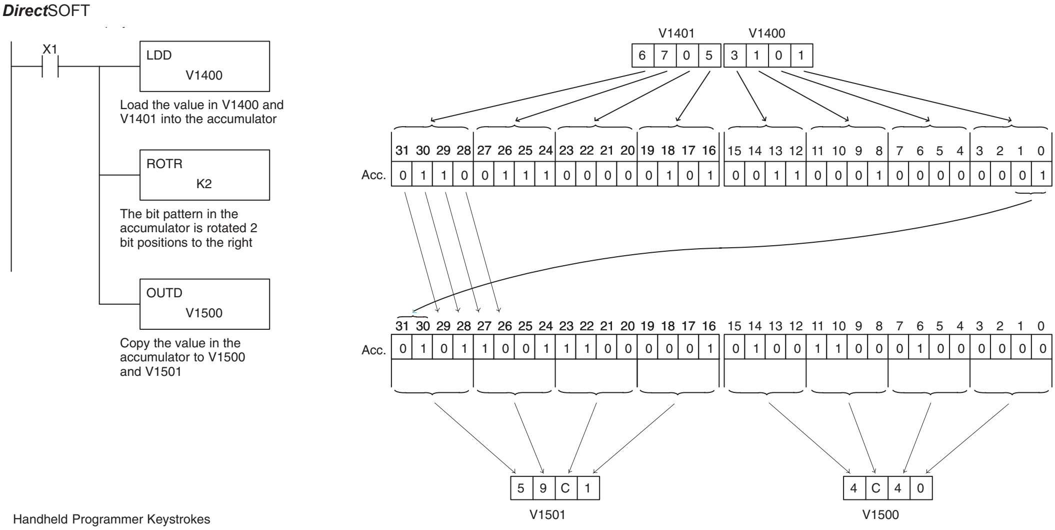
Rotate Right is a 32-bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the right.



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1-32

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is rotated 2 bit positions to the right using the Rotate Right instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Load the value in V1400 and V1401 into the accumulator

The bit pattern in the accumulator is rotated 2 bit positions to the right

Copy the value in the accumulator to V1500 and V1501

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	R ORN	O INST#	T MLR	R ORN	→	C 2	ENT		
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Encode (ENCO)

DS	Used
HPP	Used

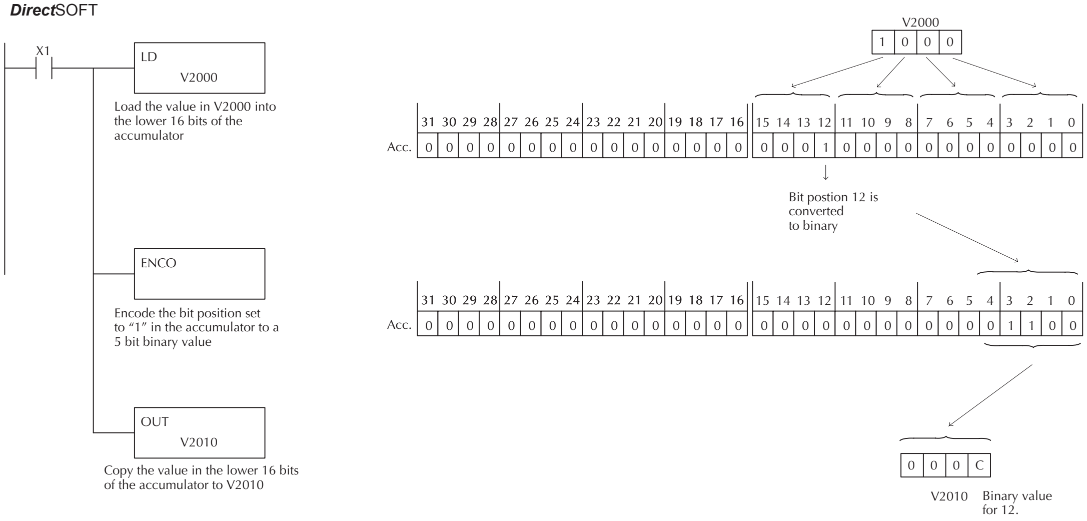
The Encode instruction encodes the bit position in the accumulator having a value of 1, and returns the appropriate binary representation. If the most significant bit is set to 1 (Bit 31), the Encode instruction would place the value HEX 1F (decimal 31) in the accumulator. If the value to be encoded is 0000 or 0001, the instruction will place a zero in the accumulator. If the value to be encoded has more than one bit position set to a "1", the least significant "1" will be encoded and SP53 will be set on.

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

***NOTE**: The status flags are only valid until another instruction that uses the same flags is executed.*

In the following example, when X1 is on, The value in V2000 is loaded into the accumulator using the Load instruction. The bit position set to a "1" in the accumulator is encoded to the corresponding 5 bit binary value using the Encode instruction. The value in the lower 16 bits of the accumulator is copied to V2010 using the Out instruction.

***Direct**SOFT*


X1
LD
V2000
Load the value in V2000 into the lower 16 bits of the accumulator
V2000
1 0 0 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
Bit postion 12 is converted to binary
ENCO
Encode the bit position set to "1" in the accumulator to a 5 bit binary value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
OUT
V2010
Copy the value in the lower 16 bits of the accumulator to V2010
0 0 0 C
V2010
Binary value for 12.

Handheld Programmer Keystrokes

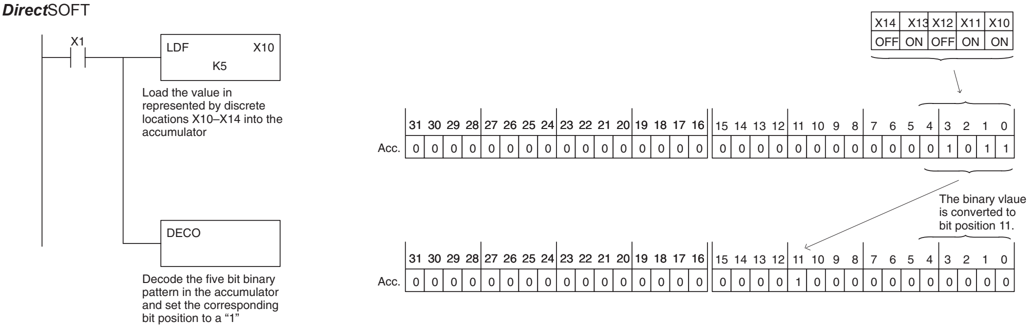
$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	E 4	N TMR	C 2	O INST#	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

Decode (DECO)

DS	Used
HPP	Used

The Decode instruction decodes a 5-bit binary value of 0–31 (0–1Fh) in the accumulator by setting the appropriate bit position to a 1. If the accumulator contains the value Fh (HEX), bit 15 will be set in the accumulator. If the value to be decoded is greater than 31, the number is divided by 32 until the value is less than 32 and then the value is decoded.

In the following example when X1 is on, the value formed by discrete locations X10–X14 is loaded into the accumulator using the Load Formatted instruction. The 5- bit binary pattern in the accumulator is decoded by setting the corresponding bit position to a "1" using the Decode instruction.


*Direct*SOFT
X1
LDF X10
K5
Load the value in represented by discrete locations X10–X14 into the accumulator
DECO
Decode the five bit binary pattern in the accumulator and set the corresponding bit position to a "1"
X14 X13 X12 X11 X10
OFF ON OFF ON ON
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1
The binary vlaue is converted to bit position 11.
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	F 5	→	B 1	A 0	→	F 5	ENT
SHFT	D 3	E 4	C 2	O INST#	ENT				

Number Conversion Instructions (Accumulator)

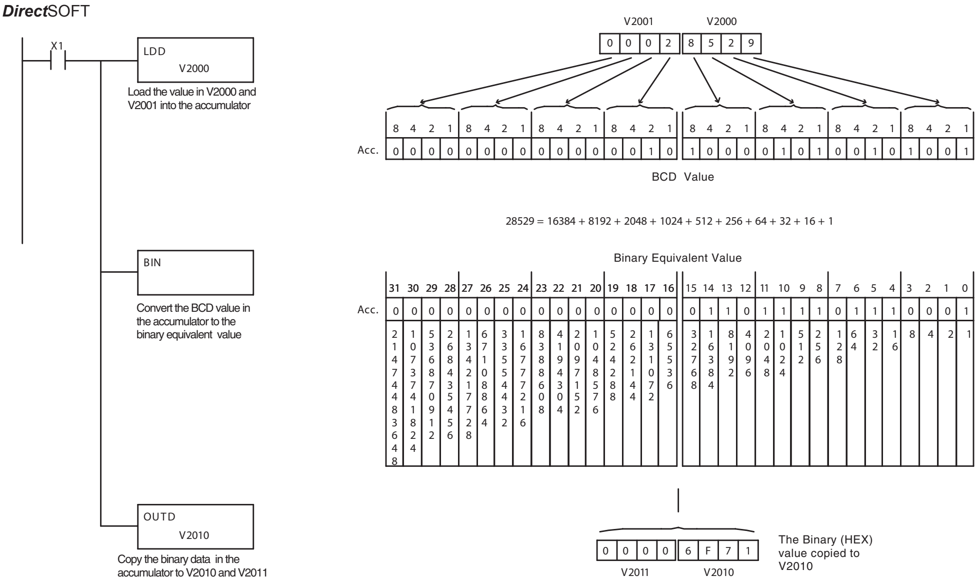
Binary (BIN)

DS	Used
HPP	Used

The Binary instruction converts a BCD value in the accumulator to the equivalent binary, or decimal, value. The result resides in the accumulator.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

In the following example, when X1 is on, the value in V2000 and V2001 is loaded into the accumulator using the Load Double instruction. The BCD value in the accumulator is converted to the binary (HEX) equivalent using the BIN instruction. The binary value in the accumulator is copied to V2010 and V2011 using the Out Double instruction. (The handheld programmer will display the binary value in V2010 and V2011 as a HEX value.)



Handheld Programmer Keystrokes

```
$ STR   →   B 1   ENT
SHFT   L ANDST   D 3   D 3   →   C 2   A 0   A 0   A 0   ENT
SHFT   B 1   I 8   N TMR   ENT
GX OUT   SHFT   D 3   →   C 2   A 0   B 1   A 0   ENT
```

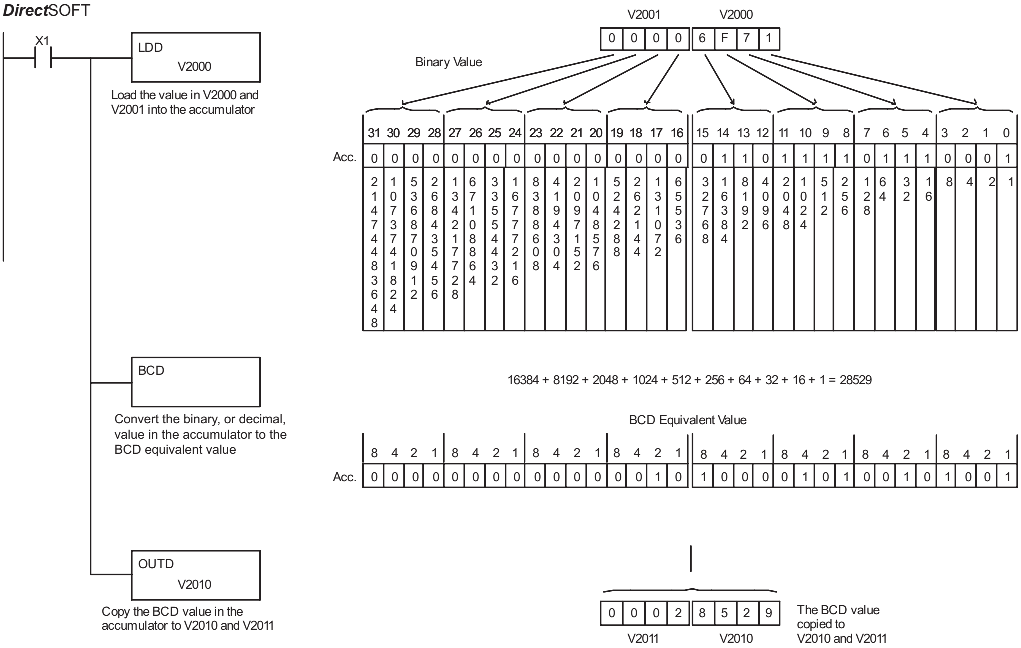
Binary Coded Decimal (BCD)

DS	Used
HPP	Used

The Binary Coded Decimal instruction converts a binary, or decimal, value in the accumulator to the equivalent BCD value. The result resides in the accumulator.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the binary, or decimal, value in V2000 and V2001 is loaded into the accumulator using the Load Double instruction. The value in the accumulator is converted to the BCD equivalent value using the BCD instruction. The BCD value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



Handheld Programmer Keystrokes

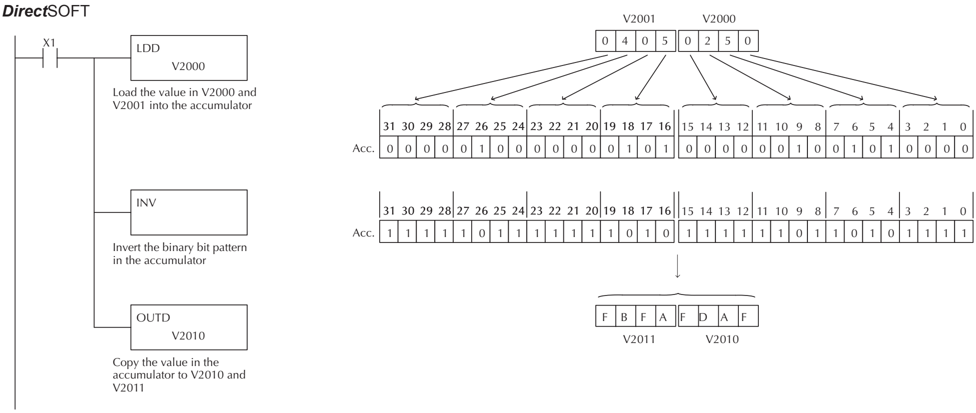
$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	B 1	C 2	D 3	ENT					
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

Invert (INV)

DS	Used
HPP	Used

The Invert instruction inverts or takes the one's complement of the 32-bit value in the accumulator. The result resides in the accumulator.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is inverted using the Invert instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.


DirectSOFT
X1
LDD
V2000
Load the value in V2000 and V2001 into the accumulator
INV
Invert the binary bit pattern in the accumulator
OUTD
V2010
Copy the value in the accumulator to V2010 and V2011
V2001
V2000
0 4 0 5 0 2 5 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Acc. 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1
F B F A F D A F
V2011
V2010

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	I 8	N TMR	V AND	ENT					
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

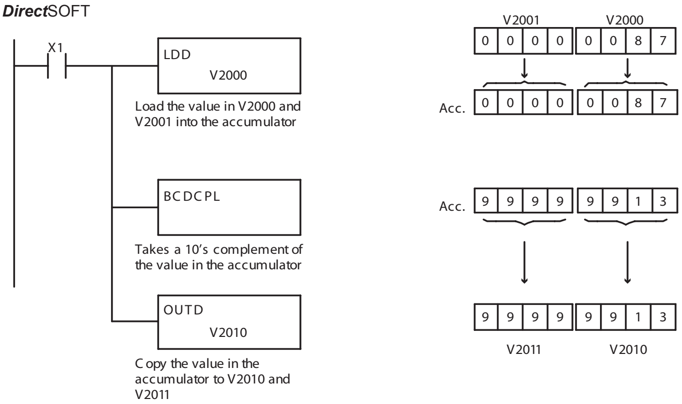
Ten's Complement (BCDCPL)

DS	Used
HPP	Used

The Ten's Complement instruction takes the 10's complement (BCD) of the 8 digit accumulator. The result resides in the accumulator. The calculation for this instruction is:

$$\begin{array}{c} 100000000 \\ -\ \text{accumulator} \\ \hline \text{10's complement value} \end{array}$$

In the following example when X1 is on, the value in V2000 and V2001 is loaded into the accumulator. The 10's complement is taken for the 8 digit accumulator using the Ten's Complement instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.


DirectSOFT
X1
LDD
V2000
Load the value in V2000 and V2001 into the accumulator
BCDCPL
Takes a 10's complement of the value in the accumulator
OUTD
V2010
Copy the value in the accumulator to V2010 and V2011
V2001
V2000
0 0 0 0 0 0 8 7
Acc.
0 0 0 0 0 0 8 7
Acc.
9 9 9 9 9 9 1 3
9 9 9 9 9 9 1 3
V2011
V2010

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	B 1	C 2	D 3	C 2	P CV	L ANDST	ENT		
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

Binary to Real Conversion (BTOR)

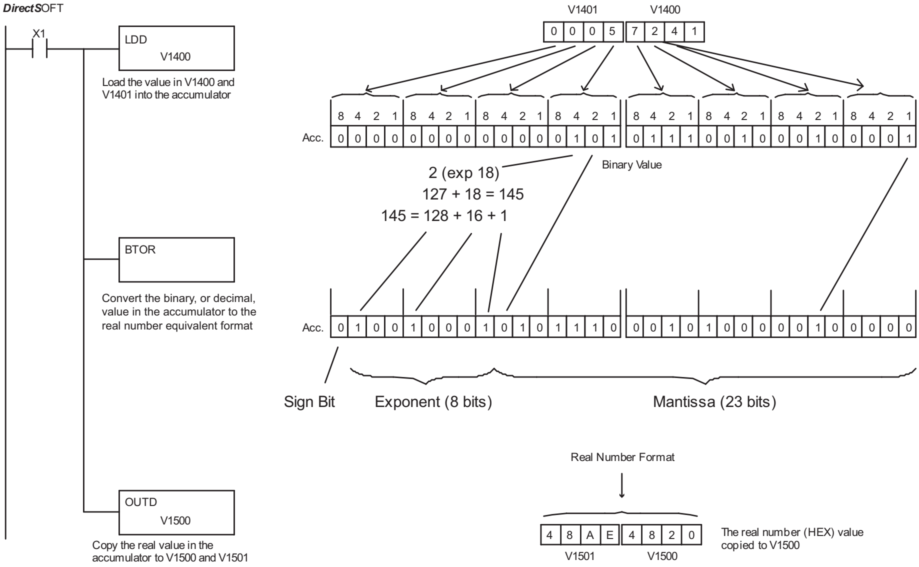
DS	Used
HPP	Used

The Binary-to-Real instruction converts a binary, or decimal, value in the accumulator to its equivalent real number (floating point) format. The result resides in the accumulator. Both the binary and the real number may use all 32 bits of the accumulator.

***NOTE**: This instruction only works with unsigned **binary, or decimal,** values. It will not work with signed decimal values.*

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the value in V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The BTOR instruction converts the binary, or decimal, value in the accumulator to the equivalent real number format. The binary weight of the MSB is converted to the real number exponent by adding it to 127 (decimal). Then the remaining bits are copied to the mantissa as shown. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction. The handheld programmer would display the binary value in V1500 and V1501 as a HEX value.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	B 1	T MLR	O INST#	R ORN	ENT				
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Real to Binary Conversion (RTOB)

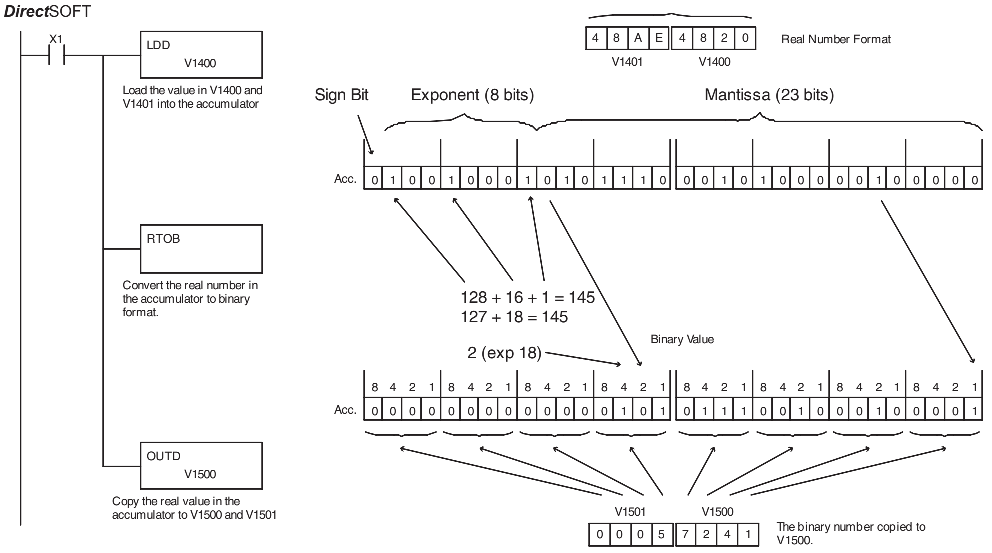
DS	Used
HPP	Used

The Real-to-Binary instruction converts the real number in the accumulator to a binary value. The result resides in the accumulator. Both the binary and the real number may use all 32 bits of the accumulator.

***NOTE$_1$:** The decimal portion of the result will be rounded down (14.1 to 14; -14.1 to -15).*
***NOTE$_2$:** if the real number is negative, it becomes a signed decimal value.*

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a number cannot be converted to binary.

In the following example, when X1 is on, the value in V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The RTOB instruction converts the real value in the accumulator the equivalent binary number format. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction. The handheld programmer would display the binary value in V1500 and V1501 as a HEX value.



Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	R ORN	T MLR	O INST#	B 1	ENT				
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Radian Real Conversion (RADR)

DS	Used
HPP	N/A

The Radian Real Conversion instruction converts the real degree value stored in the accumulator to the equivalent real number in radians. The result resides in the accumulator.

Degree Real Conversion (DEGR)

DS32	Used
HPP	N/A

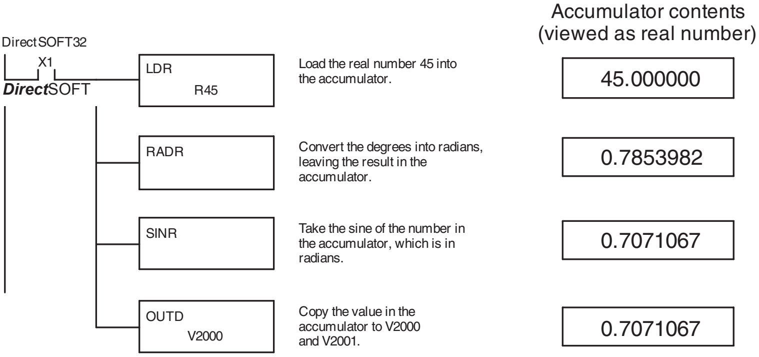
The Degree Real instruction converts the degree real radian value stored in the accumulator to the equivalent real number in degrees. The result resides in the accumulator.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a number cannot be converted to binary.

The two instructions described above convert real numbers into the accumulator from degree format to radian format, and vice-versa. In degree format, a circle contains 360 degrees. In radian format, a circle contains 2π (about 6.28) radians. These convert between both positive and negative real numbers, and for angles greater than a full circle. These functions are very useful when combined with the transcendental trigonometric functions (see the section on math instructions).

***NOTE**: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **Direct**SOFT for entering real numbers, using the LDR (Load Real) instruction.*

The following example takes the sine of 45 degrees. Since transcendental functions operate only on real numbers, we do an LDR (load real) 45. The trig functions operate only in radians, so we must convert the degrees to radians by using the RADR command. After using the SINR (Sine Real) instruction, we use an OUTD (Out Double) instruction to move the result from the accumulator to V-memory. The result is 32-bits wide, requiring the Out Double to move it.



Load the real number 45 into the accumulator. — 45.000000

Convert the degrees into radians, leaving the result in the accumulator. — 0.7853982

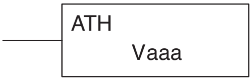
Take the sine of the number in the accumulator, which is in radians. — 0.7071067

Copy the value in the accumulator to V2000 and V2001. — 0.7071067

ASCII to HEX (ATH)

DS	Used
HPP	N/A

The ASCII TO HEX instruction converts a table of ASCII values to a specified table of HEX values. ASCII values are two digits and their HEX equivalents are one digit. This means an ASCII table of four V-memory locations would only require two V-memory locations for the equivalent HEX table. The function parameters are loaded into the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program an ASCII to HEX table function. The example on the following page shows a program for the ASCII to HEX table function.



Step 1: Load the number of V-memory locations for the ASCII table into the first level of the accumulator stack.

Step 2: Load the starting V-memory location for the ASCII table into the accumulator. This parameter must be a HEX value.

Step 3: Specify the starting V-memory location (Vaaa) for the HEX table in the ATH instruction.

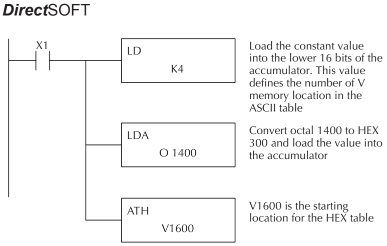
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

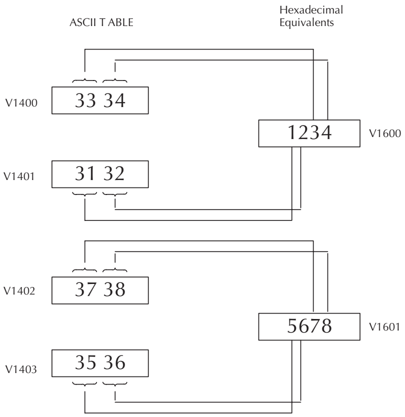
In the example on the following page, when X1 is ON the constant (K4) is loaded into the accumulator using the Load instruction and will be placed in the first level of the accumulator stack when the next Load instruction is executed. The starting location for the ASCII table (V1400) is loaded into the accumulator using the Load Address instruction. The starting location for the HEX table (V1600) is specified in the ASCII to HEX instruction. The table below lists valid ASCII values for ATH conversion.

ASCII Values Valid for ATH Conversion			
ASCII Value	Hex Value	ASCII Value	Hex Value
30	0	38	8
31	1	39	9
32	2	41	A
33	3	42	B
34	4	43	C
35	5	44	D
36	6	45	E
37	7	46	F


*Direct*SOFT
X1
LD
K4
Load the constant value into the lower 16 bits of the accumulator. This value defines the number of V memory location in the ASCII table
LDA
O 1400
Convert octal 1400 to HEX 300 and load the value into the accumulator
ATH
V1600
V1600 is the starting location for the HEX table

Handheld Programmer Keystrokes

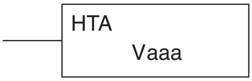
$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	PREV	E 4	ENT			
SHFT	L ANDST	D 3	A 0	→	B 1	E 4	A 0	A 0	ENT
SHFT	A 0	T MLR	H 7	→	B 1	G 6	A 0	A 0	ENT



HEX to ASCII (HTA)

DS	Used
HPP	N/A

The HEX to ASCII instruction converts a table of HEX values to a specified table of ASCII values. HEX values are one digit and their ASCII equivalents are two digits.



This means a HEX table of two V-memory locations would require four V-memory locations for the equivalent ASCII table. The function parameters are loaded into the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program a HEX to ASCII table function. The example on the following page shows a program for the HEX to ASCII table function.

Step 1: Load the number of V-memory locations in the HEX table into the first level of the accumulator stack.

Step 2: Load the starting V-memory location for the HEX table into the accumulator. This parameter must be a HEX value.

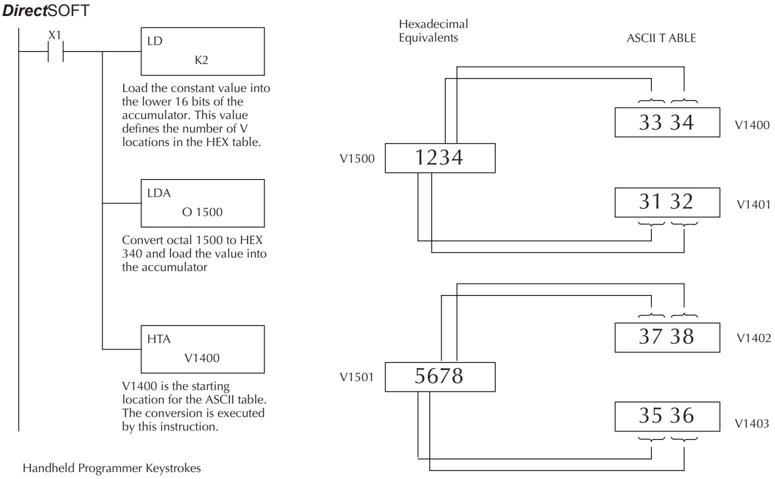
Step 3:Specify the starting V-memory location (Vaaa) for the ASCII table in the HTA instruction.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

In the following example, when X1 is ON, the constant (K2) is loaded into the accumulator using the Load instruction. The starting location for the HEX table (V1500) is loaded into the accumulator using the Load Address instruction. The starting location for the ASCII table (V1400) is specified in the HEX to ASCII instruction.


DirectSOFT
X1
LD
K2
Load the constant value into the lower 16 bits of the accumulator. This value defines the number of V locations in the HEX table.
LDA
O 1500
Convert octal 1500 to HEX 340 and load the value into the accumulator
HTA
V1400
V1400 is the starting location for the ASCII table. The conversion is executed by this instruction.
Hexadecimal Equivalents
ASCII T ABLE
33 34
V1400
V1500
1234
31 32
V1401
37 38
V1402
V1501
5678
35 36
V1403

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	ENT		
SHFT	L ANDST	D 3	A 0	→	B 1	F 5	A 0	A 0	ENT
SHFT	H 7	T MLR	A 0	→	B 1	E 4	A 0	A 0	ENT

The table below lists valid ASCII values for HTA conversion.

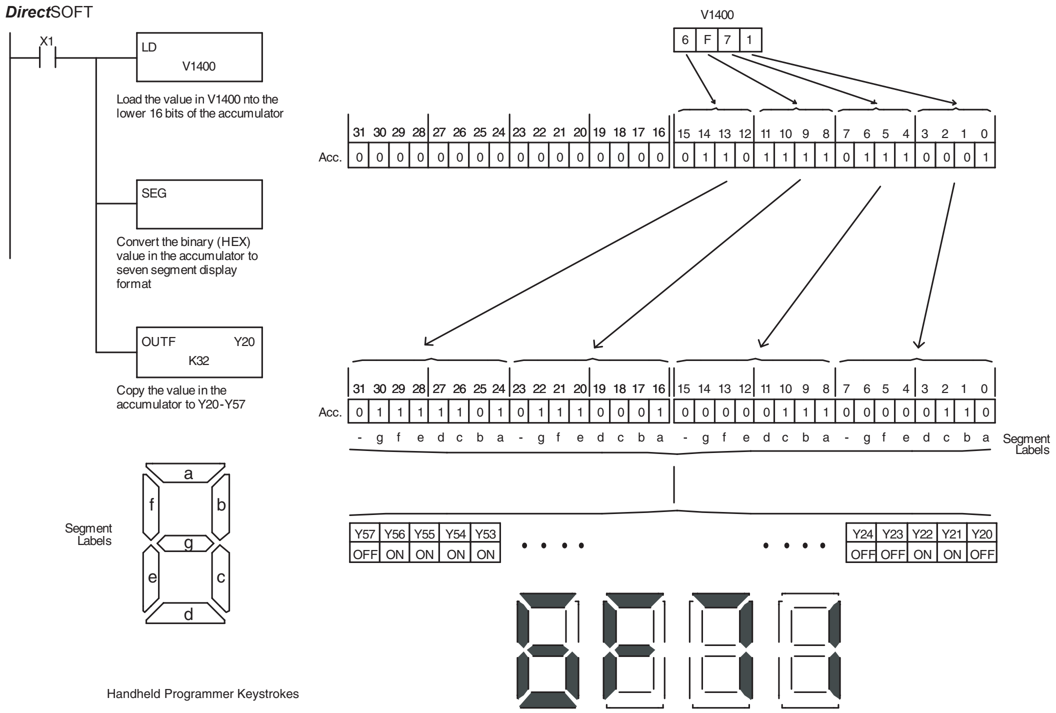
ASCII Values Valid for HTA Conversion			
Hex Value	ASCII Value	Hex Value	ASCII Value
0	30	8	38
1	31	9	39
2	32	A	41
3	33	B	42
4	34	C	43
5	35	D	44
6	36	E	45
7	37	F	46

Segment (SEG)

DS	Used
HPP	Used

The BCD / Segment instruction converts a four digit HEX value in the accumulator to seven segment display format. The result resides in the accumulator.

In the following example, when X1 is on, the value in V1400 is loaded into the lower 16 bits of the accumulator using the Load instruction. The HEX value in the accumulator is converted to seven segment format using the Segment instruction. The bit pattern in the accumulator is copied to Y20–Y57 using the Out Formatted instruction.


DirectSOFT
X1
LD
V1400
Load the value in V1400 nto the lower 16 bits of the accumulator
SEG
Convert the binary (HEX) value in the accumulator to seven segment display format
OUTF Y20
K32
Copy the value in the accumulator to Y20-Y57
V1400
6 F 7 1
Acc.
Acc.
Segment Labels
Segment Labels
a
f
b
g
e
c
d
• • • •
• • • •

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0	1	1	1	0	0	0	1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	1	0	1	1	1	0	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0
-	g	f	e	d	c	b	a	-	g	f	e	d	c	b	a	-	g	f	e	d	c	b	a	-	g	f	e	d	c	b	a

Y57	Y56	Y55	Y54	Y53	Y24	Y23	Y22	Y21	Y20
OFF	ON	ON	ON	ON	OFF	OFF	ON	ON	OFF

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
L ANDST	D 3	→	B 1	E 4	A 0	A 0	ENT		
SHFT	S RST	SHFT	E 4	G 6	ENT				
GX OUT	SHFT	F 5	→	C 2	A 0	→	D 3	C 2	ENT

Gray Code (GRAY)

DS	Used
HPP	Used

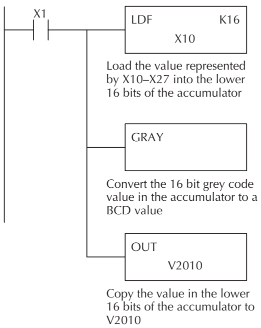
The Gray code instruction converts a 16-bit gray code value to a BCD value. The BCD conversion requires 10 bits of the accumulator. The upper 22 bits are set to "0"

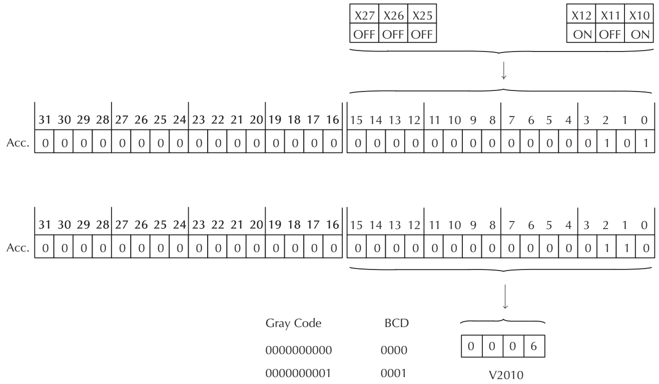
This instruction is designed for use with devices (typically encoders) that use the gray code numbering scheme. The Gray Code instruction will directly convert a gray code number to a BCD number for devices having a resolution of 512 or 1024 counts per revolution. If a device having a resolution of 360 counts per revolution is to be used, you must subtract a BCD value of 76 from the converted value to obtain the proper result. For a device having a resolution of 720 counts per revolution, you must subtract a BCD value of 152.

In the following example, when X1 is ON, the binary value represented by X10–X27 is loaded into the accumulator using the Load Formatted instruction. The gray code value in the accumulator is converted to BCD using the Gray Code instruction. The value in the lower 16 bits of the accumulator is copied to V2010.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

*Direct*SOFT


X1
LDF K16
X10
Load the value represented by X10–X27 into the lower 16 bits of the accumulator
GRAY
Convert the 16 bit grey code value in the accumulator to a BCD value
OUT
V2010
Copy the value in the lower 16 bits of the accumulator to V2010



Gray Code	BCD
0000000000	0000
0000000001	0001
0000000011	0002
0000000010	0003
0000000110	0004
0000000111	0005
0000000101	0006
0000000100	0007
1000000001	1022
1000000000	1023

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	F 5	→	B 1	A 0	→	B 1	G 6	ENT
SHFT	G 6	R ORN	A 0	Y MLS	ENT					
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT		

Shuffle Digits (SFLDGT)

DS	Used
HPP	Used

The Shuffle Digits instruction shuffles a maximum of 8 digits, rearranging them in a specified order. This function requires parameters to be loaded into the first level of the accumulator stack and the accumulator with two additional instructions. Listed below are the steps necessary to use the shuffle digit function. The example on the following page shows a program for the Shuffle Digits function.

Step 1: Load the value (digits) to be shuffled into the first level of the accumulator stack.

Step 2: Load the order that the digits will be shuffled to into the accumulator.

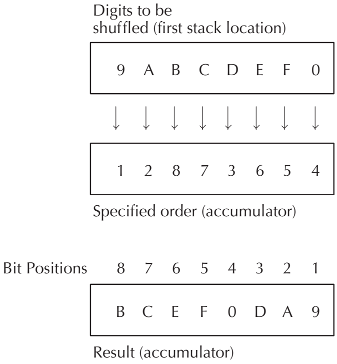
Step 3: Insert the SFLDGT instruction.

***NOTE**: If the number used to specify the order contains a 0 or 9–F, the corresponding position will be set to 0.*

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

Shuffle Digits Block Diagram

There are a maximum of 8 digits that can be shuffled. The bit positions in the first level of the accumulator stack define the digits to be shuffled. They correspond to the bit positions in the accumulator that define the order the digits will be shuffled. The digits are shuffled and the result resides in the accumulator.

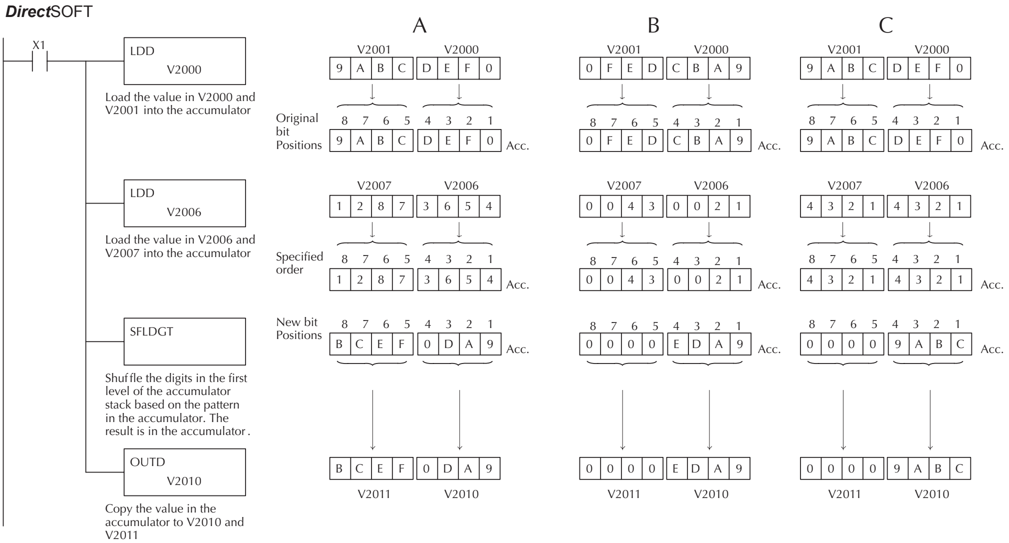

Digits to be shuffled (first stack location)
9 A B C D E F 0
1 2 8 7 3 6 5 4
Specified order (accumulator)
Bit Positions 8 7 6 5 4 3 2 1
B C E F 0 D A 9
Result (accumulator)

In the following example, when X1 is on, the value in the first level of the accumulator stack will be reorganized in the order specified by the value in the accumulator.

Example A shows how the shuffle digits works when 0 or 9–F is not used when specifying the order the digits are to be shuffled. Also, there are no duplicate numbers in the specified order.

Example B shows how the Shuffle Digits works when a 0 or 9–F is used when specifying the order the digits are to be shuffled. Notice when the Shuffle Digits instruction is executed, the bit positions in the first stack location that had a corresponding 0 or 9–F in the accumulator (order specified) are set to “0”.

Example C shows how the Shuffle Digits works when duplicate numbers are used specifying the order the digits are to be shuffled. Notice when the Shuffle Digits instruction is executed, the most significant duplicate number in the order specified is used in the result.


DirectSOFT
X1
LDD V2000
Load the value in V2000 and V2001 into the accumulator
LDD V2006
Load the value in V2006 and V2007 into the accumulator
SFLDGT
Shuffle the digits in the first level of the accumulator stack based on the pattern in the accumulator. The result is in the accumulator.
OUTD V2010
Copy the value in the accumulator to V2010 and V2011
A
B
C
V2001
V2000
Original bit Positions
8 7 6 5 4 3 2 1
Acc.
V2007
V2006
Specified order
New bit Positions
V2011
V2010
A: 9 A B C D E F 0; 1 2 8 7 3 6 5 4; B C E F 0 D A 9
B: 0 F E D C B A 9; 0 0 4 3 0 0 2 1; 0 0 0 0 E D A 9
C: 9 A B C D E F 0; 4 3 2 1 4 3 2 1; 0 0 0 0 9 A B C

Handheld Programmer Keystrokes

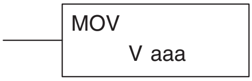
$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	G 6	ENT
SHFT	S RST	SHFT	F 5	L ANDST	D 3	G 6	T MLR	ENT	
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

Table Instructions

Move (MOV)

DS	Used
HPP	Used

The Move instruction moves the values from a V-memory table to another V-memory table the same length (a table being a consecutive group of V-memory locations). The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. The MOV instruction can be used to write data to non-volatile V-memory (see Appendix F). Listed below are the steps necessary to program the MOV function.



- Step 1 Load the number of V-memory locations to be moved into the first level of the accumulator stack. This parameter is a HEX value (KFFF max, 7777 octal, 4096 decimal).
- Step 2 Load the starting V-memory location for the locations to be moved into the accumulator. This parameter is a HEX value.
- Step 3 Insert the MOV instruction which specifies starting V-memory location (Vaaa) for the destination table.

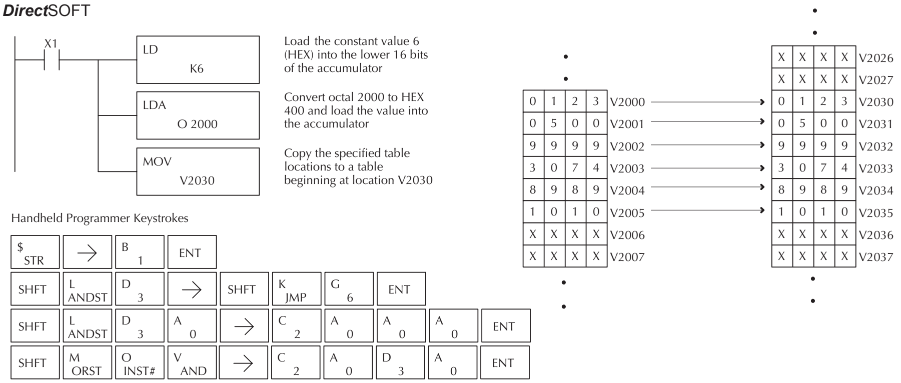
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 2000 (V2000), the starting location for the source table, is loaded into the accumulator. The destination table location (V2030) is specified in the Move instruction.

*Direct*SOFT



Load the constant value 6 (HEX) into the lower 16 bits of the accumulator

Convert octal 2000 to HEX 400 and load the value into the accumulator

Copy the specified table locations to a table beginning at location V2030

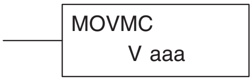
Handheld Programmer Keystrokes

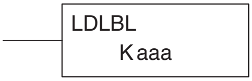
Move Memory Cartridge (MOVMC)

Load Label (LDLBL)

DS	Used
HPP	Used

The Move Memory Cartridge instruction is used to copy data between V-memory and program ladder memory. The Load Label instruction is *only* used with the MOVMC instruction when copying data *from* program ladder memory to V-memory.





To copy data between V-memory and program ladder memory, the function parameters are loaded into the first two levels of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the MOVMC and LDLBL functions.

- Step 1: Load the number of words to be copied into the second level of the accumulator stack.
- Step 2: Load the offset for the data label area in ladder memory and the beginning of the V-memory block into the first level of the stack.
- Step 3: Load the *source data label* (LDLBL Kaaa) into the accumulator when copying data from ladder memory to V-memory. Load the *source address* into the accumulator when copying data from V-memory to ladder memory. This is where the value will be copied from. If the source address is a V-memory location, the value must be entered in HEX.
- Step 4: Insert the MOVMC instruction which specifies destination in V-memory (Vaaa). This is the copy destination.

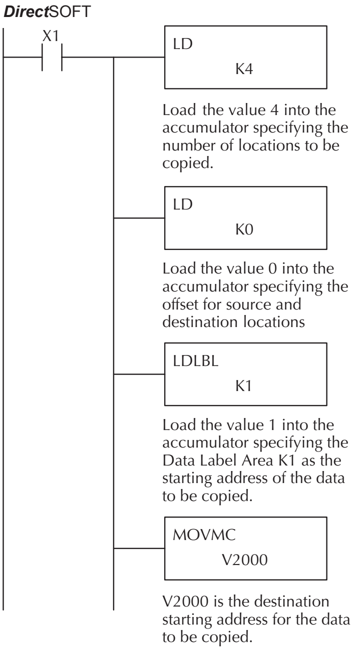
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

NOTE: *Refer to page 5-188 for an example.*

WARNING: The offset for this usage of the instruction starts at 0, but may be any number that does not result in data outside of the source data area being copied into the destination table. When an offset is outside of the source information boundaries, then unknown data values will be transferred into the destination table.

Copy Data From a Data Label Area to V-memory

In the example below, data is copied from a Data Label Area to V-memory. When X1 is on, the constant value (K4) is loaded into the accumulator using the Load (LD) instruction. This value specifies the length of the table and is placed in the second stack location after the next Load and Load Label (LDLBL) instructions are executed. The constant value (K0) is loaded into the accumulator, specifying the offset for the source and destination data. It is placed in the first stack location after the LDLBL instruction is executed. The source address where data is being copied from is loaded into the accumulator using the LDLBL instruction. The MOVMC instruction specifies the destination starting location and executes the copying of data from the Data Label Area to V-memory.

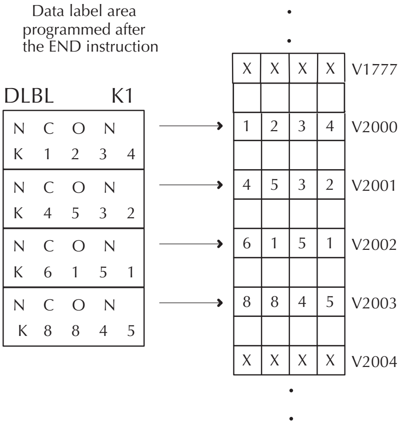


Load the value 4 into the accumulator specifying the number of locations to be copied.

Load the value 0 into the accumulator specifying the offset for source and destination locations

Load the value 1 into the accumulator specifying the Data Label Area K1 as the starting address of the data to be copied.

V2000 is the destination starting address for the data to be copied.



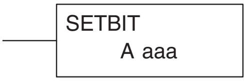
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT								
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	ENT				
SHFT	L ANDST	D 3	→	SHFT	K JMP	A 0	ENT				
SHFT	L ANDST	D 3	L ANDST	B 1	L ANDST	→	B 1	ENT			
SHFT	M ORST	O INST#	V AND	M ORST	C 2	→	C 2	A 0	A 0	A 0	ENT

SETBIT

DS	Used
HPP	Used

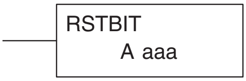
The Set Bit instruction sets a single bit to one within a range of V-memory locations.



RSTBIT

DS	Used
HPP	Used

The Reset Bit instruction resets a single bit to zero within a range of V-memory locations.



The following description applies to both the Set Bit and Reset Bit table instructions.

Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: Insert the Set Bit or Reset Bit instruction. This specifies the reference for the bit number of the bit you want to set or reset. The bit number is in octal, and the first bit in the table is number “0”.

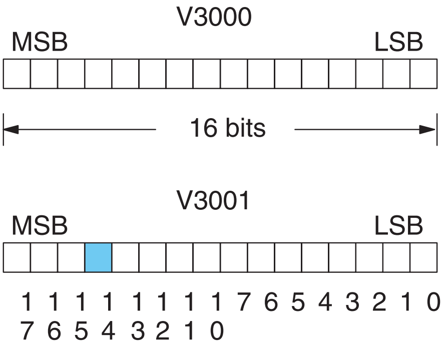
Helpful hint: — Remember that each V-memory location contains 16 bits. So, the bits of the first word of the table are numbered from 0 to 17 octal. For example, if the table length is six words, then 6 words = (6 x 16) bits, = 96 bits (decimal), or 140 octal. The permissible range of bit reference numbers would be 0 to 137 octal. SP 53 will be set if the bit specified is outside the range of the table.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

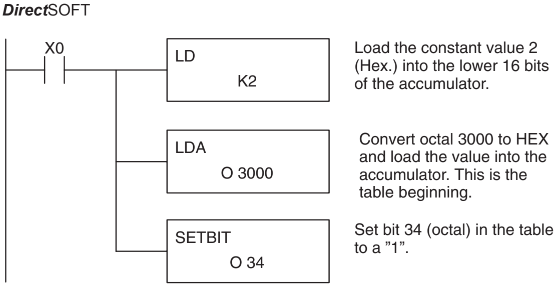
Discrete Bit Flags	Description
SP53	On when the specified bit is outside the range of the table.

NOTE*: Status flags are only valid until the end of the scan or until another instruction that uses the same flag is executed.*

For example, suppose we have a table starting at V3000 that is two words long, as shown to the right. Each word in the table contains 16 bits, or 0 to 17 in octal. To set bit 12 in the second word, we use its octal reference (bit 14). Then we compute the bit’s octal address from the start of the table, so 17 + 14 = 34 octal. The following program shows how to set the bit as shown to a “1”.



In this ladder example, we will use input X0 to trigger the Set Bit operation. First, we will load the table length (2 words) into the accumulator stack. Next, we load the starting address into the accumulator. Since V3000 is an octal number, we have to convert it to hex by using the LDA command. Finally, we use the Set Bit (or Reset Bit) instruction and specify the octal address of the bit (bit 34), referenced from the table.

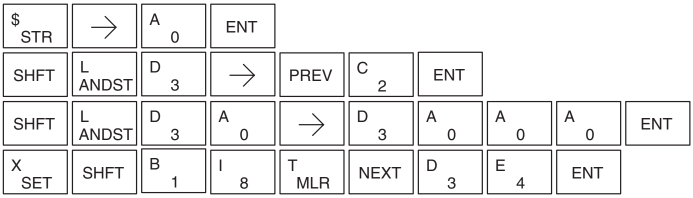


Load the constant value 2 (Hex.) into the lower 16 bits of the accumulator.

Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning.

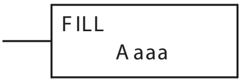
Set bit 34 (octal) in the table to a ”1”.

Handheld Programmer Keystrokes


$ STR | → | A 0 | ENT
SHFT | L ANDST | D 3 | → | PREV | C 2 | ENT
SHFT | L ANDST | D 3 | A 0 | → | D 3 | A 0 | A 0 | A 0 | ENT
X SET | SHFT | B 1 | I 8 | T MLR | NEXT | D 3 | E 4 | ENT

Fill (FILL)

DS	Used
HPP	Used



The Fill instruction fills a table of up to 255 V-memory locations with a value (Aaaa), which is either a V-memory location or a 4-digit constant. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions

Listed below are the steps necessary to program the Fill function.

Step 1: Load the number of V-memory locations to be filled into the first level of the accumulator stack. This parameter must be a HEX value, 0–FF.

Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value.

Step 3: Insert the Fill instruction which specifies the value to fill the table with.

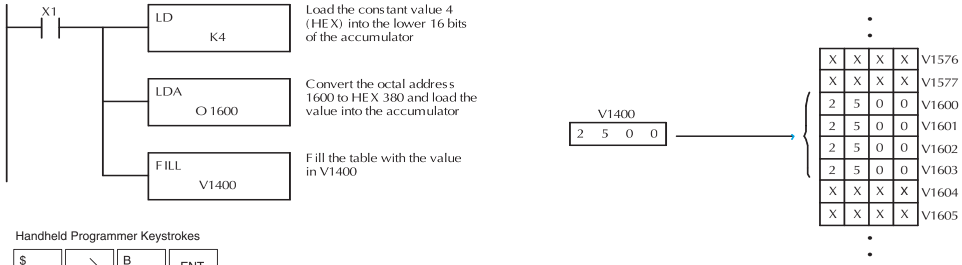
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FF

Discrete Bit Flags	Description
SP53	On if the V-memory address is out of range.

In the following example, when X1 is on, the constant value (K4) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed on the first level of the accumulator stack when the Load Address instruction is executed. The octal address 1600 (V1600) is the starting location for the table and is loaded into the accumulator using the Load Address instruction. The value to fill the table with (V1400) is specified in the Fill instruction.

*Direct*SOFT

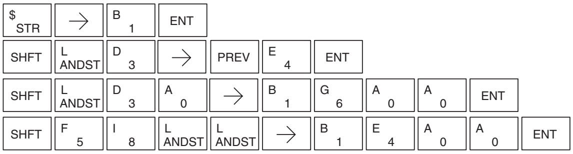


Load the constant value 4 (HEX) into the lower 16 bits of the accumulator

Convert the octal address 1600 to HEX 380 and load the value into the accumulator

Fill the table with the value in V1400

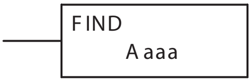
Handheld Programmer Keystrokes


$ STR | → | B 1 | ENT
SHFT | L ANDST | D 3 | → | PREV | E 4 | ENT
SHFT | L ANDST | D 3 | A 0 | → | B 1 | G 6 | A 0 | A 0 | ENT
SHFT | F 5 | I 8 | L ANDST | L ANDST | → | B 1 | E 4 | A 0 | A 0 | ENT

Find (FIND)

DS	Used
HPP	Used

The Find instruction is used to search for a specified value in a V-memory table of up to 255 locations. The function parameters are loaded into the first and second levels of the accumulator stack and the accumulator by three additional instructions.



Listed below are the steps necessary to program the Find function.

Step 1: Load the length of the table (number of V-memory locations) into the second level of the accumulator stack. This parameter must be a HEX value, 0–FF.

Step 2: Load the starting V-memory location for the table into the first level of the accumulator stack. This parameter must be a HEX value.

Step 3: Load the offset from the starting location to begin the search. This parameter must be a HEX value.

Step 4: Insert the Find instruction which specifies the first value to be found in the table.

Results:— The offset from the starting address to the first V-memory location which contains the search value (in HEX) is returned to the accumulator. SP53 will be set On if an address outside the table is specified in the offset or the value is not found. If the value is not found 0 will be returned in the accumulator.

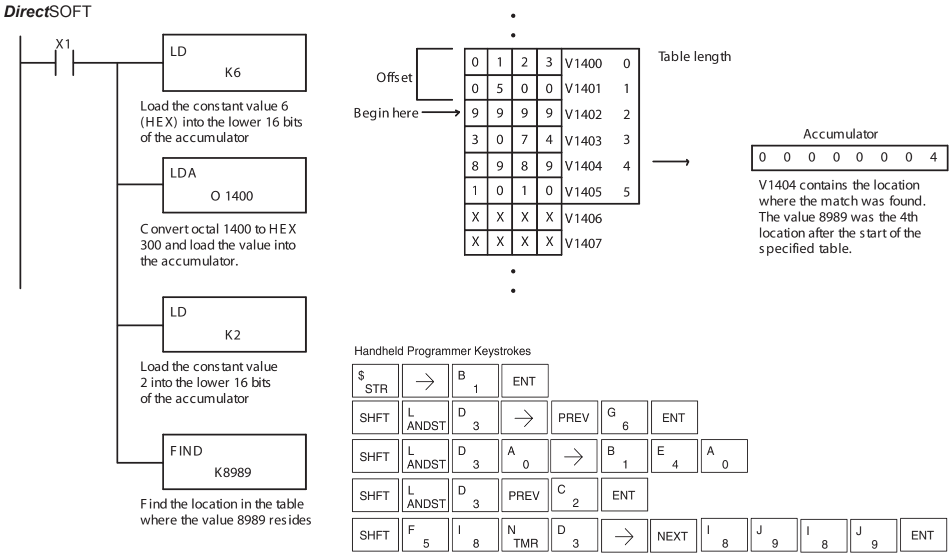
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	0–FF

Discrete Bit Flags	Description
SP53	On if there is no value in the table that is equal to the search value.

***NOTE**: Status flags are only valid until another instruction that uses the same flags is executed. The pointer for this instruction starts at 0 and resides in the accumulator.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the second stack location when the following Load Address and Load instruction is executed. The octal address 1400 (V1400) is the starting location for the table and is loaded into the accumulator. This value is placed in the first level of the accumulator stack when the following Load instruction is executed. The offset (K2) is loaded into the lower 16 bits of the accumulator using the Load instruction. The value to be found in the table is specified in the Find instruction. If a value is found equal to the search value, the offset (from the starting location of the table) where the value is located will reside in the accumulator.



Load the constant value 6 (HEX) into the lower 16 bits of the accumulator

Convert octal 1400 to HEX 300 and load the value into the accumulator.

Load the constant value 2 into the lower 16 bits of the accumulator

Find the location in the table where the value 8989 resides

0	1	2	3	V1400	0
0	5	0	0	V1401	1
9	9	9	9	V1402	2
3	0	7	4	V1403	3
8	9	8	9	V1404	4
1	0	1	0	V1405	5
X	X	X	X	V1406	
X	X	X	X	V1407	

V1404 contains the location where the match was found. The value 8989 was the 4th location after the start of the specified table.

Handheld Programmer Keystrokes

$ STR | → | B 1 | ENT

SHFT | L ANDST | D 3 | → | PREV | G 6 | ENT

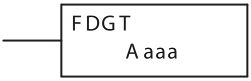
SHFT | L ANDST | D 3 | A 0 | → | B 1 | E 4 | A 0

SHFT | L ANDST | D 3 | PREV | C 2 | ENT

SHFT | F 5 | I 8 | N TMR | D 3 | → | NEXT | I 8 | J 9 | I 8 | J 9 | ENT

Find Greater Than (FDGT)

The Find Greater Than instruction is used to search for the first occurrence of a value in a V-memory table that is greater than the specified value (Aaaa), which can be either a V-memory location or a 4-digit constant. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Find Greater Than function.



Step 1: Load the length of the table (up to 255 locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0–FF.

Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value.

Step 3: Insert the FDGT instruction which specifies the greater than search value. Results:— The offset from the starting address to the first V-memory location which contains the greater than search value (in HEX) which is returned to the accumulator. SP53 will be set On if the value is not found and 0 will be returned in the accumulator.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

***NOTE**: This instruction does not have an offset, such as the one required for the FIND instruction.*

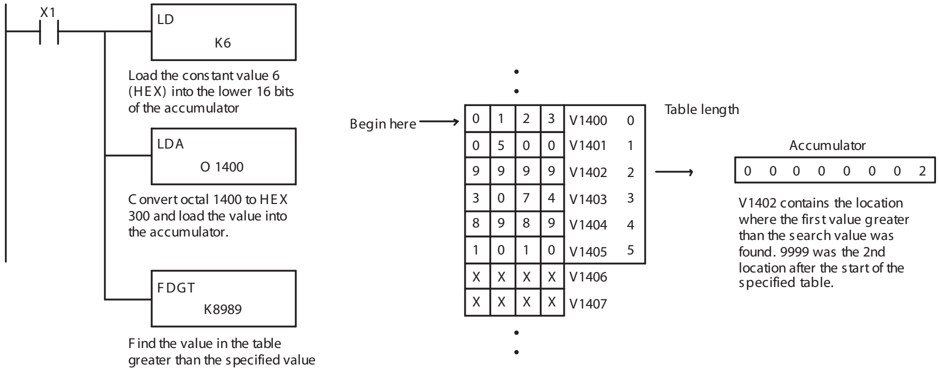
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	0–FF

Discrete Bit Flags	Description
SP53	On if there is no value in the table that is equal to the search value.

NOTE: Status flags are only valid until another instruction that uses the same flags is executed. The pointer for this instruction starts at 0 and resides in the accumulator.

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the table and is loaded into the accumulator. The Greater Than search value is specified in the Find Greater Than instruction. If a value is found greater than the search value, the offset (from the starting location of the table) where the value is located will reside in the accumulator. If there is no value in the table that is greater than the search value, a zero is stored in the accumulator and SP53 will come ON.

DirectSOFT



X1
LD
K6
Load the constant value 6 (HEX) into the lower 16 bits of the accumulator
LDA
O 1400
Convert octal 1400 to HEX 300 and load the value into the accumulator.
FDGT
K8989
Find the value in the table greater than the specified value
Begin here
Table length
Accumulator
0 0 0 0 0 0 0 2
V1402 contains the location where the first value greater than the search value was found. 9999 was the 2nd location after the start of the specified table.

0	1	2	3	V1400	0
0	5	0	0	V1401	1
9	9	9	9	V1402	2
3	0	7	4	V1403	3
8	9	8	9	V1404	4
1	0	1	0	V1405	5
X	X	X	X	V1406	
X	X	X	X	V1407	

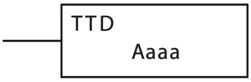
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT								
SHFT	L ANDST	D 3	→	PREV	G 6	ENT					
SHFT	L ANDST	D 3	A 0	→	B 1	E 4	A 0	A 0	ENT		
SHFT	F 5	D 3	G 6	T MLR	→	NEXT	I 8	J 9	I 8	J 9	ENT

Table to Destination (TTD)

DS	Used
HPP	Used

The Table To Destination instruction moves a value from a V-memory table to a V-memory location and increments the table pointer by 1. The first V-memory location in the table contains the table pointer which indicates the next location in the table to be moved. The instruction will be executed once per scan provided the input remains on. The table pointer will reset to 1 when the value equals the last location in the table. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions.



Listed below are the steps necessary to program the Table To Destination function.

Step 1: Load the length of the data table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table pointer.) This parameter must be a HEX value.

Step 3: Insert the TTD instruction which specifies destination V-memory location (Vaaa).

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Helpful Hint: — The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one-shot (PD) should be used in the input logic.

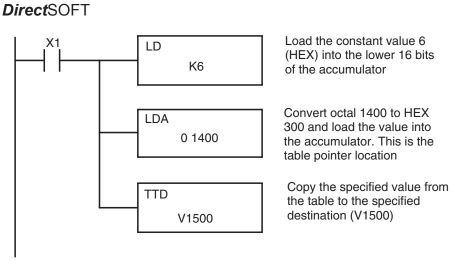
Helpful Hint: — The pointer location should be set to the value where the table operation will begin. The special relay SP0 or a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP56	On when the table pointer equals the table length.

***NOTE:** Status flags (SPs) are only valid until another instruction that uses the same flag is executed, or the end of the scan. The pointer for this instruction starts at 0 and resets when the table length is reached. At first glance it may appear that the pointer should reset to 0. However, it resets to 1, not 0.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the source table and is loaded into the accumulator. Remember, V1400 is used as the pointer location, and is not actually part of the table data source. The destination location (V1500) is specified in the Table to Destination instruction. The table pointer (V1400 in this case) will be increased by “1” after each execution of the TTD instruction.



Load the constant value 6 (HEX) into the lower 16 bits of the accumulator

Convert octal 1400 to HEX 300 and load the value into the accumulator. This is the table pointer location

Copy the specified value from the table to the specified destination (V1500)

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	PREV	G 6	ENT			
SHFT	L ANDST	D 3	A 0	→	B 1	E 4	A 0	A 0	ENT
SHFT	T MLR	T MLR	D 3	→	B 1	F 5	A 0	A 0	ENT

It is important to understand how the table locations are numbered. If you examine the example table, you’ll notice that the first data location, V1401, will be used when the pointer is equal to zero, and again when the pointer is equal to six. Why? Because the pointer is only equal to zero before the very first execution. From then on, it increments from one to six, and then resets to one.

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

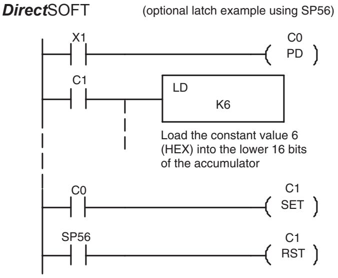
Table Pointer

0	0	0	0	V1400

Destination

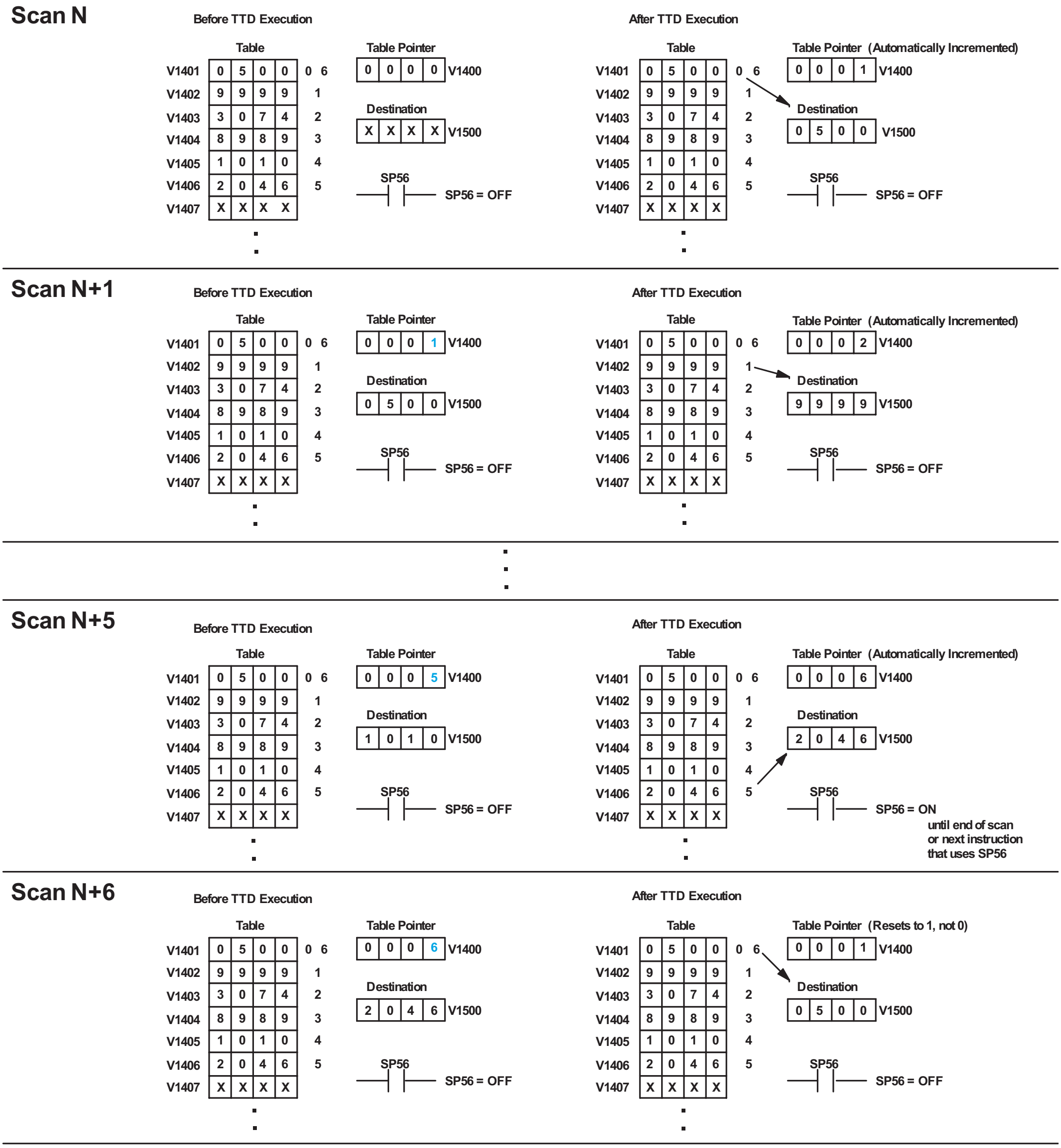
X	X	X	X	V1500

Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer increments automatically, the table would cycle through the locations very quickly. If this is a problem, you have an option of using SP56 in conjunction with a one-shot (PD) and a latch (C1 for example) to allow the table to cycle through all locations one time and then stop. The logic shown here is not required, it’s just an optional method.


*Direct*SOFT
(optional latch example using SP56)
X1
C0
PD
C1
LD
K6
Load the constant value 6 (HEX) into the lower 16 bits of the accumulator
C0
C1
SET
SP56
C1
RST

Since Special Relays are reset at the end of the scan, this latch must follow the TTD instruction in the program

The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically cycles from 0 – 6, and then starts over at 1 instead of 0. Also, notice how SP56 is only on until the end of the scan.



Scan N

Before TTD Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

Table Pointer: 0 0 0 0 V1400

Destination: X X X X V1500

SP56 — SP56 = OFF

After TTD Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

Table Pointer (Automatically Incremented): 0 0 0 1 V1400

Destination: 0 5 0 0 V1500

SP56 — SP56 = OFF

Scan N+1

Before TTD Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

Table Pointer: 0 0 0 1 V1400

Destination: 0 5 0 0 V1500

SP56 — SP56 = OFF

After TTD Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

Table Pointer (Automatically Incremented): 0 0 0 2 V1400

Destination: 9 9 9 9 V1500

SP56 — SP56 = OFF

Scan N+5

Before TTD Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

Table Pointer: 0 0 0 5 V1400

Destination: 1 0 1 0 V1500

SP56 — SP56 = OFF

After TTD Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

Table Pointer (Automatically Incremented): 0 0 0 6 V1400

Destination: 2 0 4 6 V1500

SP56 — SP56 = ON until end of scan or next instruction that uses SP56

Scan N+6

Before TTD Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

Table Pointer: 0 0 0 6 V1400

Destination: 2 0 4 6 V1500

SP56 — SP56 = OFF

After TTD Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

Table Pointer (Resets to 1, not 0): 0 0 0 1 V1400

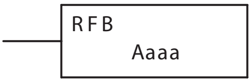
Destination: 0 5 0 0 V1500

SP56 — SP56 = OFF

Remove from Bottom (RFB)

DS	Used
HPP	Used

The Remove From Bottom instruction moves a value from the bottom of a V-memory table to a V-memory location and decrements a table pointer by 1. The first V-memory location in the table contains the table pointer which indicates the next location in the table to be moved. The instruction will be executed once per scan provided the input remains on. The instruction will stop operation when the pointer equals 0. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Remove From Bottom function.



Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table blank is used as the table pointer.) This parameter must be a HEX value.

Step 3: Insert the RFB instruction which specifies destination V-memory location (Vaaa).

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Helpful Hint: — The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one-shot (PD) should be used in the input logic.

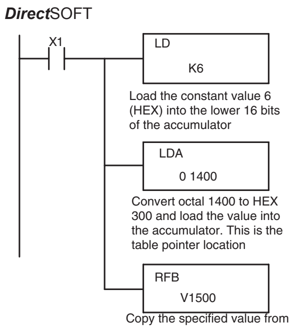
Helpful Hint: — The pointer location should be set to the value where the table operation will begin. The special relay SP0 or a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP56	On when the table pointer equals zero..

***NOTE**: Status flags (SPs) are only valid until another instruction that uses the same flag is executed or the end of the scan The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You must load a value into the pointer somewhere in your program.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the source table and is loaded into the accumulator. Remember, V1400 is used as the pointer location, and is not actually part of the table data source. The destination location (V1500) is specified in the Remove From Bottom. The table pointer (V1400 in this case) will be decremented by "1" after each execution of the RFB instruction.

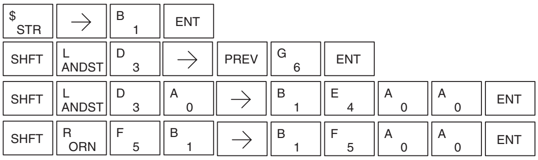


Load the constant value 6 (HEX) into the lower 16 bits of the accumulator

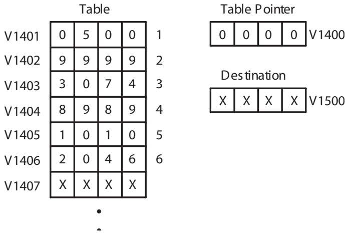
Convert octal 1400 to HEX 300 and load the value into the accumulator. This is the table pointer location

Copy the specified value from the table to the specified destination (V1500)

Handheld Programmer Keystrokes


$ STR | → | B 1 | ENT
SHFT | L ANDST | D 3 | → | PREV | G 6 | ENT
SHFT | L ANDST | D 3 | A 0 | → | B 1 | E 4 | A 0 | A 0 | ENT
SHFT | R ORN | F 5 | B 1 | → | B 1 | F 5 | A 0 | A 0 | ENT

It is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the first data location, V1401, will be used when the pointer is equal to one. The second data location, V1402, will be used when the pointer is equal to two, etc.



Table

V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Pointer

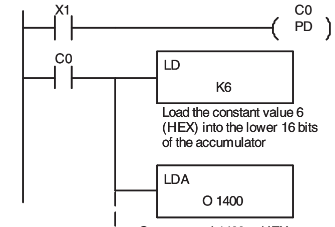
0	0	0	0	V1400

Destination

X	X	X	X	V1500

Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer decrements automatically, the table would cycle through the locations very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to remove one value each time the input contact transitions from low to high.

DirectSOFT (optional one-shot metod)

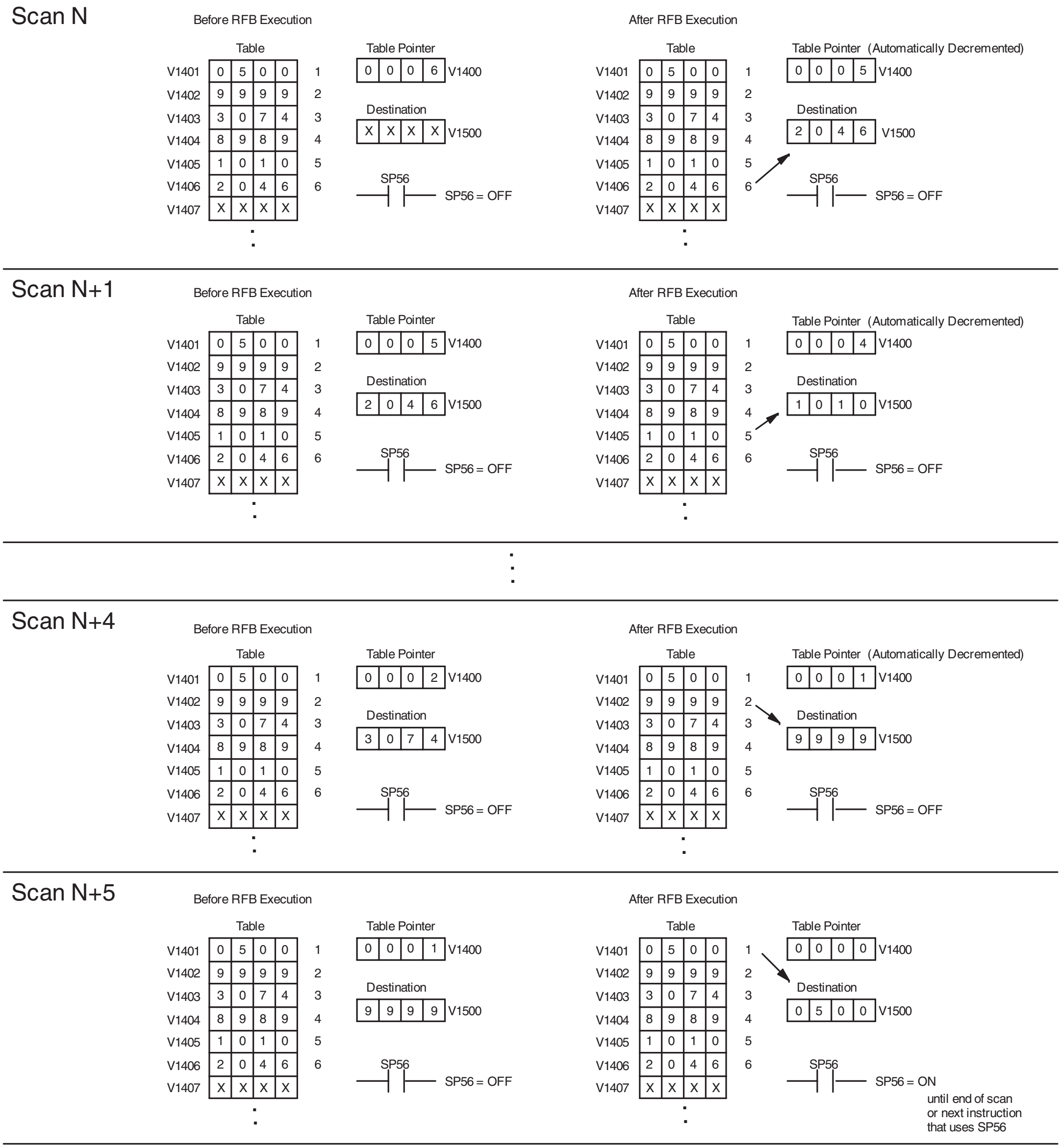


Load the constant value 6 (HEX) into the lower 16 bits of the accumulator

Convert octal 1400 to HEX 300 and load the value into the accumulator. This is the table pointer location.

The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically decrements from 6 to 0. Also, notice how SP56 is only on until the end of the scan.

Example of Execution



Scan N

Before RFB Execution

	Table				
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Pointer: 0 0 0 6 V1400

Destination: X X X X V1500

SP56 — SP56 = OFF

After RFB Execution

	Table				
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Pointer (Automatically Decremented): 0 0 0 5 V1400

Destination: 2 0 4 6 V1500

SP56 — SP56 = OFF

Scan N+1

Before RFB Execution

	Table				
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Pointer: 0 0 0 5 V1400

Destination: 2 0 4 6 V1500

SP56 — SP56 = OFF

After RFB Execution

	Table				
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Pointer (Automatically Decremented): 0 0 0 4 V1400

Destination: 1 0 1 0 V1500

SP56 — SP56 = OFF

Scan N+4

Before RFB Execution

	Table				
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Pointer: 0 0 0 2 V1400

Destination: 3 0 7 4 V1500

SP56 — SP56 = OFF

After RFB Execution

	Table				
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Pointer (Automatically Decremented): 0 0 0 1 V1400

Destination: 9 9 9 9 V1500

SP56 — SP56 = OFF

Scan N+5

Before RFB Execution

	Table				
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Pointer: 0 0 0 1 V1400

Destination: 9 9 9 9 V1500

SP56 — SP56 = OFF

After RFB Execution

	Table				
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Pointer: 0 0 0 0 V1400

Destination: 0 5 0 0 V1500

SP56 — SP56 = ON
until end of scan
or next instruction
that uses SP56

Source to Table (STT)

DS	Used
HPP	Used

The Source To Table instruction moves a value from a V-memory location into a V-memory table and increments a table pointer by 1. When the table pointer reaches the end of the table, it resets to 1. The first V-memory location in the table contains the table pointer which indicates the next location in the table to store a value. The instruction will be executed once per scan, provided the input remains on. The function parameters are loaded into the first level of the accumulator stack and the accumulator with two additional instructions.y

Listed below are the steps necessary to program the Source To Table function.

Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table pointer.) This parameter must be a HEX value.

Step 3: Insert the STT instruction which specifies the source V-memory location (Vaaa). This is where the value will be moved from.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Helpful Hint:— The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one-shot (PD) should be used in the input logic.

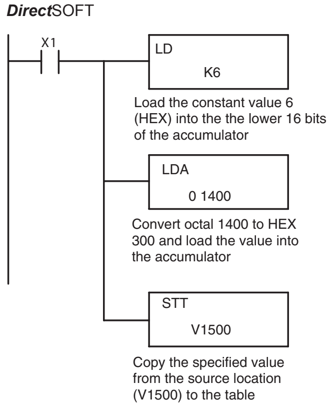
Helpful Hint: — The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the pointer should be between 0 and 6. If the value is outside of this range, the data will not be moved. Also, a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP56	On when the table pointer equals the table length.

***NOTE**: Status flags (SPs) are only valid until another instruction that uses the same flag is executed, or the end of the scan. The pointer for this instruction starts at 0 and resets to 1 automatically when the table length is reached.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400), which is the starting location for the destination table and table pointer, is loaded into the accumulator. The data source location (V1500) is specified in the Source to Table instruction. The table pointer will be increased by “1” after each time the instruction is executed.



Load the constant value 6 (HEX) into the the lower 16 bits of the accumulator

Convert octal 1400 to HEX 300 and load the value into the accumulator

Copy the specified value from the source location (V1500) to the table

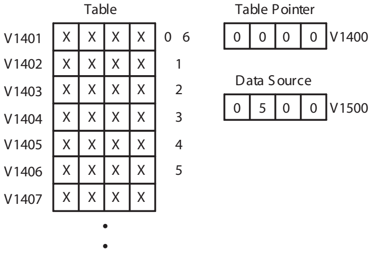
Handheld Programmer Keystrokes

$ STR | → | B 1 | ENT

SHFT | L ANDST | D 3 | → | PREV | G 6 | ENT

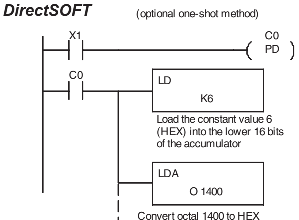
SHFT | L ANDST | D 3 | A 0 | → | B 1 | E 4 | A 0 | A 0 | ENT

SHFT | S RST | SHFT | T MLR | T MLR | → | B 1 | F 5 | A 0 | A 0 | ENT



It is important to understand how the table locations are numbered. If you examine the example table, you’ll notice that the first data storage location, V1401, will be used when the pointer is equal to zero, and again when the pointer is equal to six. Why? Because the pointer is only equal to zero before the very first execution. From then on, it increments from one to six, and then resets to one.

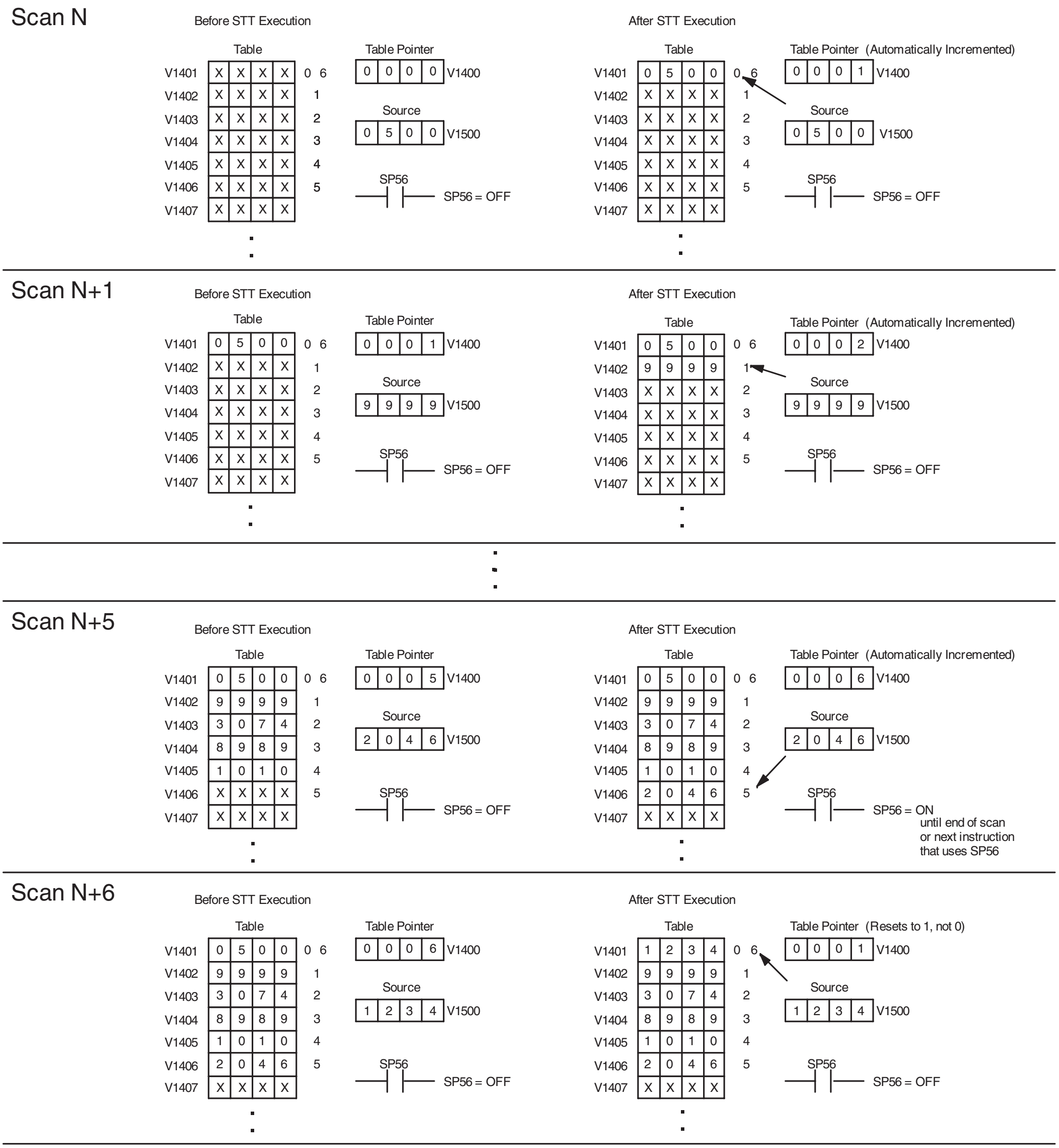
Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer increments automatically, the source data would be moved into all the table locations very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to move one value each time the input contact transitions from low to high.



Load the constant value 6 (HEX) into the lower 16 bits of the accumulator

Convert octal 1400 to HEX 300 and load the value into the accumulator. This is the starting table location.

The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically cycles from 0 to 6, and then starts over at 1 instead of 0. Also, notice how SP56 is affected by the execution. Although our example does not show it, we are assuming that there is another part of the program that changes the value in V1500 (data source) prior to the execution of the STT instruction. This is not required, but it makes it easier to see how the data source is copied into the table.



Scan N

Before STT Execution

Table

V1401	X	X	X	X	0 6
V1402	X	X	X	X	1
V1403	X	X	X	X	2
V1404	X	X	X	X	3
V1405	X	X	X	X	4
V1406	X	X	X	X	5
V1407	X	X	X	X	

Table Pointer: 0 0 0 0 V1400

Source: 0 5 0 0 V1500

SP56: SP56 = OFF

After STT Execution

Table

V1401	0	5	0	0	0 6
V1402	X	X	X	X	1
V1403	X	X	X	X	2
V1404	X	X	X	X	3
V1405	X	X	X	X	4
V1406	X	X	X	X	5
V1407	X	X	X	X	

Table Pointer (Automatically Incremented): 0 0 0 1 V1400

Source: 0 5 0 0 V1500

SP56: SP56 = OFF

Scan N+1

Before STT Execution

Table

V1401	0	5	0	0	0 6
V1402	X	X	X	X	1
V1403	X	X	X	X	2
V1404	X	X	X	X	3
V1405	X	X	X	X	4
V1406	X	X	X	X	5
V1407	X	X	X	X	

Table Pointer: 0 0 0 1 V1400

Source: 9 9 9 9 V1500

SP56: SP56 = OFF

After STT Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	X	X	X	X	2
V1404	X	X	X	X	3
V1405	X	X	X	X	4
V1406	X	X	X	X	5
V1407	X	X	X	X	

Table Pointer (Automatically Incremented): 0 0 0 2 V1400

Source: 9 9 9 9 V1500

SP56: SP56 = OFF

Scan N+5

Before STT Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	X	X	X	X	5
V1407	X	X	X	X	

Table Pointer: 0 0 0 5 V1400

Source: 2 0 4 6 V1500

SP56: SP56 = OFF

After STT Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

Table Pointer (Automatically Incremented): 0 0 0 6 V1400

Source: 2 0 4 6 V1500

SP56: SP56 = ON until end of scan or next instruction that uses SP56

Scan N+6

Before STT Execution

Table

V1401	0	5	0	0	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

Table Pointer: 0 0 0 6 V1400

Source: 1 2 3 4 V1500

SP56: SP56 = OFF

After STT Execution

Table

V1401	1	2	3	4	0 6
V1402	9	9	9	9	1
V1403	3	0	7	4	2
V1404	8	9	8	9	3
V1405	1	0	1	0	4
V1406	2	0	4	6	5
V1407	X	X	X	X	

Table Pointer (Resets to 1, not 0): 0 0 0 1 V1400

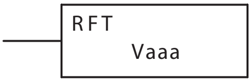
Source: 1 2 3 4 V1500

SP56: SP56 = OFF

Remove from Table (RFT)

DS	Used
HPP	Used

The Remove From Table instruction pops a value off a table and stores it in a V-memory location. When a value is removed from the table all other values are shifted up 1 location. The first V-memory location in the table contains the table length counter. The table counter decrements by 1 each time the instruction is executed. If the length counter is zero or greater than the maximum table length (specified in the first level of the accumulator stack) the instruction will not execute and SP56 will be On.



The instruction will be executed once per scan, provided the input remains on. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Remove From Table function.

Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table length counter.) This parameter must be a HEX value.

Step 3: Insert the RFT instructions which specifies destination V-memory location (Vaaa). This is where the value will be moved to.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Helpful Hint:— The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one-shot (PD) should be used in the input logic.

Helpful Hint: — The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the table counter should be between 1 and 6. If the value is outside of this range or zero, the data will not be moved from the table. Also, a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

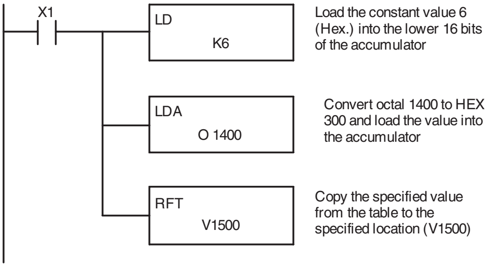
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP56	On when the table pointer equals zero.

***NOTE**: Status flags (SPs) are only valid until another instruction that uses the same flag is executed, or the end of the scan The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You must load a value into the pointer somewhere in your program.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the source table and is loaded into the accumulator. The destination location (V1500) is specified in the Remove from Table instruction. The table counter will be decreased by "1" after the instruction is executed.

***Direct*SOFT**

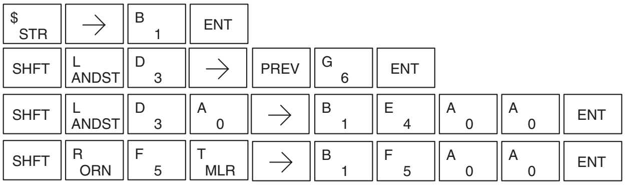


Load the constant value 6 (Hex.) into the lower 16 bits of the accumulator

Convert octal 1400 to HEX 300 and load the value into the accumulator

Copy the specified value from the table to the specified location (V1500)

Handheld Programmer Keystrokes



Since the table counter specifies the range of data that will be removed from the table, it is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the data locations are numbered from the top of the table. For example, if the table counter started at 6, then all six of the locations would be affected during the instruction execution.

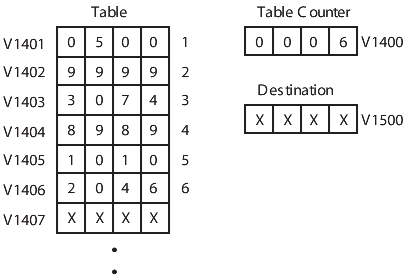


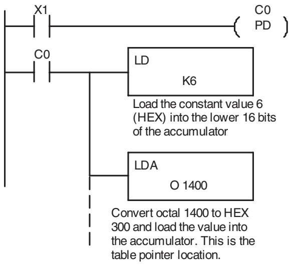
	Table				
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Counter				
0	0	0	6	V1400

Destination				
X	X	X	X	V1500

Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer decrements automatically, the data would be removed from the table very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to remove one value each time the input contact transitions from low to high.

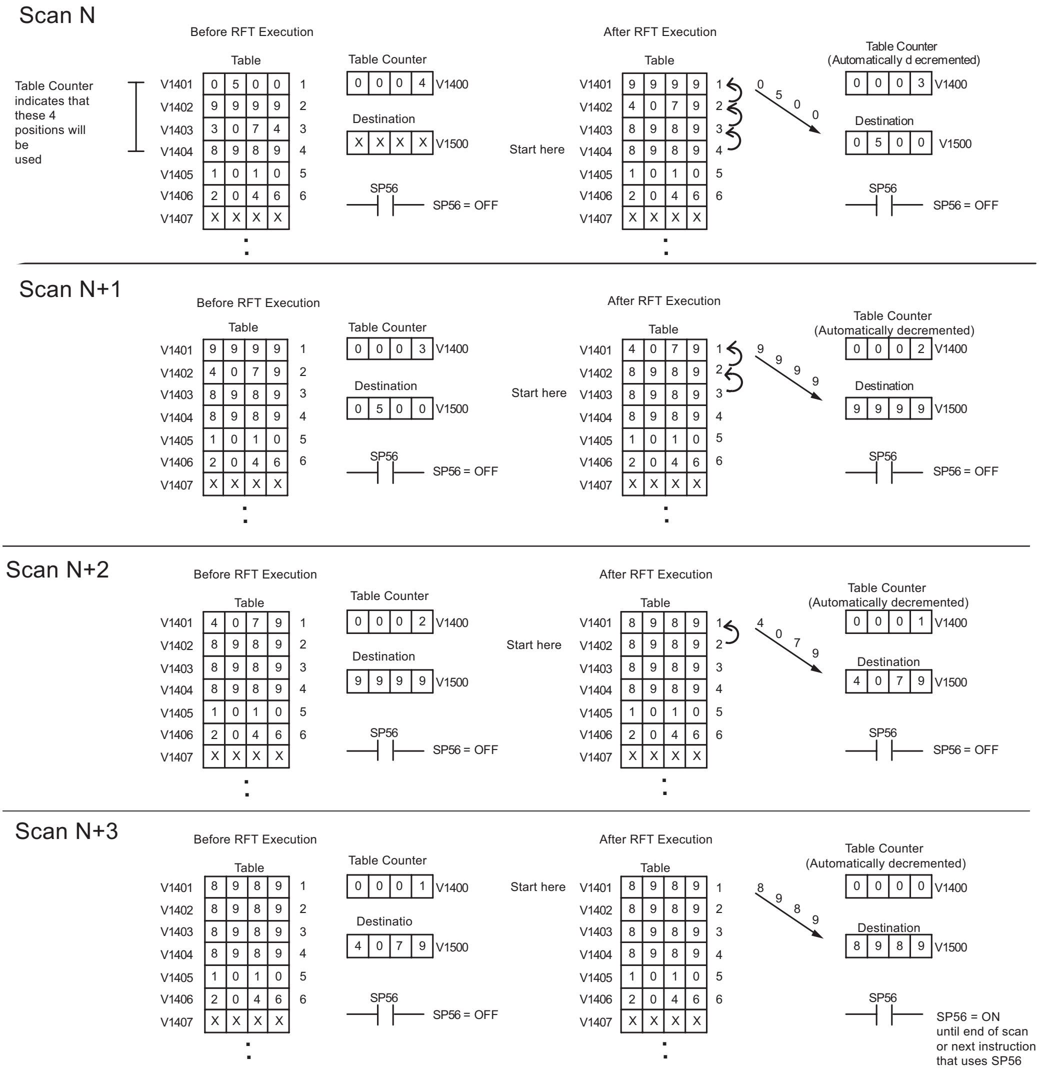
DirectSOFT32 Display (optional one-shot method)



Load the constant value 6 (HEX) into the lower 16 bits of the accumulator

Convert octal 1400 to HEX 300 and load the value into the accumulator. This is the table pointer location.

The following diagram shows the scan-by-scan results of the execution for our example program. In our example, we show the table counter set to 4, initially. (Remember, you can set the table counter to any value that is within the range of the table.) The table counter automatically decrements from 4 to 0 as the instruction is executed. Notice how the last two table positions, 5 and 6, are not moved up through the table. Also, notice that SP56, which comes on when the table counter is zero, is only on until the end of the scan.



Scan N

Before RFT Execution: Table						After RFT Execution: Table					
V1401	0	5	0	0	1	V1401	9	9	9	9	1
V1402	9	9	9	9	2	V1402	4	0	7	9	2
V1403	3	0	7	4	3	V1403	8	9	8	9	3
V1404	8	9	8	9	4	V1404	8	9	8	9	4
V1405	1	0	1	0	5	V1405	1	0	1	0	5
V1406	2	0	4	6	6	V1406	2	0	4	6	6
V1407	X	X	X	X		V1407	X	X	X	X	

Before: Table Counter 0 0 0 4 V1400; Destination X X X X V1500; SP56 = OFF
After: Table Counter (Automatically d ecremented) 0 0 0 3 V1400; Destination 0 5 0 0 V1500; SP56 = OFF

Scan N+1

Before RFT Execution: Table						After RFT Execution: Table					
V1401	9	9	9	9	1	V1401	4	0	7	9	1
V1402	4	0	7	9	2	V1402	8	9	8	9	2
V1403	8	9	8	9	3	V1403	8	9	8	9	3
V1404	8	9	8	9	4	V1404	8	9	8	9	4
V1405	1	0	1	0	5	V1405	1	0	1	0	5
V1406	2	0	4	6	6	V1406	2	0	4	6	6
V1407	X	X	X	X		V1407	X	X	X	X	

Before: Table Counter 0 0 0 3 V1400; Destination 0 5 0 0 V1500; SP56 = OFF
After: Table Counter (Automatically decremented) 0 0 0 2 V1400; Destination 9 9 9 9 V1500; SP56 = OFF

Scan N+2

Before RFT Execution: Table						After RFT Execution: Table					
V1401	4	0	7	9	1	V1401	8	9	8	9	1
V1402	8	9	8	9	2	V1402	8	9	8	9	2
V1403	8	9	8	9	3	V1403	8	9	8	9	3
V1404	8	9	8	9	4	V1404	8	9	8	9	4
V1405	1	0	1	0	5	V1405	1	0	1	0	5
V1406	2	0	4	6	6	V1406	2	0	4	6	6
V1407	X	X	X	X		V1407	X	X	X	X	

Before: Table Counter 0 0 0 2 V1400; Destination 9 9 9 9 V1500; SP56 = OFF
After: Table Counter (Automatically decremented) 0 0 0 1 V1400; Destination 4 0 7 9 V1500; SP56 = OFF

Scan N+3

Before RFT Execution: Table						After RFT Execution: Table					
V1401	8	9	8	9	1	V1401	8	9	8	9	1
V1402	8	9	8	9	2	V1402	8	9	8	9	2
V1403	8	9	8	9	3	V1403	8	9	8	9	3
V1404	8	9	8	9	4	V1404	8	9	8	9	4
V1405	1	0	1	0	5	V1405	1	0	1	0	5
V1406	2	0	4	6	6	V1406	2	0	4	6	6
V1407	X	X	X	X		V1407	X	X	X	X	

Before: Table Counter 0 0 0 1 V1400; Destinatio 4 0 7 9 V1500; SP56 = OFF
After: Table Counter (Automatically decremented) 0 0 0 0 V1400; Destination 8 9 8 9 V1500; SP56 = ON until end of scan or next instruction that uses SP56

Add to Top (ATT)

DS	Used
HPP	Used

The Add To Top instruction pushes a value on to a V-memory table from a V-memory location. When the value is added to the table all other values are pushed down 1 location.

The instruction will be executed once per scan, provided the input remains on. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Add To Top function.

Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table length counter.) This parameter must be a HEX value.

Step 3: Insert the ATT instructions which specifies source V-memory location (Vaaa). This is where the value will be moved from.

Helpful Hint:— The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one-shot (PD) should be used in the input logic.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

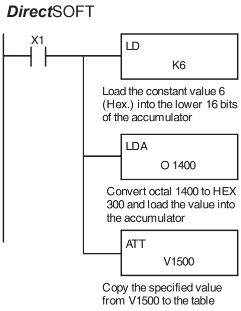
Helpful Hint: — The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the table counter should be between 1 and 6. If the value is outside of this range or zero, the data will not be moved into the table. Also, a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP56	On when the table pointer equal to the table size.

NOTE: *Status flags (SPs) are only valid until another instruction that uses the same flag is executed or the end of the scan. The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You must load a value into the pointer somewhere in your program.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400), which is the starting location for the destination table and table counter, is loaded into the accumulator. The source location (V1500) is specified in the Add to Top instruction. The table counter will be increased by "1" after the instruction is executed.


*Direct*SOFT
X1
LD
K6
Load the constant value 6 (Hex.) into the lower 16 bits of the accumulator
LDA
O 1400
Convert octal 1400 to HEX 300 and load the value into the accumulator
ATT
V1500
Copy the specified value from V1500 to the table

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	PREV	G 6	ENT			
SHFT	L ANDST	D 3	A 0	→	B 1	E 4	A 0	A 0	ENT
SHFT	A 0	T MLR	T MLR	→	B 1	F 5	A 0	A 0	ENT

For the ATT instruction, the table counter determines the number of additions that can be made before the instruction will stop executing. So, it is helpful to understand how the system uses this counter to control the execution.

For example, if the table counter was set to 2, and the table length was 6 words, then there could only be 4 additions of data before the execution was stopped. This can easily be calculated by:

Table length – table counter = number of executions

Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the table counter increments automatically, the data would be moved into the table very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to add one value each time the input contact transitions from low to high.

Table

V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

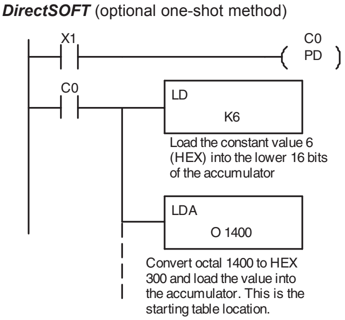
Table Counter

0	0	0	2	V1400

Data Source

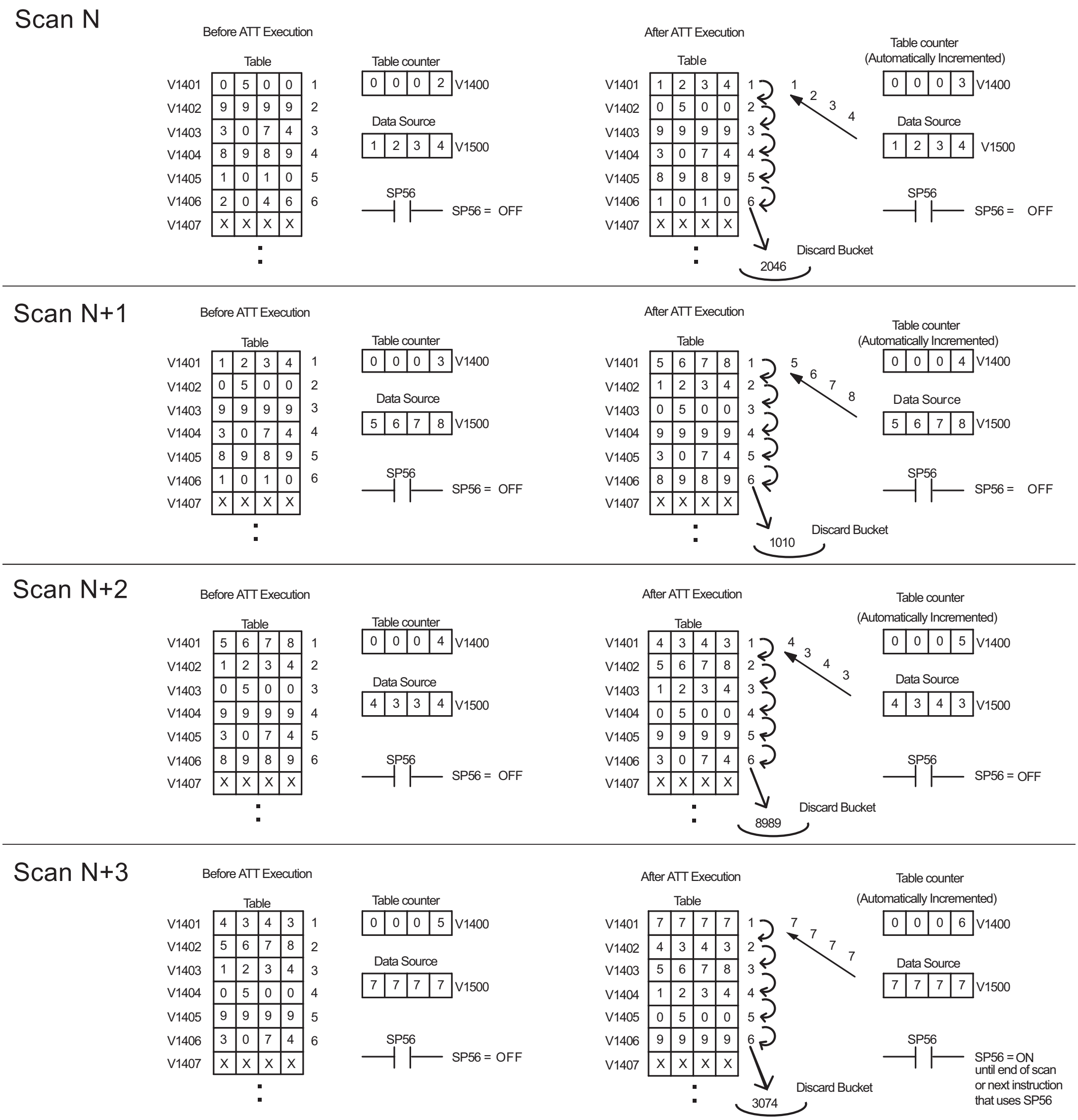
X	X	X	X	V1500

(e.g.: 6 - 2 = 4)


***Direct*SOFT** (optional one-shot method)
X1
C0
PD
C0
LD
K6
Load the constant value 6 (HEX) into the lower 16 bits of the accumulator
LDA
O 1400
Convert octal 1400 to HEX 300 and load the value into the accumulator. This is the starting table location.

The following diagram shows the scan-by-scan results of the execution for our example program. The table counter is set to 2 initially, and it will automatically increment from 2 to 6 as the instruction is executed. Notice how SP56 comes on when the table counter is 6, which is equal to the table length. Plus, although our example does not show it, we are assuming that there is another part of the program that changes the value in V1500 (data source) prior to the execution of the ATT instruction.

Example of Execution



Scan N

Before ATT Execution

	Table				
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table counter: 0 0 0 2 V1400

Data Source: 1 2 3 4 V1500

SP56 — SP56 = OFF

After ATT Execution

	Table				
V1401	1	2	3	4	1
V1402	0	5	0	0	2
V1403	9	9	9	9	3
V1404	3	0	7	4	4
V1405	8	9	8	9	5
V1406	1	0	1	0	6
V1407	X	X	X	X	

Table counter (Automatically Incremented): 0 0 0 3 V1400

Data Source: 1 2 3 4 V1500

SP56 — SP56 = OFF

Scan N+1

Before ATT Execution

	Table				
V1401	1	2	3	4	1
V1402	0	5	0	0	2
V1403	9	9	9	9	3
V1404	3	0	7	4	4
V1405	8	9	8	9	5
V1406	1	0	1	0	6
V1407	X	X	X	X	

Table counter: 0 0 0 3 V1400

Data Source: 5 6 7 8 V1500

SP56 — SP56 = OFF

After ATT Execution

	Table				
V1401	5	6	7	8	1
V1402	1	2	3	4	2
V1403	0	5	0	0	3
V1404	9	9	9	9	4
V1405	3	0	7	4	5
V1406	8	9	8	9	6
V1407	X	X	X	X	

Table counter (Automatically Incremented): 0 0 0 4 V1400

Data Source: 5 6 7 8 V1500

SP56 — SP56 = OFF

Scan N+2

Before ATT Execution

	Table				
V1401	5	6	7	8	1
V1402	1	2	3	4	2
V1403	0	5	0	0	3
V1404	9	9	9	9	4
V1405	3	0	7	4	5
V1406	8	9	8	9	6
V1407	X	X	X	X	

Table counter: 0 0 0 4 V1400

Data Source: 4 3 3 4 V1500

SP56 — SP56 = OFF

After ATT Execution

	Table				
V1401	4	3	4	3	1
V1402	5	6	7	8	2
V1403	1	2	3	4	3
V1404	0	5	0	0	4
V1405	9	9	9	9	5
V1406	3	0	7	4	6
V1407	X	X	X	X	

Table counter (Automatically Incremented): 0 0 0 5 V1400

Data Source: 4 3 4 3 V1500

SP56 — SP56 = OFF

Scan N+3

Before ATT Execution

	Table				
V1401	4	3	4	3	1
V1402	5	6	7	8	2
V1403	1	2	3	4	3
V1404	0	5	0	0	4
V1405	9	9	9	9	5
V1406	3	0	7	4	6
V1407	X	X	X	X	

Table counter: 0 0 0 5 V1400

Data Source: 7 7 7 7 V1500

SP56 — SP56 = OFF

After ATT Execution

	Table				
V1401	7	7	7	7	1
V1402	4	3	4	3	2
V1403	5	6	7	8	3
V1404	1	2	3	4	4
V1405	0	5	0	0	5
V1406	9	9	9	9	6
V1407	X	X	X	X	

Table counter (Automatically Incremented): 0 0 0 6 V1400

Data Source: 7 7 7 7 V1500

SP56 — SP56 = ON until end of scan or next instruction that uses SP56

Table Shift Left (TSHFL)

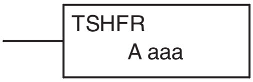
DS	Used
HPP	Used

The Table Shift Left instruction shifts all the bits in a V-memory table to the left, the specified number of bit positions.

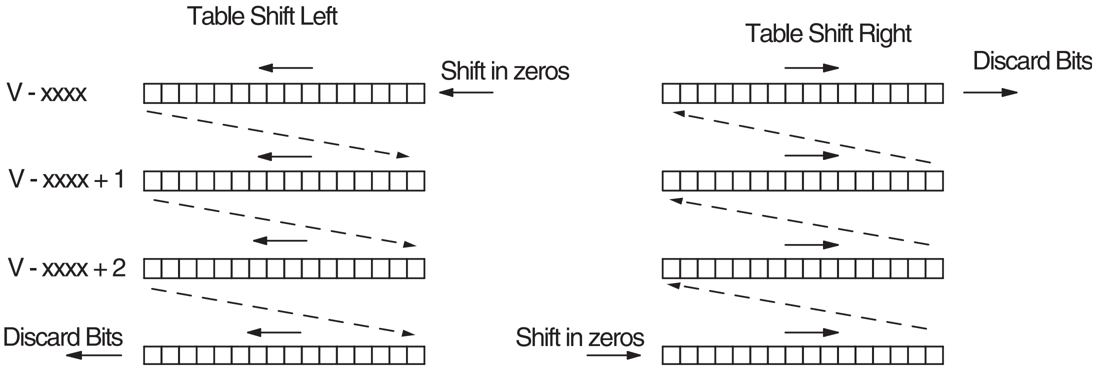
Table Shift Right (TSHFR)

DS	Used
HPP	Used

The Table Shift Right instruction shifts all the bits in a V-memory table to the right, a specified number of bit positions.



The following description applies to both the Table Shift Left and Table Shift Right instructions. A table is just a range of V-memory locations. The Table Shift Left and Table Shift Right instructions shift bits serially throughout the entire table. Bits are shifted out the end of one word and into the opposite end of an adjacent word. At the ends of the table, bits are either discarded, or zeros are shifted into the table. The example tables below are arbitrarily four words long.



Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: Insert the Table Shift Left or Table shift Right instruction. This specifies the number of bit positions you wish to shift the entire table. The number of bit positions must be in octal.

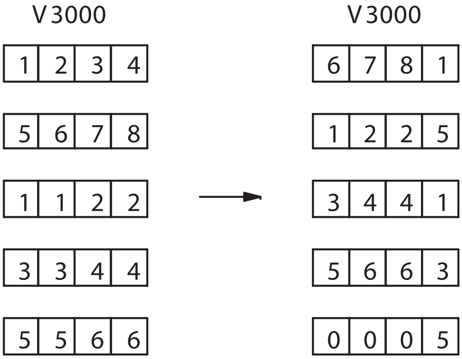
Helpful hint: — Remember that each V-memory location contains 16 bits. So, the bits of the first word of the table are numbered from 0 to 17 octal. If you want to shift the entire table by 20 bits, that is 24 octal. SP 53 will be set if the number of bits to be shifted is larger than the total bits contained within the table. Flag 67 will be set if the last bit shifted (just before it is discarded) is a “1”.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP53	On when the number of bits to be shifted is larger than the total bits contained within the table
SP67	On when the last bit shifted (just before it is discarded) is a **1**

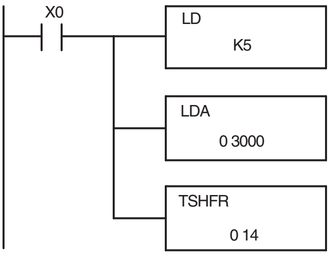
NOTE: *Status flags are only valid until the end of the scan or another instruction that uses the same flag is executed.*

The example table to the right contains BCD data as shown (for demonstration purposes). Suppose we want to do a table shift right by 3 BCD digits (12 bits). Converting to octal, 12 bits is 14 octal. Using the Table Shift Right instruction and specifying a shift by octal 14, we have the resulting table shown at the far right. Notice that the 2–3–4 sequence has been discarded, and the 0–0–0 sequence has been shifted in at the bottom.



The following ladder example assumes the data at V3000 to V3004 already exists as shown above. We will use input X0 to trigger the Table Shift Right operation. First, we will load the table length (5 words) into the accumulator stack. Next, we load the starting address into the accumulator. Since V3000 is an octal number, we have to convert it to hex by using the LDA command. Finally, we use the Table Shift Right instruction and specify the number of bits to be shifted (12 decimal), which is 14 octal.

DirectSOFT



Load the constant value 5 (Hex.) into the lower 16 bits of the accumulator.

Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning.

Do a table shift right by 12 bits, which is 14 octal.

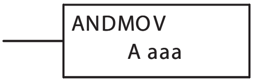
Handheld Programmer Keystrokes

$ STR	→	A 0	ENT							
SHFT	L ANDST	D 3	→	PREV	F 5	ENT				
SHFT	L ANDST	D 3	A 0	→	D 3	A 0	A 0	A 0	ENT	
SHFT	T MLR	SHFT	S RST	H 7	F 5	R ORN	→	NEXT	B 1	E 4	ENT

AND Move (ANDMOV)

DS	Used
HPP	Used

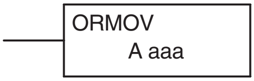
The AND Move instruction copies data from a table to the specified memory location, ANDing each word with the accumulator data as it is written.



OR Move (ORMOV)

DS	Used
HPP	Used

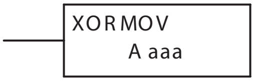
The OR Move instruction copies data from a table to the specified memory location, ORing each word with the accumulator contents as it is written



Exclusive OR Move (XORMOV)

DS	Used
HPP	Used

The Exclusive OR Move instruction copies data from a table to the specified memory location, XORing each word with the accumulator value as it is written.



The following description applies to the AND Move, OR Move, and Exclusive OR Move instructions. A table is just a range of V-memory locations. These instructions copy the data of a table to another specified location, preforming a logical operation on each word with the accumulator contents as the new table is written.

Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

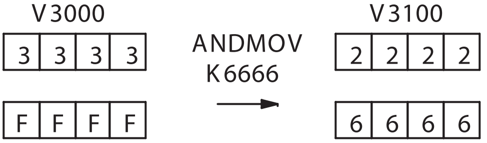
Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: Load the BCD/hex bit pattern into the accumulator which will be logically combined with the table contents as they are copied.

Step 4: Insert the AND Move, OR Move, or XOR Move instruction. This specifies the starting location of the copy of the original table. This new table will automatically be the same length as the original table.

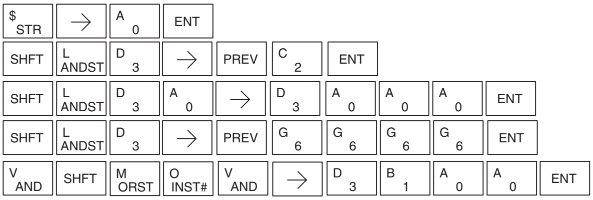
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

The example table to the right contains BCD data as shown (for demonstration purposes). Suppose we want to move a table of two words at V3000 and AND it with K6666. The copy of the table at V3100 shows the result of the AND operation for each word.

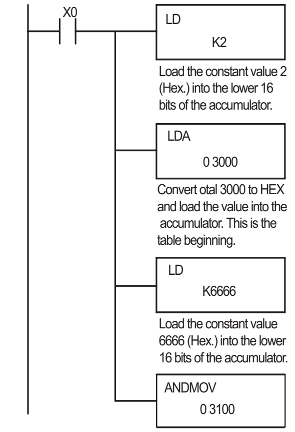


The program on the next page performs the ANDMOV operation example above. It assumes that the data in the table at V3000 – V3001 already exists. First we load the table length (two words) into the accumulator. Next we load the starting address of the source table, using the LDA instruction. Then we load the data into the accumulator to be ANDed with the table. In the ANDMOV command, we specify the table destination, V3100.

Handheld Programmer Keystrokes


$ STR | → | A 0 | ENT
SHFT | L ANDST | D 3 | → | PREV | C 2 | ENT
SHFT | L ANDST | D 3 | A 0 | → | D 3 | A 0 | A 0 | A 0 | ENT
SHFT | L ANDST | D 3 | → | PREV | G 6 | G 6 | G 6 | G 6 | ENT
V AND | SHFT | M ORST | O INST# | V AND | → | D 3 | B 1 | A 0 | A 0 | ENT

DirectSOFT



Load the constant value 2 (Hex.) into the lower 16 bits of the accumulator.

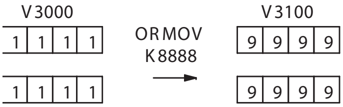
Convert otal 3000 to HEX and load the value into the accumulator. This is the table beginning.

Load the constant value 6666 (Hex.) into the lower 16 bits of the accumulator.

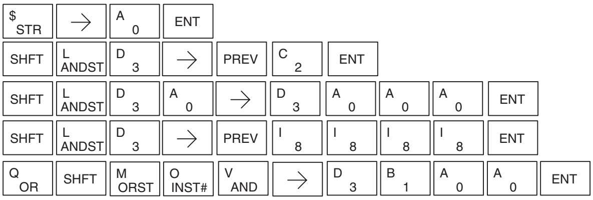
Copy the table to V3100, ANDing its contents with the accumulator as it is written.

The example on top the right shows a table of two words at V3000 and logically ORs it with K8888. The copy of the table at V3100 shows the result of the OR operation for each word.

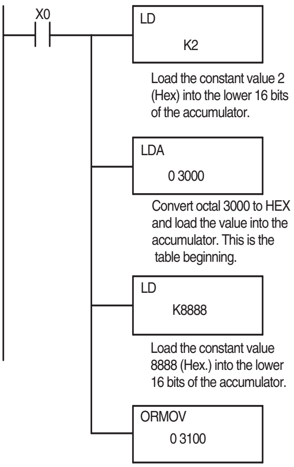
The program to the right performs the ORMOV example above. It assumes that the data in the table at V3000 – V3001 already exists. First we load the table length (two words) into the accumulator. Next we load the starting address of the source table, using the LDA instruction. Then we load the data into the accumulator to be ORed with the table. In the ORMOV command, we specify the table destination, V3100.



Handheld Programmer Keystrokes


$ STR | → | A 0 | ENT
SHFT | L ANDST | D 3 | → | PREV | C 2 | ENT
SHFT | L ANDST | D 3 | A 0 | → | D 3 | A 0 | A 0 | A 0 | ENT
SHFT | L ANDST | D 3 | → | PREV | I 8 | I 8 | I 8 | I 8 | ENT
Q OR | SHFT | M ORST | O INST# | V AND | → | D 3 | B 1 | A 0 | A 0 | ENT

DirectSOFT



Load the constant value 2 (Hex) into the lower 16 bits of the accumulator.

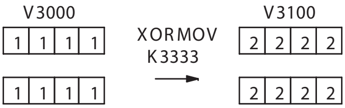
Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning.

Load the constant value 8888 (Hex.) into the lower 16 bits of the accumulator.

Copy the table to V3100, ORing its contents with the accumulator as it is written.

The example to the right shows a table of two words at V3000 and logically XORs it with K3333. The copy of the table at V3100 shows the result of the XOR operation for each word.

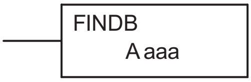
The ladder program example for the XORMOV is similar to the one above for the ORMOV. Just use the XORMOV instruction. On the handheld programmer, you must use the SHFT key and spell "XORMOV" explicitly.



Find Block (FINDB)

DS	Used
HPP	N/A

The Find Block instruction searches for an occurrence of a specified block of values in a V-memory table. The function parameters are loaded into the first and second levels of the accumulator stack and the accumulator by three additional instructions. If the block is found, its starting address will be stored in the accumulator. If the block is not found, flag SP53 will be set.



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
V-memory	P	See memory map

Discrete Bit Flags	Description
SP56	On when the specified block is not found.

The steps listed below are the steps necessary to program the Find Block function.

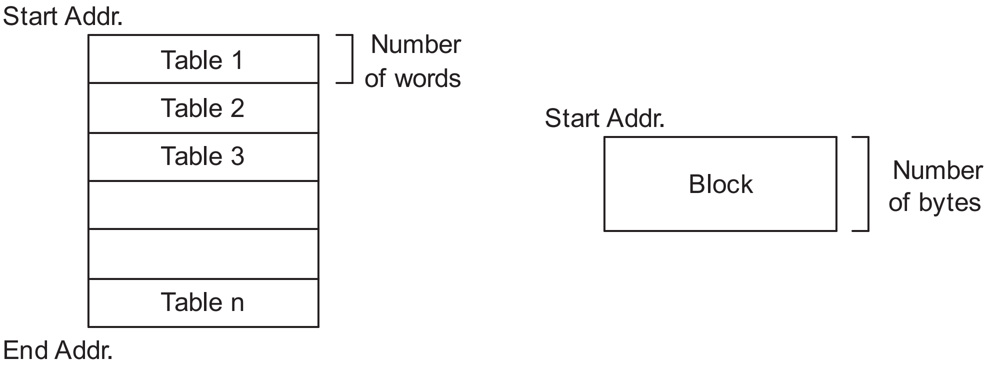
Step 1: Load the number of bytes in the block to be located. This parameter must be a HEX value, 0 to FF.

Step 2: Load the length of a table (number of words) to be searched. The Find Block will search multiple tables that are adjacent in V-memory. This parameter must be a HEX value, 0 to FF.

Step 3: Load the ending location for all the tables into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 4: Load the table starting location for all the tables into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

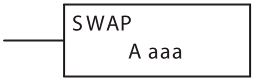
Step 5: Insert the Find Block instruction. This specifies the starting location of the block of data you are trying to locate.



Swap (SWAP)

DS	Used
HPP	Used

The Swap instruction exchanges the data in two tables of equal length.



Step 1: Load the length of the tables (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF. Remember that the tables must be of equal length.

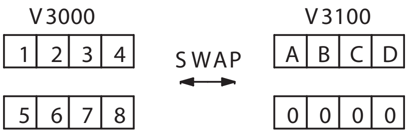
Step 2: Load the starting V-memory location for the first table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: Insert the Swap instruction. This specifies the starting address of the second table. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

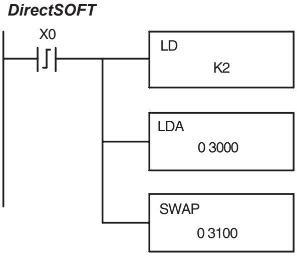
Helpful hint: — The data swap occurs within a single scan. If the instruction executes on multiple consecutive scans, it will be difficult to know the actual contents of either table at any particular time. So, remember to swap just on a single scan.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

The example to the right shows a table of two words at V3000. We will swap its contents with another table of two words at 3100 by using the Swap instruction. The required ladder program is given below.



The example program below uses a PD contact (triggers for one scan for off-to-on transition). First, we load the length of the tables (two words) into the accumulator. Then we load the address of the first table (V3000) into the accumulator using the LDA instruction, converting the octal address to hex. Note that it does not matter which table we declare “first”, because the swap results will be the same.



Load the constant value 2 (Hex.) into the lower 16 bits of the accumulator.

Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning.

Swap the contents of the table in the previous instruction with the one at V3100.

Handheld Programmer Keystrokes

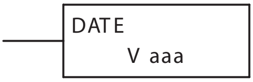
$ STR	SHFT	P CV	D 3	→	A 0	ENT					
SHFT	L ANDST	D 3	→	PREV	C 2	ENT					
SHFT	L ANDST	D 3	A 0	→	D 3	A 0	A 0	A 0	ENT		
SHFT	S RST	SHFT	W ANDN	A 0	P CV	→	D 3	B 1	A 0	A 0	ENT

Clock/Calendar Instructions

Date (DATE)

DS	Used
HPP	Used

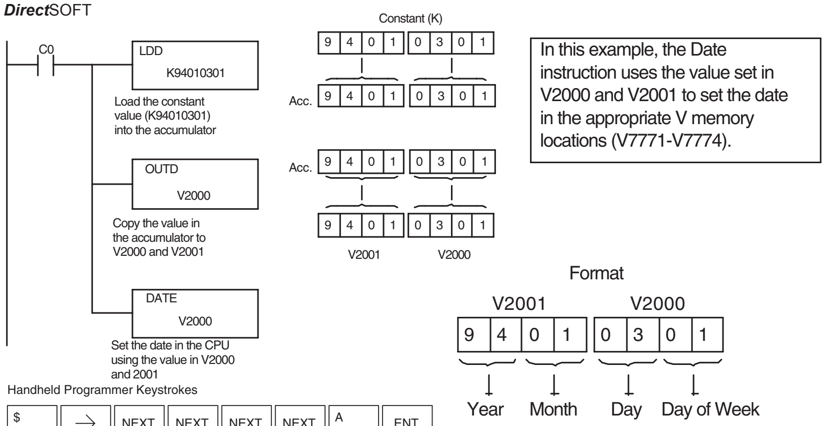
The Date instruction can be used to set the date in the CPU. The instruction requires two consecutive V-memory locations (Vaaa) to set the date. If the values in the specified locations are not valid, the date will not be set. The current date can be read from 4 consecutive V-memory locations (V7771–V7774).



In the following example, when C0 is on, the constant value (K94010301) is loaded into the accumulator using the Load Double instruction (C0 should be a contact from a one-shot (PD) instruction). The value in the accumulator is output to V2000 using the Out Double instruction. The Date instruction uses the value in V2000 to set the date in the CPU.

Date	Range	V-memory Location (BCD) (READ Only)
Year	0-99	V7774
Month	1-12	V7773
Day	1-31	V7772
Day of Week	0-06	V7771
The values entered for the day of week are: 0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday		

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map



In this example, the Date instruction uses the value set in V2000 and V2001 to set the date in the appropriate V memory locations (V7771-V7774).

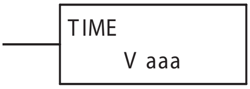
Handheld Programmer Keystrokes

$ STR	→	NEXT	NEXT	NEXT	NEXT	A 0	ENT			
SHFT	L ANDST	D 3	D 3	→	PREV	9	E 4	A 0	B 1	ENT
A 0	D 3	A 0	B 1	ENT						
GX OUT	SHFT	D 3	→	C 2	A 0	A 0	A 0	ENT		
SHFT	D 3	A 0	T MLR	E 4	→	C 2	A 0	A 0	A 0	ENT

Time (TIME)

DS	Used
HPP	Used

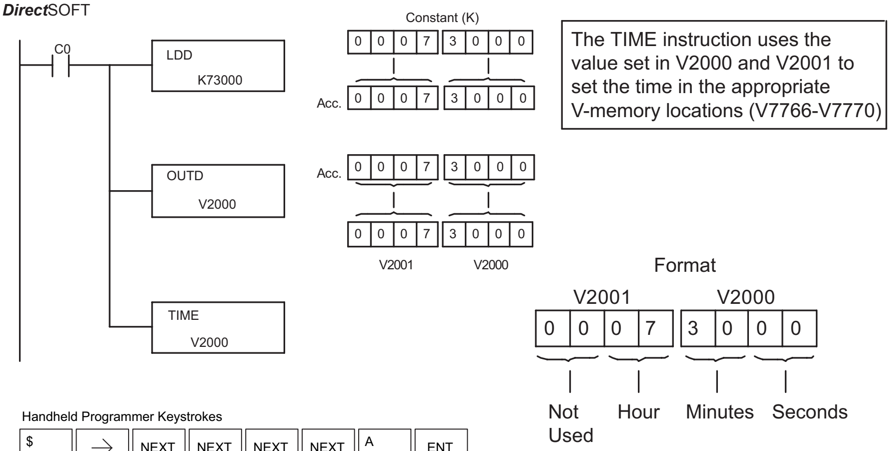
The Time instruction can be used to set the time (24 hour clock) in the CPU. The instruction requires two consecutive V-memory locations (Vaaa) which are used to set the time. If the values in the specified locations are not valid, the time will not be set. The current time can be read from memory locations V7747 and V7766–V7770.



Date	Range	VMemory Location (BCD) (READ Only)
1/100 seconds (10ms)	0-99	V7747
Seconds	0-59	V7766
Minutes	0-59	V7767
Hour	0-23	V7770

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

In the following example, when C0 is on, the constant value (K73000) is loaded into the accumulator using the Load Double instruction (C0 should be a contact from a one-shot (PD) instruction). The value in the accumulator is output to V2000 using the Out Double instruction. The Time instruction uses the value in V2000 to set the time in the CPU.



The TIME instruction uses the value set in V2000 and V2001 to set the time in the appropriate V-memory locations (V7766-V7770)

Handheld Programmer Keystrokes

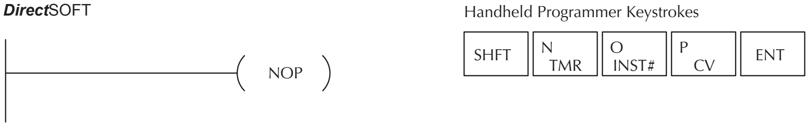
$ STR	→	NEXT	NEXT	NEXT	NEXT	A 0	ENT				
SHFT	L ANDST	D 3	D 3	→	PREV	H 7	D 3	A 0	A 0	A 0	ENT
A 0	D 3	A 0	B 1	ENT							
GX OUT	SHFT	D 3	→	C 2	A 0	A 0	A 0	ENT			
SHFT	T MLR	SHFT	I 8	M ORST	E 4	→	C 2	A 0	A 0	A 0	ENT

CPU Control Instructions

No Operation (NOP)

DS	Used
HPP	Used

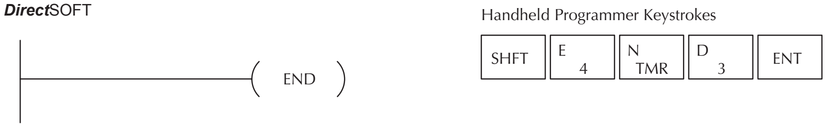
The No Operation is an empty (not programmed) memory location.



End (END)

DS	Used
HPP	Used

The End instruction marks the termination point of the normal program scan. An End instruction is required at the end of the main program body. If the End instruction is omitted, an error will occur and the CPU will not enter the Run Mode. Data labels, subroutines and interrupt routines are placed after the End instruction. The End instruction is not conditional; therefore, no input contact is allowed.

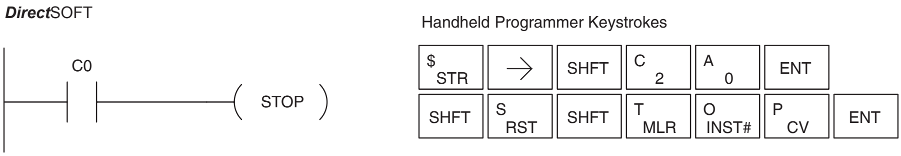


Stop (STOP)

DS	Used
HPP	Used

The Stop instruction changes the operational mode of the CPU from Run to Program (Stop) mode. This instruction is typically used to stop PLC operation in an error condition.

In the following example, when C0 turns on, the CPU will stop operation and switch to the program mode.



Discrete Bit Flags	Description
SP16	On when the DL06 goes into the TERM_PRG mode.
SP53	On when the DL06 goes into the PRG mode.

Reset Watch Dog Timer (RSTWT)

DS	Used
HPP	Used

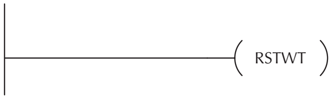
The Reset Watch Dog Timer instruction resets the CPU scan timer. The default setting for the watch dog timer is 200ms. Scan times very seldom exceed 200ms, but it is possible. For/next loops, subroutines, interrupt routines, and table instructions can be programmed such that the scan becomes longer than 200ms. When instructions are used in a manner that could exceed the watch dog timer setting, this instruction can be used to reset the timer.

A software timeout error (E003) will occur and the CPU will enter the program mode if the scan time exceeds the watch dog timer setting. Placement of the RSTWT instruction in the program is very important. The instruction has to be executed before the scan time exceeds the watch dog timer's setting.

If the scan time is consistently longer than the watch dog timer's setting, the timeout value may be permanently increased from the default value of 200ms by AUX 55 on the HPP or the appropriate auxiliary function in your programming package. This eliminates the need for the RSTWT instruction.

In the following example, the CPU scan timer will be reset to 0 when the RSTWT instruction is executed. See the For/Next instruction for a detailed example.

*Direct*SOFT



Handheld Programmer Keystrokes

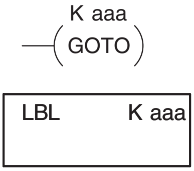
SHFT	R ORN	S RST	T MLR	W ANDN	T MLR	ENT

Program Control Instructions

Goto Label (GOTO) (LBL)

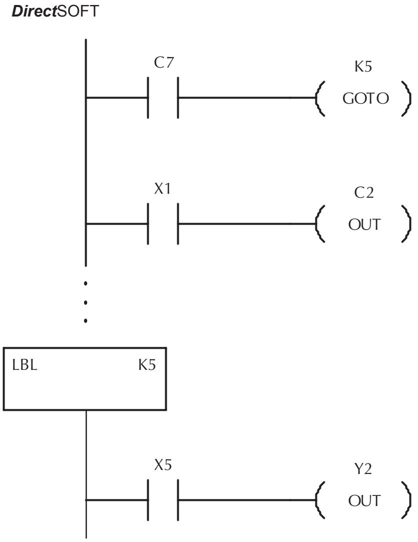
DS	Used
HPP	Used

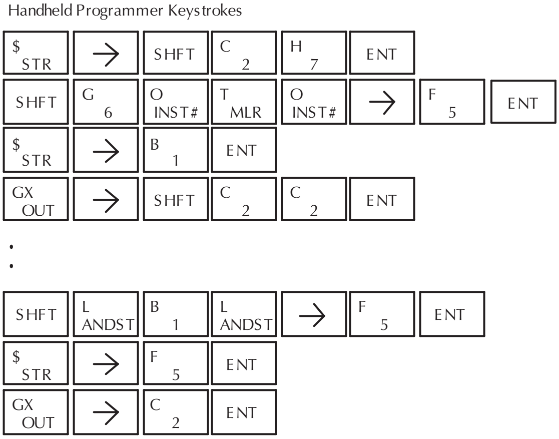
The Goto / Label skips all instructions between the Goto and the corresponding LBL instruction. The operand value for the Goto and the corresponding LBL instruction are the same. The logic between Goto and LBL instruction is not executed when the Goto instruction is enabled. Up to 256 Goto instructions and 256 LBL instructions can be used in the program.



Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

In the following example, when C7 is on, all the program logic between the GOTO and the corresponding LBL instruction (designated with the same constant Kaaa value) will be skipped. The instructions being skipped will not be executed by the CPU.




Handheld Programmer Keystrokes

$ STR | → | SHFT | C 2 | H 7 | ENT

SHFT | G 6 | O INST# | T MLR | O INST# | → | F 5 | ENT

$ STR | → | B 1 | ENT

GX OUT | → | SHFT | C 2 | C 2 | ENT

:

SHFT | L ANDST | B 1 | L ANDST | → | F 5 | ENT

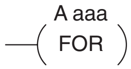
$ STR | → | F 5 | ENT

GX OUT | → | C 2 | ENT

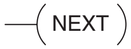
For / Next (FOR) (NEXT)

DS	Used
HPP	Used

The For and Next instructions are used to execute a section of ladder logic between the For and Next instruction a specified numbers of times. When the For instruction is enabled, the program will loop the specified number of times. If the For instruction is not energized, the section of ladder logic between the For and Next instructions is not executed.

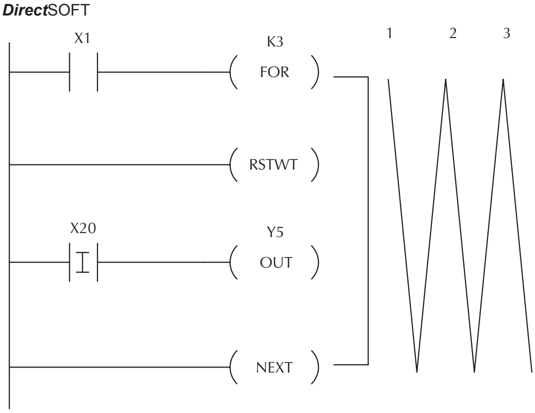


For / Next instructions cannot be nested. The normal I/O update and CPU housekeeping are suspended while executing the For / Next loop. The program scan can increase significantly, depending on the amount of times the logic between the For and Next instruction is executed. With the exception of immediate I/O instructions, I/O will not be updated until the program execution is completed for that scan. Depending on the length of time required to complete the program execution, it may be necessary to reset the watch dog timer inside of the For / Next loop using the RSTWT instruction.



Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map
Constant	K	1-9999

In the following example, when X1 is on, the application program inside the For / Next loop will be executed three times. If X1 is off, the program inside the loop will not be executed. The immediate instructions may or may not be necessary, depending on your application. Also, The RSTWT instruction is not necessary if the For / Next loop does not extend the scan time beyond the Watch Dog Timer setting. For more information on the Watch Dog Timer, refer to the RSTWT instruction.



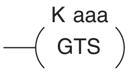
Handheld Programmer Keystrokes

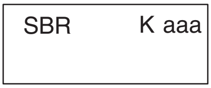
$ STR	→	B 1	ENT			
SHFT	F 5	O INST#	R ORN	→	D 3	ENT
SHFT	R ORN	S RST	T MLR	W ANDN	T MLR	ENT
$ STR	SHFT	I 8	→	C 2	A 0	ENT
GX OUT	→	F 5	ENT			
SHFT	N TMR	E 4	X SET	T MLR	ENT	

Goto Subroutine (GTS) (SBR)

DS	Used
HPP	Used

The Goto Subroutine instruction allows a section of ladder logic to be placed outside the main body of the program, to execute only when needed. There can be a maximum of 256 GTS instructions and 256 SBR instructions used in a program. The GTS instructions can be nested up to 8 levels. An error E412 will occur if the maximum limits are exceeded. Typically this will be used in an application where a block of program logic may be slow to execute and is not required to execute every scan. The subroutine label and all associated logic is placed after the End statement in the program. When the subroutine is called from the main program, the CPU will execute the subroutine (SBR) with the same constant number (K) as the GTS instruction which called the subroutine.





By placing code in a subroutine it is only scanned and executed when needed, since it resides after the End instruction. Code which is not scanned does not impact the overall scan time of the program.

Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

Subroutine Return (RT)

DS	Used
HPP	Used

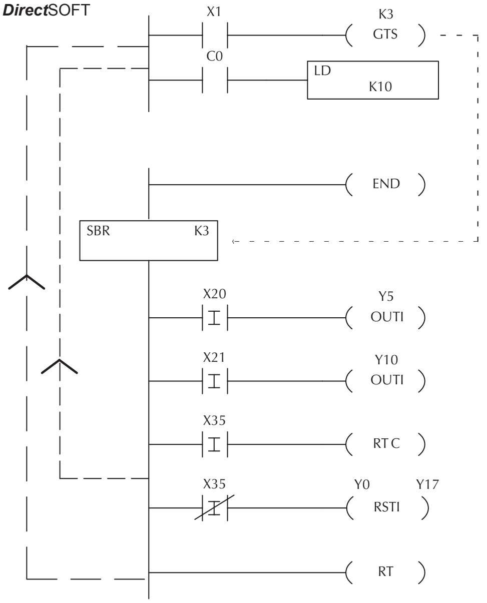
When a Subroutine Return is executed in the subroutine the CPU will return to the point in the main body of the program from which it was called. The Subroutine Return is used as termination of the subroutine. It must be the last instruction in the subroutine and is a stand alone instruction (no input contact on the rung).

Subroutine Return Conditional (RTC)

DS	Used
HPP	Used

The Subroutine Return Conditional instruction is an optional instruction used with an input contact to implement a conditional return from the subroutine. The Subroutine Return (RT) is still required for termination of the Subroutine.

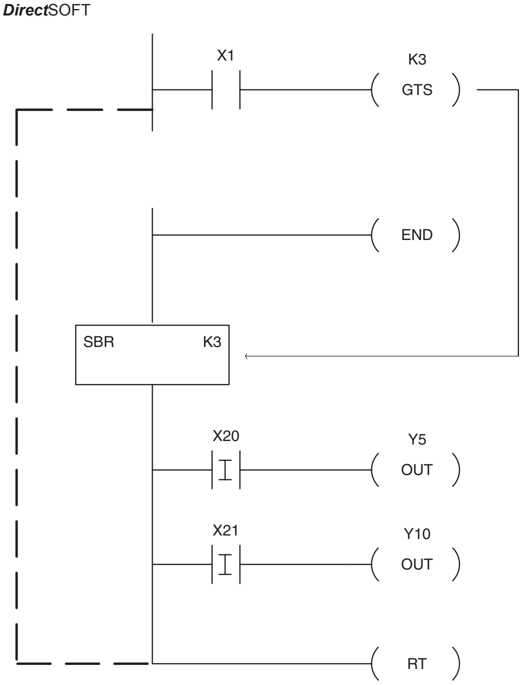
In the following example, when X1 is on, Subroutine K3 will be called. The CPU will jump to the Subroutine Label K3 and the ladder logic in the subroutine will be executed. If X35 is on, the CPU will return to the main program at the RTC instruction. If X35 is not on, Y0–Y17 will be reset to off and the CPU will return to the main body of the program.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	G 6	T MLR	S RST	→	D 3	ENT		
SHFT	E 4	N TMR	D 3	ENT				
SHFT	S RST	SHFT	B 1	R ORN	→	D 3	ENT	
$ STR	SHFT	I 8	→	C 2	A 0	ENT		
GX OUT	SHFT	I 8	→	F 5	ENT			
$ STR	SHFT	I 8	→	C 2	B 1	ENT		
GX OUT	SHFT	I 8	→	B 1	A 0	ENT		
$ STR	SHFT	I 8	→	D 3	F 5	ENT		
SHFT	R ORN	T MLR	C 2	ENT				
SP STRN	SHFT	I 8	→	D 3	F 5	ENT		
S RST	SHFT	I 8	→	A 0	→	B 1	H 7	ENT
SHFT	R ORN	T MLR	ENT					

In the following example, when X1 is on, Subroutine K3 will be called. The CPU will jump to the Subroutine Label K3 and the ladder logic in the subroutine will be executed. The CPU will return to the main body of the program after the RT instruction is executed.



Handheld Programmer Keystrokes

$ STR | → | B 1 | ENT

SHFT | G 6 | T MLR | S RST | → | D 3 | ENT

SHFT | E 4 | N TMR | D 3 | ENT

SHFT | S RST | SHFT | B 1 | R ORN | → | D 3 | ENT

$ STR | SHFT | I 8 | → | C 2 | A 0 | ENT

GX OUT | → | F 5 | ENT

$ STR | SHFT | I 8 | → | C 2 | B 1 | ENT

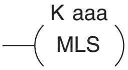
GX OUT | → | B 1 | A 0 | ENT

SHFT | R ORN | T MLR | ENT

Master Line Set (MLS)

DS	Used
HPP	Used

The Master Line Set instruction allows the program to control sections of ladder logic by forming a new power rail controlled by the main left power rail. The main left rail is always master line 0. When an MLS K1 instruction is used, a new power rail is created at level 1. Master Line Sets and Master Line Resets can be used to nest power rails up to seven levels deep.

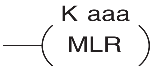


Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

Master Line Reset (MLR)

DS	Used
HPP	Used

The Master Line Reset instruction marks the end of control for the corresponding MLS instruction. The MLR reference is one less than the corresponding MLS.

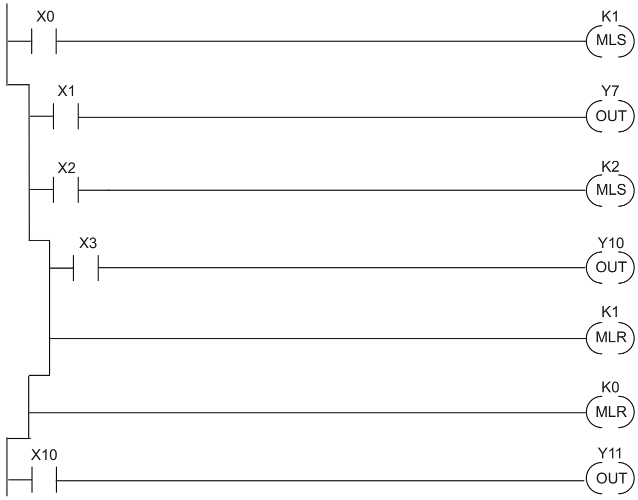


Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

Understanding Master Control Relays

The Master Line Set (MLS) and Master Line Reset (MLR) instructions allow you to quickly enable (or disable) sections of the RLL program. This provides program control flexibility. The following example shows how the MLS and MLR instructions operate by creating a sub power rail for control logic.

*Direct*SOFT



When contact X0 is ON, logic under the first MLS will be executed.

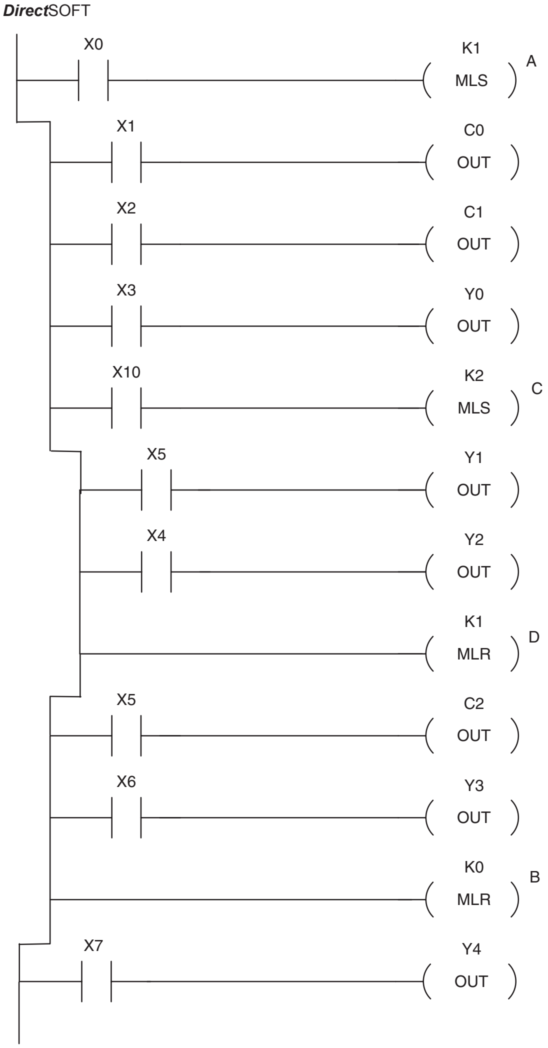
When contact X0 and X2 are ON, logic under the second MLS will be executed.

The MLR instructions note the end of the Master Control area.

MLS/MLR Example

In the following MLS/MLR example logic between the first MLS K1 (A) and MLR K0 (B) will function only if input X0 is on. The logic between the MLS K2 (C) and MLR K1 (D) will function only if input X10 and X0 is on. The last rung is not controlled by either of the MLS coils.

*Direct*SOFT



Handheld Programmer Keystrokes

$ STR	→	A 0	ENT		
Y MLS	→	B 1	ENT		
$ STR	→	B 1	ENT		
GX OUT	→	SHFT	C 2	A 0	ENT
$ STR	→	C 2	ENT		
GX OUT	→	SHFT	C 2	B 1	ENT
$ STR	→	D 3	ENT		
GX OUT	→	A 0	ENT		
$ STR	→	B 1	A 0	ENT	
Y MLS	→	C 2	ENT		
$ STR	→	F 5	ENT		
GX OUT	→	B 1	ENT		
$ STR	→	E 4	ENT		
GX OUT	→	C 2	ENT		
T MLR	→	B 1	ENT		
$ STR	→	F 5	ENT		
GX OUT	→	SHFT	C 2	C 2	ENT
$ STR	→	G 6	ENT		
GX OUT	→	D 3	ENT		
T MLR	→	A 0	ENT		
$ STR	→	H 7	ENT		
GX OUT	→	E 4	C 2	ENT	

Interrupt Instructions

Interrupt (INT)

DS	Used
HPP	Used

INT O aaa

The Interrupt instruction allows a section of ladder logic to be placed below the main body of the program and executed only when needed. High-Speed I/O Modes 10, 20, and 40 can generate an interrupt. With Mode 40, you may select an external interrupt (input X0), or a time-based interrupt (3–999 ms).

Typically, interrupts are used in an application when a fast response to an input is needed or a program section must execute faster than the normal CPU scan. The interrupt label and all associated logic must be placed after the End statement in the program. When an interrupt occurs, the CPU will complete execution of the current instruction it is processing in ladder logic, then execute the interrupt routine. After interrupt routine execution, the ladder program resumes from the point at which it was interrupted.

See Chapter 3, the section on Mode 40 (Interrupt) Operation for more details on interrupt configuration. In the DL06, only one software interrupt is available. The software interrupt uses interrupt #00 (INT 0), which means the hardware interrupt #0 and the software interrupt cannot be used together. Hardware interrupts are labeled in octal to correspond with the hardware input signal (e.g. X1 will initiate INT 1).

Operand Data Type		DL06 Range
		aaa
Constant	0	1-FFFF

Interrupt Return (IRT)

DS	Used
HPP	Used

An Interrupt Return is normally executed as the last instruction in the interrupt routine. It returns the CPU to the point in the main program from which it was called. The Interrupt Return is a stand-alone instruction (no input contact on the rung).

—(IRT)

Interrupt Return Conditional (IRTC)

DS	Used
HPP	Used

The Interrupt Return Conditional instruction is a optional instruction used with an input contact to implement a conditional return from the interrupt routine. The Interrupt Return is required to terminate the interrupt routine.

—(IRTC)

Enable Interrupts (ENI)

DS	Used
HPP	Used

The Enable Interrupt instruction is placed in the main ladder program (before the End instruction), enabling the interrupt. The interrupt remains enabled until the program executes a Disable Interrupt instruction.

—(ENI)

Disable Interrupts (DISI)

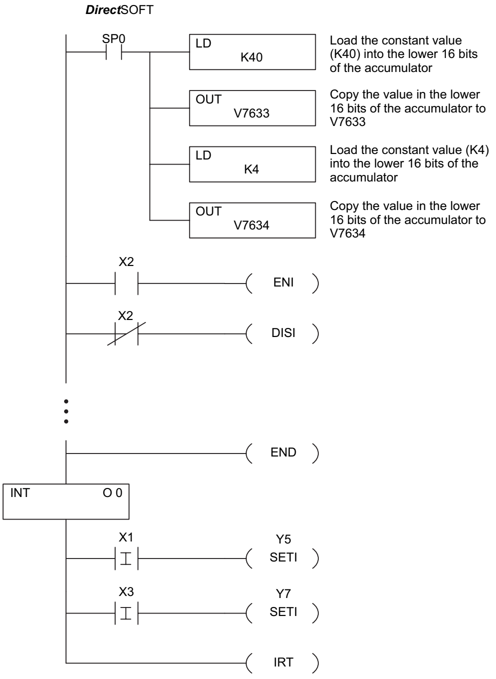
DS	Used
HPP	Used

A Disable Interrupt instruction in the main body of the application program (before the End instruction) will disable the interrupt (either external or timed). The interrupt remains disabled until the program executes an Enable Interrupt instruction.

External Interrupt Program Example

In the following example, we do some initialization on the first scan, using the first-scan contact SP0. The interrupt feature is the HSIO Mode 40. Then, we configure X0 as the external interrupt by writing to its configuration register, V7634. See Appendix E, Mode 40 Operation for more details.

During program execution, when X2 is on, the interrupt is enabled. When X2 is off, the interrupt will be disabled. When an interrupt signal (X0) occurs, the CPU will jump to the interrupt label INT O 0. The application ladder logic in the interrupt routine will be performed. The CPU will return to the main body of the program after the IRT instruction is executed.


*Direct*SOFT
SP0
LD
K40
Load the constant value (K40) into the lower 16 bits of the accumulator
OUT
V7633
Copy the value in the lower 16 bits of the accumulator to V7633
LD
K4
Load the constant value (K4) into the lower 16 bits of the accumulator
OUT
V7634
Copy the value in the lower 16 bits of the accumulator to V7634
X2
ENI
X2
DISI
END
INT
O 0
X1
Y5
SETI
X3
Y7
SETI
IRT

Handheld Programmer Keystrokes

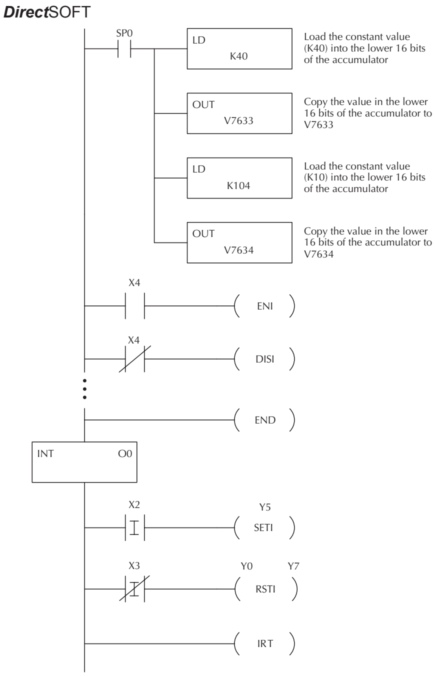
$ STR	→	SHFT	SP STRN	A 0	ENT			
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	A 0	ENT
GX OUT	→	SHFT	V AND	H 7	G 6	D 3	D 3	ENT
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	ENT	
GX OUT	→	SHFT	V AND	H 7	G 6	D 3	E 4	ENT
$ STR	→	C 2	ENT					
SHFT	E 4	N TMR	I 8	ENT				
SP STRN	→	C 2	ENT					
SHFT	D 3	I 8	S RST	I 8	ENT			

SHFT	E 4	N TMR	D 3	ENT		
SHFT	I 8	N TMR	T MLR	→	A 0	ENT
$ STR	SHFT	I 8	→	B 1	ENT	
X SET	SHFT	I 8	→	F 5	ENT	
$ STR	SHFT	I 8	→	D 3	ENT	
X SET	SHFT	I 8	→	H 7	ENT	
SHFT	I 8	R ORN	T MLR	ENT		

Timed Interrupt Program Example

In the following example, we do some initialization on the first scan, using the first-scan contact SP0. The interrupt feature is the HSIO Mode 40. Then we configure the HSIO timer as a 10 mS interrupt by writing K104 to the configuration register for X0 (V7634). See Appendix E, Mode 40 Operation for more details.

When X4 turns on, the interrupt will be enabled. When X4 turns off, the interrupt will be disabled. Every 10 mS the CPU will jump to the interrupt label INT O 0. The application ladder logic in the interrupt routine will be performed. If X3 is not on, Y0–Y7 will be reset to off and then the CPU will return to the main body of the program.



Load the constant value (K40) into the lower 16 bits of the accumulator

Copy the value in the lower 16 bits of the accumulator to V7633

Load the constant value (K10) into the lower 16 bits of the accumulator

Copy the value in the lower 16 bits of the accumulator to V7634

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	A 0	ENT	
GX OUT	→	SHFT	V AND	H 7	G 6	D 3	D 3	ENT	
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 1	A 0	E 4	ENT
GX OUT	→	SHFT	V AND	H 7	G 6	D 3	E 4	ENT	
$ STR	→	E 4	ENT						
SHFT	E 4	N TMR	I 8	ENT					
SP STRN	→	E 4	ENT						
SHFT	D 3	I 8	S RST	I 8	ENT				

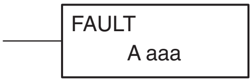
SHFT	E 4	N TMR	D 3	ENT			
SHFT	I 8	N TMR	T MLR	→	A 0	ENT	
$ STR	SHFT	I 8	→	C 2	ENT		
X SET	SHFT	I 8	→	F 5	ENT		
SP STRN	SHFT	I 8	→	D 3	ENT		
X SET	SHFT	I 8	→	A 0	→	H 7	ENT
SHFT	I 8	R ORN	T MLR	ENT			

Message Instructions

Fault (FAULT)

DS	Used
HPP	Used

The Fault instruction is used to display a message on the handheld programmer, the optional LCD display or in the *Direct*SOFT status bar. The message has a maximum of 23 characters and can be either V-memory data, numerical constant data or ASCII text.



To display the value in a V-memory location, specify the V-memory location in the instruction. To display the data in ACON (ASCII constant) or NCON (Numerical constant) instructions, specify the constant (K) value for the corresponding data label area.

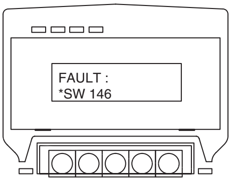
See Appendix G for the ASCII conversion table.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map
Constant	K	1-FFFF

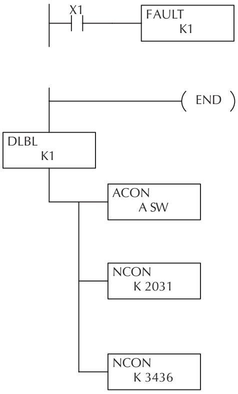
Discrete Bit Flags	Description
SP50	On when the FAULT instruction is executed

Fault Example

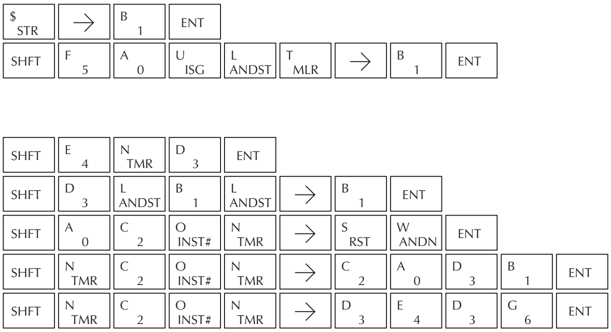
In the following example when X1 is on, the message SW 146 will display on the handheld programmer. The NCONs use the HEX ASCII equivalent of the text to be displayed. (The HEX ASCII for a blank is 20, a 1 is 31, 4 is 34 ...)



*Direct*SOFT



Handheld Programmer Keystrokes


$ STR | → | B 1 | ENT
SHFT | F 5 | A 0 | U ISG | L ANDST | T MLR | → | B 1 | ENT
SHFT | E 4 | N TMR | D 3 | ENT
SHFT | D 3 | L ANDST | B 1 | L ANDST | → | B 1 | ENT
SHFT | A 0 | C 2 | O INST# | N TMR | → | S RST | W ANDN | ENT
SHFT | N TMR | C 2 | O INST# | N TMR | → | C 2 | A 0 | D 3 | B 1 | ENT
SHFT | N TMR | C 2 | O INST# | N TMR | → | D 3 | E 4 | D 3 | G 6 | ENT

Data Label (DLBL)

DS	Used
HPP	Used

The Data Label instruction marks the beginning of an ASCII/ numeric data area. DLBLs are programmed after the End statement. A maximum of 64 DLBL instructions can be used in a program. Multiple NCONs and ACONs can be used in a DLBL area.

```
DLBL
      K aaa
```

Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

ASCII Constant (ACON)

DS	Used
HPP	Used

The ASCII Constant instruction is used with the DLBL instruction to store ASCII text for use with other instructions. Two ASCII characters can be stored in an ACON instruction. If only one character is stored in a ACON a leading space will be inserted.

```
ACON
      A aaa
```

Operand Data Type		DL06 Range
		aaa
ASCII	A	0-9 A-Z

Numerical Constant (NCON)

DS	Used
HPP	Used

The Numerical Constant instruction is used with the DLBL instruction to store the HEX ASCII equivalent of numerical data for use with other instructions. Two digits can be stored in an NCON instruction.

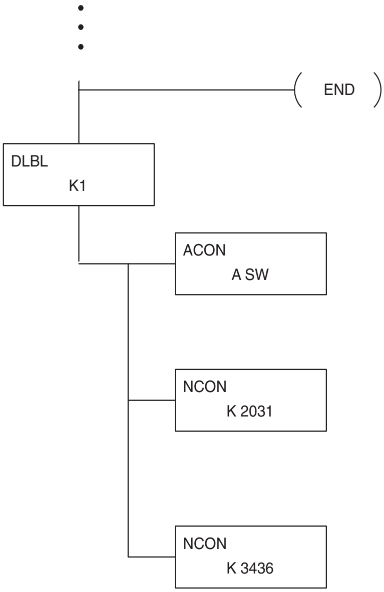
```
NCON
      K aaa
```

Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

Data Label Example

In the following example, an ACON and two NCON instructions are used within a DLBL instruction to build a text message. See the FAULT instruction for information on displaying messages. The DV-1000 Manual also has information on displaying messages.

*Direct*SOFT



Handheld Programmer Keystrokes

SHFT | E 4 | N TMR | D 3 | ENT

SHFT | D 3 | L ANDST | B 1 | L ANDST | → | B 1 | ENT

SHFT | A 0 | C 2 | O INST# | N TMR | → | S RST | W ANDN | ENT

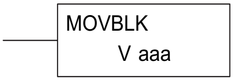
SHFT | N TMR | C 2 | O INST# | N TMR | → | C 2 | A 0 | D 3 | B 1 | ENT

SHFT | N TMR | C 2 | O INST# | N TMR | → | D 3 | E 4 | D 3 | G 6 | ENT

Move Block Instruction (MOVBLK)

DS	Used
HPP	Used

The Move Block instruction copies a specified number of words from a Data Label Area of program memory (ACON, NCON) to the specified V-memory location.

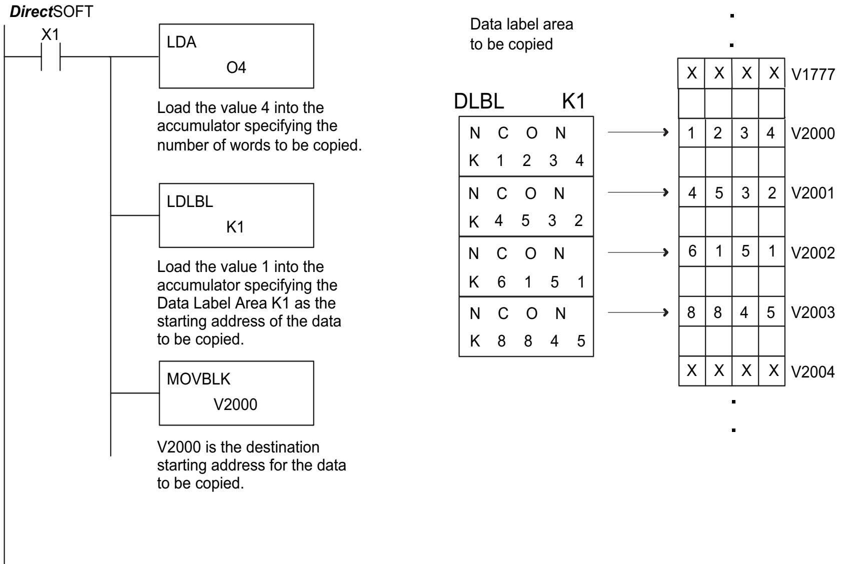


Below are the steps for using the Move Block function:

- Step 1: Load the number of words (octal) to be copied into the 1st level of the accumulator stack.
- Step 2: Load the source data label (LDLBL Kaaa) into the accumulator. This is where the data will be copied from.
- Step 3: Insert the MOVBLK instruction that specifies the V-memory destination. This is where the data will be copied to.

Copy Data From a Data Label Area to V-memory

In the example below, data is copied from a Data Label Area to V-memory. When X1 is on, the octal value (O4) is copied to the first level of the accumulator stack using the Load Address (LDA) instruction. This value specifies the number of words to be copied. Load Label (LDLBL) instruction will load the source data address (K1) into the accumulator. This is where the data will be copied from. The MOVBLK instruction specifies the destination starting location and executes the copying of data from the Data Label Area to V-memory.



Load the value 4 into the accumulator specifying the number of words to be copied.

Load the value 1 into the accumulator specifying the Data Label Area K1 as the starting address of the data to be copied.

V2000 is the destination starting address for the data to be copied.

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT									
SHFT	L ANDST	D 3	A 0	→	E 4	ENT						
SHFT	L ANDST	D 3	L ANDST	B 1	L ANDST	→	B 1	ENT				
SHFT	M ORST	O INST#	V AND	B 1	L ANDST	K JMP	→	C 2	A 0	A 0	A 0	ENT

Print Message (PRINT)

DS	Used
HPP	N/A

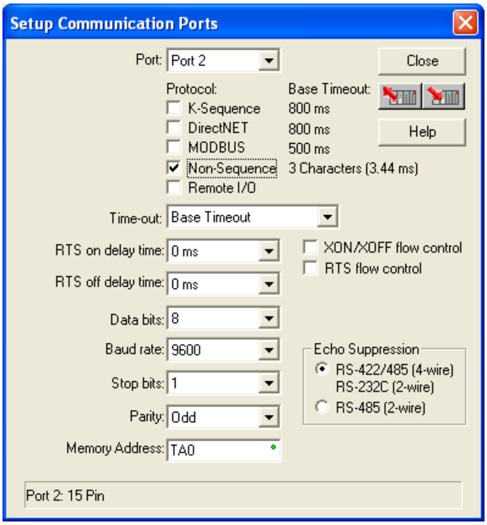
The Print Message instruction prints the embedded text or text/data variable message (maximum 128 characters) to the specified communications port (Port 2 on the DL06 CPU), which must have the communications port configured.

```
PRINT      A aaa
"Hello, this is a PLC message"
```

Operand Data Type		DL06 Range
		aaa
Constant	K	2

You may recall, from the CPU specifications in Chapter 3, that the DL06's ports are capable of several protocols. Port 1 cannot be configured for the non-sequence protocol. To configure port 2 using the Handheld Programmer, use AUX 56 and follow the prompts, making the same choices as indicated below on this page. To configure a port in ***Direct***SOFT, choose the PLC menu, then Setup, then Setup Secondary Comm Port.

- **Port**: From the port number list box at the top, choose **Port 2**.
- **Protocol**: Click the check box to the left of **Non-sequence**, and then you'll see the dialog box shown below.



Setup Communication Ports

Port: Port 2

Protocol:

	Base Timeout:
□ K-Sequence	800 ms
□ DirectNET	800 ms
□ MODBUS	500 ms
☑ Non-Sequence	3 Characters (3.44 ms)
□ Remote I/O	

Time-out: Base Timeout

RTS on delay time: 0 ms

RTS off delay time: 0 ms

□ XON/XOFF flow control

□ RTS flow control

Data bits: 8

Baud rate: 9600

Stop bits: 1

Parity: Odd

Echo Suppression

● RS-422/485 (4-wire) RS-232C (2-wire)

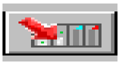
○ RS-485 (2-wire)

Memory Address: TA0

Port 2: 15 Pin

- **Baud Rate**: Choose the baud rate that matches your printer.
- **Stop Bits, Parity**: Choose number of stop bits and parity setting to match your printer.
- **Memory Address**: Choose a V-memory address for ***Direct***SOFT to use to store the port setup information. You will need to reserve 66 contiguous words in V-memory for this purpose.

Before ending the setup, click the button indicated to send Port 2 configuration to the CPU, and click **Close**. See Chapter 3 for port wiring information, in order to connect your printer to the DL06.



Port 2 on the DL06 has standard RS232 levels, and should work with most printer serial input connections.

Text element – this is used for printing character strings. The character strings are defined as the character (more than 0) ranged by the double quotation marks. Two hex numbers preceded by the dollar sign means an 8-bit ASCII character code. Also, two characters preceded by the dollar sign is interpreted according to the following table:

#	Character code	Description
1	$$	Dollar sign ($)
2	$”	Double quotation (”)
3	$L or $l	Line feed (LF)
4	$N or $n	Carriage return line feed (CRLF)
5	$P or $p	Form feed
6	$R or $r	Carriage return (CR)
7	$T or $t	Tab

The following examples show various syntax conventions and the length of the output to the printer.

Example:

” ” Length 0 without character

”A” Length 1 with character A

” ” Length 1 with blank

” $” ” Length 1 with double quotation mark

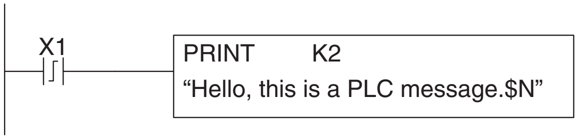
” $ R $ L ” Length 2 with one CR and one LF

” $ 0 D $ 0 A ” Length 2 with one CR and one LF

” $ $ ” Length 1 with one $ mark

In printing an ordinary line of text, you will need to include **double quotation** marks before and after the text string. Error code 499 will occur in the CPU when the print instruction contains invalid text or no quotations. It is important to test your PRINT instruction data during the application development.

The following example prints the message to port 2. We use a PD contact, which causes the message instruction to be active for just one scan. Note the $N at the end of the message, which produces a carriage return / line feed on the printer. This prepares the printer to print the next line, starting from the left margin.



Print the message to Port 2 when X1 makes an off-to-on transition.

V-memory element - this is used for printing V-memory contents in the integer format or real format. Use V-memory number or V-memory number with ":" and data type. The data types are shown in the table below. The Character code must be capital letters.

***NOTE**: There must be a space entered before and after the V-memory address to separate it from the text string. Failure to do this will result in an error code 499.*

#	Character code	Description
1	none	16-bit binary (decimal number)
2	: B	4 digit BCD
3	: D	32-bit binary (decimal number)
4	: D B	8 digit BCD
5	: R	Floating point number (real number)
6	: E	Floating point number (real number with exponent)

Example:

V2000 Print binary data in V2000 for decimal number

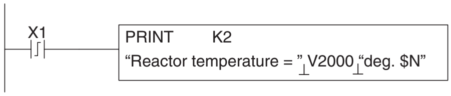
V2000 : B Print BCD data in V2000

V2000 : D Print binary number in V2000 and V2001 for decimal number

V2000 : D B Print BCD data in V2000 and V2001

V2000 : R Print floating point number in V2000/V2001 as real number

V2000 : E Print floating point number in V2000/V2001 as real number with exponent



```
PRINT      K2
"Reactor temperature = "⊥V2000⊥"deg. $N"
```

Print the message to Port 2 when X1 makes an off-to-on transition.

⊥ represents a space

Message will read:
Reactor temperature = 0156 deg.

Example: The following example prints a message containing text and a variable. The "reactor temperature" labels the data, which is at V2000. You can use the : B qualifier after the V2000 if the data is in BCD format, for example. The final string adds the units of degrees to the line of text, and the $N adds a carriage return / line feed.

V-memory text element - This is used for printing text stored in V-memory. Use the % followed by the number of characters after V-memory number for representing the text. If you assign "0" as the number of characters, the print function will read the character count from the first location. Then it will start at the next V-memory location and read that number of ASCII codes for the text from memory.

Example:

V2000 % 16	16 characters in V2000 to V2007 are printed.
V2000 % 0	The characters in V2001 to Vxxxx (determined by the number in V2000) will be printed.

Bit element

This is used for printing the state of the designated bit in V-memory or a relay bit. The bit element can be assigned by the designating point (.) and bit number preceded by the V-memory number or relay number. The output type is described as shown in the table below.

#	Data Format	Description
1	none	Print 1 for an ON state, and 0 for an OFF state
2	:BOOL	Print “TRUE” for an ON state, and “FALSE” for an OFF state
3	:ONOFF	Print “ON” for an ON state, and “OFF” for an OFF state

Example:

V2000 . 15 Prints the status of bit 15 in V2000, in 1/0 format

C100 Prints the status of C100 in 1/0 format

C100 : BOOL Prints the status of C100 in TRUE/FALSE format

C100 : ON/OFF Prints the status of C100 in ON/OFF format

V2000.15 : BOOL Prints the status of bit 15 in V2000 in TRUE/FALSE format

The maximum numbers of characters you can print is 128. The number of characters for each element is listed in the table below:

Element Type	Maximum Characters
Text, 1 character	1
16 bit binary	6
32 bit binary	11
4 digit BCD	4
8 digit BCD	8
Floating point (real number)	12
Floating point (real with exponent)	12
V-memory/text	2
Bit (1/0 format)	1
Bit (TRUE/FALSE format)	5
Bit (ON/OFF format)	3

The handheld programmer’s mnemonic is “PRINT” followed by the DEF field.

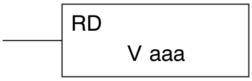
Special relay flags SP116 and SP117 indicate the status of the DL06 CPU ports (busy, or communications error). See the appendix on special relays for a description.

NOTE: *You must use the appropriate special relay in conjunction with the PRINT command to ensure the ladder program does not try to PRINT to a port that is still busy from a previous PRINT or WX or RX instruction.*

Intelligent I/O Instructions

Read from Intelligent Module (RD)

DS32	Used
HPP	Used



The Read from Intelligent Module instruction reads a block of data (1-128 bytes maximum) from an intelligent I/O module into the CPU's V-memory. It loads the function parameters into the first and second level of the accumulator stack and the accumulator by three additional instructions.

Listed below are the steps to program the Read from Intelligent module function.

Step 1: Load the base number (0-3) into the first byte and the slot number (0-7) into the second byte of the second level of the accumulator stack.

Step 2: Load the number of bytes to be transferred into the first level of the accumulator stack (maximum of 128 bytes).

Step 3: Load the address from which the data will be read into the accumulator. This parameter must be a HEX value.

Step 4: Insert the RD instruction which specifies the starting V-memory location (Vaaa) where the data will be read into.

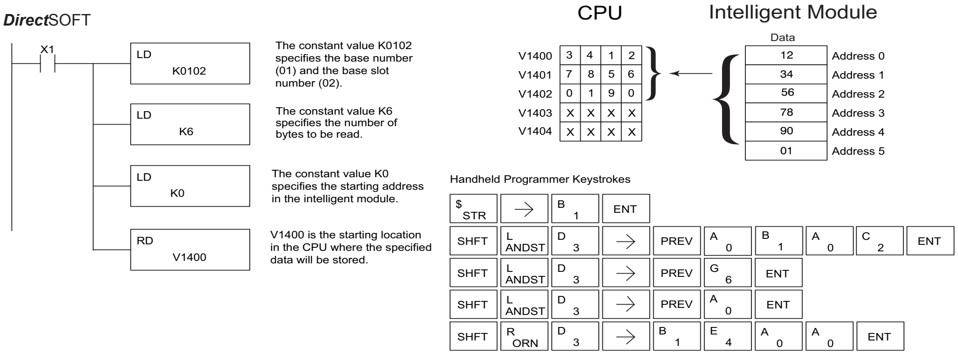
Helpful Hint: — Use the LDA instruction to convert an octal address to its HEX equivalent and load it into the accumulator when the HEX format is required.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP54	On when RX, WX RD, WT instructions are executed with the wrong parameters.

***NOTE:** Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is ON, the RD instruction will read six bytes of data from a intelligent module in base 1, slot 2, starting at address 0 in the intelligent module, and copy the information into V-memory loacations V1400-V1402.



The constant value K0102 specifies the base number (01) and the base slot number (02).

The constant value K6 specifies the number of bytes to be read.

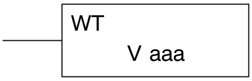
The constant value K0 specifies the starting address in the intelligent module.

V1400 is the starting location in the CPU where the specified data will be stored.

Write to Intelligent Module (WT)

DS32	Used
HPP	Used

The Write to Intelligent Module instruction writes a block of data (1-128 bytes maximum) to an intelligent I/O module from a block of V-memory in the CPU. The function parameters are loaded into the first and second level of the accumulator stack and the accumulator by three additional instructions.



Listed below are the steps to program the Read from Intelligent module function.

Step 1: Load the base number (0-3) into the first byte and the slot number (0-7) into the second byte of the second level of the accumulator stack.

Step 2: Load the number of bytes to be transferred into the first level of the accumulator stack (maximum of 128 bytes).

Step 3: Load the intelligent module address which will receive the data into the accumulator. This parameter must be a HEX value.

Step 4: Insert the WT instruction which specifies the starting V-memory location (Vaaa) where the data will be written from in the CPU.

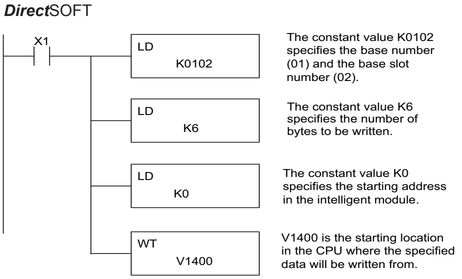
Helpful Hint: — Use the LDA instruction to convert an octal address to its HEX equivalent and load it into the accumulator when the HEX format is required.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP54	On when RX, WX RD, WT instructions are executed with the wrong parameters.

NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the WT instruction will write six bytes of data to an intelligent module in base 1, slot 2,starting at address 0 in the intelligent module, and copy the data from V-memory locations V1400-V1402.

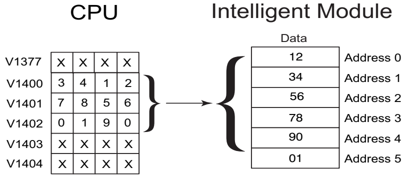


The constant value K0102 specifies the base number (01) and the base slot number (02).

The constant value K6 specifies the number of bytes to be written.

The constant value K0 specifies the starting address in the intelligent module.

V1400 is the starting location in the CPU where the specified data will be written from.



Handheld Programmer Keystrokes

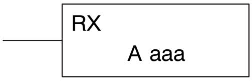
$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	PREV	A 0	B 1	A 0	C 2	ENT
SHFT	L ANDST	D 3	→	PREV	G 6	ENT			
SHFT	L ANDST	D 3	→	PREV	A 0	ENT			
SHFT	W ANDN	T MLR	→	B 1	E 4	A 0	A 0	ENT	

Network Instructions

Read from Network (RX)

DS32	Used
HPP	Used

The Read from Network instruction is used by the master device on a network to read a block of data from a slave device on the same network. The function parameters are loaded into the first and second level of the accumulator stack and the accumulator by three additional instructions.



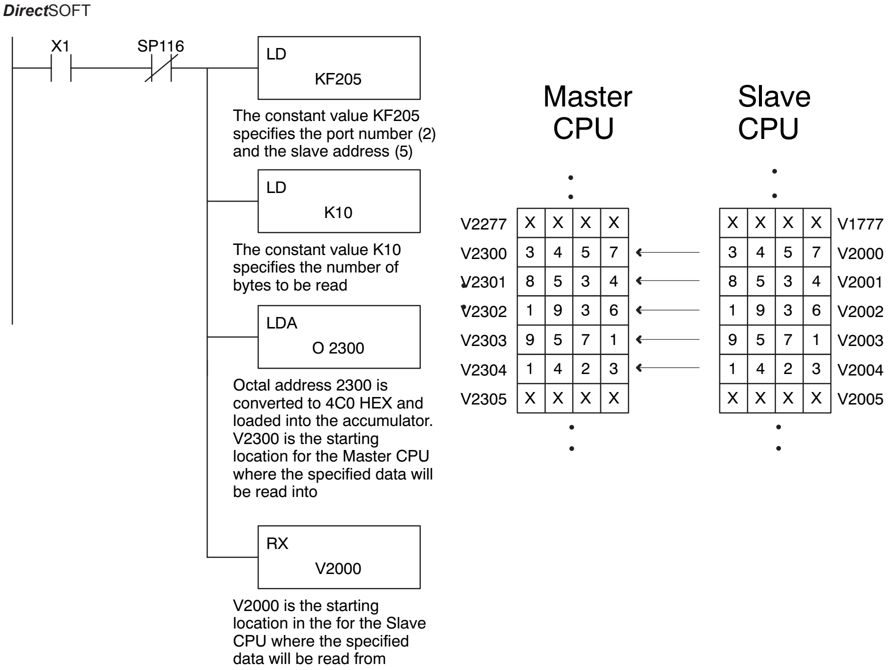
Listed below are the steps necessary to program the Read from Network function.

- Step 1: Load the slave address (0-- 90 BCD) into the first byte and the PLC internal port (KF2) or slot number of the master DCM or ECOM (0-- 7) into the second byte of the second level of the accumulator stack.
- Step 2: Load the number of bytes to be transferred into the first level of the accumulator stack.
- Step 3: Load the address of the data to be read into the accumulator. This parameter requires a HEX value.
- Step 4: Insert the RX instruction which specifies the starting Vmemory location (Aaaa) where the data will be read from in the slave.

Helpful Hint: — For parameters that require HEX values, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
	A	**aaa**
V-memory	V	See memory map
Pointer	P	See memory map
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0–777
Program Memory	$	0–7680 (2K program mem.)

In the following example, when X1 is on and the port busy relay SP116 (see special relays) is not on, the RX instruction will access port 2 operating as a master. Ten consecutive bytes of data (V2000 – V2004) will be read from a CPU at station address 5 and copied into V-memory locations V2300–V2304 in the CPU with the master port.



The constant value KF205 specifies the port number (2) and the slave address (5)

The constant value K10 specifies the number of bytes to be read

Octal address 2300 is converted to 4C0 HEX and loaded into the accumulator. V2300 is the starting location for the Master CPU where the specified data will be read into

V2000 is the starting location in the for the Slave CPU where the specified data will be read from

Master CPU										Slave CPU
V2277	X	X	X	X		X	X	X	X	V1777
V2300	3	4	5	7	←	3	4	5	7	V2000
V2301	8	5	3	4	←	8	5	3	4	V2001
V2302	1	9	3	6	←	1	9	3	6	V2002
V2303	9	5	7	1	←	9	5	7	1	V2003
V2304	1	4	2	3	←	1	4	2	3	V2004
V2305	X	X	X	X		X	X	X	X	V2005

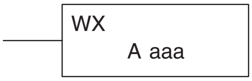
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT										
W ANDN	→	SHFT	SP STRN	B 1	B 1	G 6	ENT						
SHFT	L ANDST	D 3	→	SHFT	K JMP	SHFT	F 5	SHFT	C 2	A 0	F 5	ENT
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 1	A 0	ENT				
SHFT	L ANDST	D 3	A 0	→	C 2	D 3	A 0	A 0	ENT			
SHFT	R ORN	X SET	→	C 2	A 0	A 0	A 0	ENT				

Write to Network (WX)

DS	Used
HPP	Used

The Write to Network instruction is used to write a block of data from the master device to a slave device on the same network. The function parameters are loaded into the accumulator and the first and second levels of the stack.



Listed below are the program steps necessary to execute the Write to Network function.

Step 1: Load the slave address (0–90 BCD) into the low byte and "F2" into the high byte of the accumulator (the next two instructions push this word down to the second layer of the stack).

Step 2: Load the number of bytes to be transferred into the accumulator (the next instruction pushes this word onto the top of the stack).

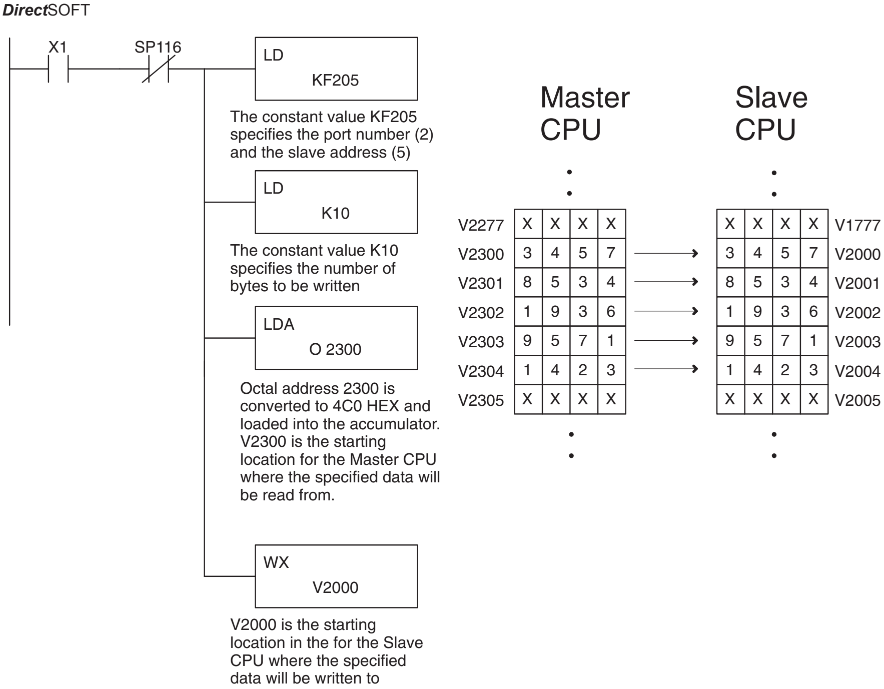
Step 3: Load the starting Master CPU address into the accumulator. This is the memory location where the data will be written from. This parameter requires a HEX value.

Step 4: Insert the WX instruction which specifies the starting V-memory location (Aaaa) where the data will be written to in the slave.

Helpful Hint: — For parameters that require HEX values, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0–777
Program Memory	$	0–7680 (2K program mem.)

In the following example, when X1 is on and the module busy relay SP116 (see special relays) is not on, the WX instruction will access port 2 operating as a master. Ten consecutive bytes of data are read from the Master CPU and copied to V-memory locations V2000–V2004 in the slave CPU at station address 5.



The constant value KF205 specifies the port number (2) and the slave address (5)

The constant value K10 specifies the number of bytes to be written

Octal address 2300 is converted to 4C0 HEX and loaded into the accumulator. V2300 is the starting location for the Master CPU where the specified data will be read from.

V2000 is the starting location in the for the Slave CPU where the specified data will be written to

Master CPU										Slave CPU
V2277	X	X	X	X		X	X	X	X	V1777
V2300	3	4	5	7	→	3	4	5	7	V2000
V2301	8	5	3	4	→	8	5	3	4	V2001
V2302	1	9	3	6	→	1	9	3	6	V2002
V2303	9	5	7	1	→	9	5	7	1	V2003
V2304	1	4	2	3	→	1	4	2	3	V2004
V2305	X	X	X	X		X	X	X	X	V2005

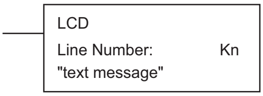
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT									
W ANDN	→	SHFT	SP STRN	B 1	C 1	E 6	ENT					
SHFT	L ANDST	D 3	→	SHFT	K JMP	SHFT	F 5	SHFT	C 2	A 0	F 5	ENT
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 1	A 0	ENT				
SHFT	L ANDST	D 3	A 0	→	C 2	D 3	A 0	A 0	ENT			
SHFT	W ANDN	X SET	→	C 2	A 0	A 0	A 0	ENT				

LCD

DS	Used
HPP	N/A

When enabled, the LCD instruction causes a user-defined text message to be displayed on the LCD Display Panel. The display is 16 characters wide by 2 rows high so a total of 32 characters can be displayed. Each row is addressed separately; the maximum number of characters the instruction will accept is 16.



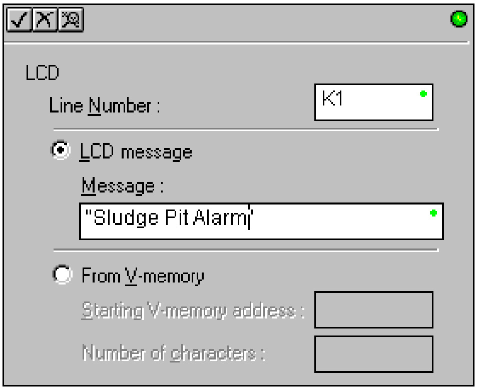
The text message can be entered directly into the message field of the instruction set-up dialog, or it can be located anywhere in user V-memory. If the text is located in V-memory, the LCD instruction is used to point to the memory location where the desired text originates. The length of the text string is also required.

From the *Direct*SOFT project folder, use the Instruction Browser to locate the LCD instruction. When you select the LCD instruction and click OK, the LCD dialog will appear, as shown in the examples. The LCD instruction is inserted into the ladder program via this set-up dialog box.

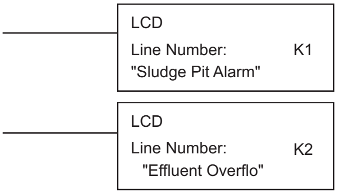
Display text strings can include embedded variables. Date and time settings and V-memory values can be embedded in the displayed text. Examples of each are shown.

Direct Text Entry

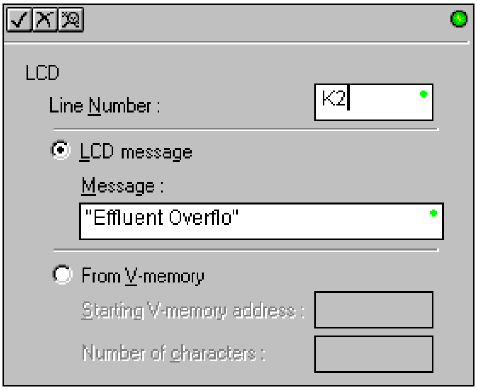
The two dialogs to the right show the selections necessary to create the two ladder instructions below. Double quotation marks are required to delineate the text string. In the first dialog, the text “Sludge Pit Alarm“ uses sixteen character spaces and will appear on line 1 when the instruction is enabled. Note, the line number is K1. Clicking the “check” button causes the instruction to be inserted into the ladder program.



LCD
Line Number : K1
● LCD message
Message :
"Sludge Pit Alarm"
○ From V-memory
Starting V-memory address :
Number of characters :



By identifying the second Line Number as K2, the text string “Effluent Overflow” will appear on the second line of the display when the second instruction is enabled.

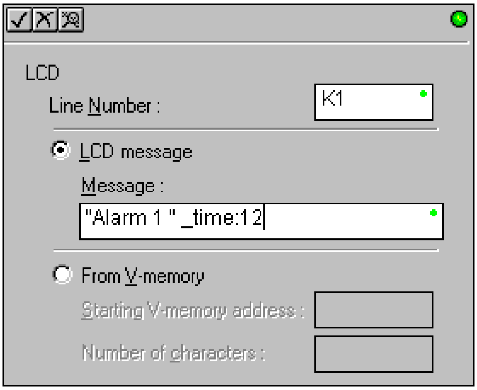


LCD
Line Number : K2
● LCD message
Message :
"Effluent Overflo"
○ From V-memory
Starting V-memory address :
Number of characters :

S	l	u	d	g	e		P	i	t		A	l	a	r	m
E	f	f	l	u	e	n	t		O	v	e	r	f	l	o

Embedding date and/or time variables

The date and/or time can be embedded in the displayed text by using the variables listed in the table below. These variables can be included in the **LCD message** field of the LCD dialog. In the example, the time variable (12 hour format) is embedded by adding _time:12. This time format uses a maximum of seven character spaces. The second dialog creates an instruction that prints the date on the second line of the display, when enabled.



LCD

Line Number : K1

● LCD message

Message :

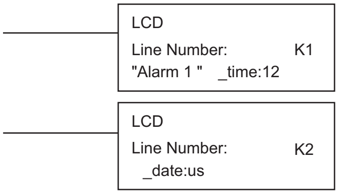
"Alarm 1 " _time:12

○ From V-memory

Starting V-memory address :

Number of characters :

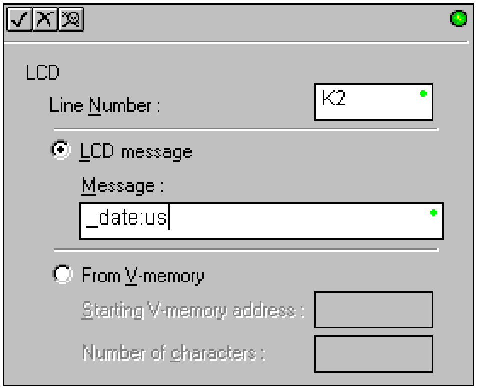
Date and Time Variables and Formats		
_date:us	US format	MM/DD/YY
_date:e	European format	DD/MM/YY
_date:a	Asian format	YY/MM/DD
_time:12	12 hour format	HH:MMAM/PM
_time:24	24 hour format	HH:MM:SS



LCD
Line Number: K1
"Alarm 1 " _time:12

LCD
Line Number: K2
_date:us

A	l	a	r	m		1			1	1	:	2	1	P	M
0	5	-	0	8	-	0	2								



LCD

Line Number : K2

● LCD message

Message :

_date:us

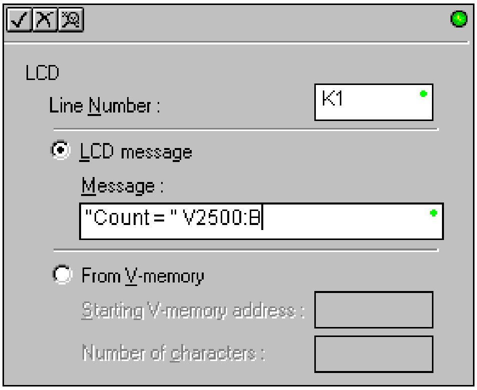
○ From V-memory

Starting V-memory address :

Number of characters :

Embedding V-memory data

Any V-memory data can be displayed in any one of six available data formats. An example appears to the right. A list of data formats and modifiers is on the next page. Note that different data formats require differing numbers of character positions on the display.



LCD

Line Number : K1

● LCD message

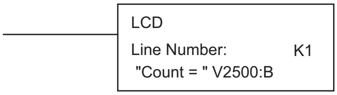
Message :

"Count = " V2500:B

○ From V-memory

Starting V-memory address :

Number of characters :



LCD
Line Number: K1
"Count = " V2500:B

C	o	u	n	t		=		0	4	1	2				

Data Format Suffixes for Embedded V-memory Data

Several data formats are available for displaying V-memory data on the LCD. The choices are shown in the table below. A colon is used to separate the embedded V-memory location from the data format suffix and modifier. An example appears on the previous page.

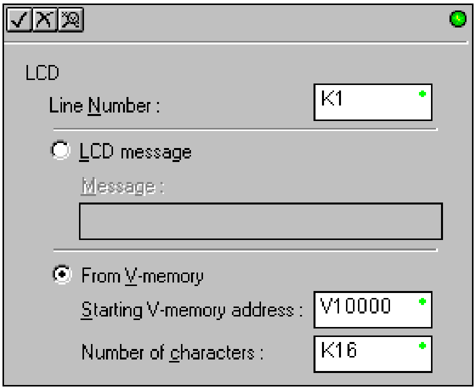
Data Format	Modifier	Example	Displayed Characters												
none (16-bit format)		V2000 = 0000 0000 0001 0010	**1**	**2**	**3**	**4**									
		V2000			1	8									
	[:S]	V2000:S	1	8											
	[:C0]	V2000:C0	0	0	1	8									
	[:0]	V2000:0			1	8									
:B (4 digit BCD)		V2000 = 0000 0000 0001 0010	**1**	**2**	**3**	**4**									
	[:B]	V2000:B	0	0	1	2									
	[:BS]	V2000:BS	1	2											
	[:BC0]	V2000:BC0	0	0	1	2									
	[:B0]	V2000:B0			1	2									
:D (32-bit decimal)		V2000 = 0000 0000 0000 0000	**Double Word**												
		V2001 = 0000 0000 0000 0001	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**		
	[:D]	V2000:D							6	5	5	3	6		
	[:DS]	V2000:DS	6	5	5	3	6								
	[:DC0]	V2000:DC0	0	0	0	0	0	0	6	5	5	3	6		
	[:D0]	V2000:D0							6	5	5	3	6		
:DB (8 digit BCD)		V2000 = 0000 0000 0000 0000	**Double Word**												
		V2001 = 0000 0000 0000 0011	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**					
	[:DB]	V2000:DB	0	0	0	3	0	0	0	0					
	[:DBS]	V2000:DBS	3	0	0	0	0								
	[:DBC0]	V2000:DBC0	0	0	0	3	0	0	0	0					
	[:DB0]	V2000:DB0				3	0	0	0	0					
:R (DWord floating point number)		V2001/V2000 = 222.11111 (real number)	**Double Word**												
			1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
	[:R]	V2000:R				f	2	2	2	.	1	1	1	1	1
	[:RS]	V2000:RS	f	2	2	2	.	1	1	1	1	1			
	[:RC0]	V2000:RC0	f	0	0	0	2	2	2	.	1	1	1	1	1
	[:R0]	V2000:R0				f	2	2	2	.	1	1	1	1	1
:E (DWord floating point number with exponent)		V2001/V2000 = 222.1 (real number)	**Double Word**												
			1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
	[:E]	V2000:E		f	2	.	2	2	1	0	0	E	+	0	2
	[:ES]	V2000:ES	f	2	.	2	2	1	0	0	E	+	0	2	
	[:EC0]	V2000:EC0	f	2	.	2	2	1	0	0	E	+	0	2	
	[:E0]	V2000:E0	f	2	.	2	2	1	0	0	E	+	0	2	
f = plus/minus flag (plus = no symbol, minus = -)															

The S, C0, and 0 modifiers alter the presentation of leading zeros and spaces. S removes leading spaces and left justifies the result. C0 replaces leading spaces with leading zeros. 0 is a modification of C0. 0 eliminates any leading zeros in the C0 format version and converts them to spaces.

Text Entry from V-memory

Alternatively, text that resides in V-memory can be displayed on the LCD following the example on this page. The LCD dialog is used twice, once for each line on the display. The dialog requires the address of the first character to be displayed and the number of characters to be displayed.

For example, the two dialogs shown on this page would create the two LCD instructions below. When enabled, these instructions would cause the ASCII characters in V10000 to V10017 to be displayed. The ASCII characters and their corresponding memory locations are shown in the table below.



LCD

Line Number : K1

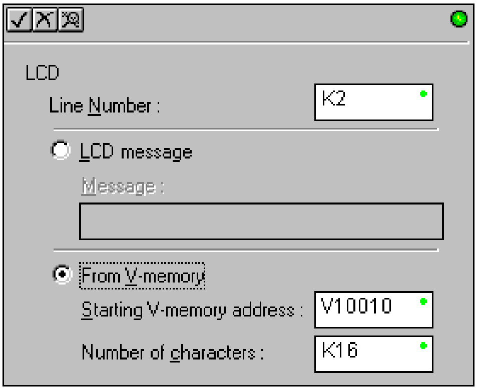
○ LCD message

Message :

● From V-memory

Starting V-memory address : V10000

Number of characters : K16



LCD

Line Number : K2

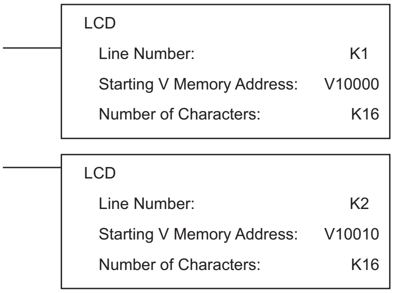
○ LCD message

Message :

● From V-memory

Starting V-memory address : V10010

Number of characters : K16



LCD
Line Number: K1
Starting V Memory Address: V10000
Number of Characters: K16

LCD
Line Number: K2
Starting V Memory Address: V10010
Number of Characters: K16

V10000	d	A
V10001	i	m
V10002		n
V10003	f	O
V10004	i	f
V10005	e	c
V10006		
V10007		
V10010	i	H
V10011	h	g
V10012	T	
V10013	m	e
V10014		p
V10015	l	A
V10016	r	a
V10017		m

A	d	m	i	n		O	f	f	i	c	e				
H	i	g	h		T	e	m	p		A	l	a	r	m	

MODBUS RTU Instructions

MODBUS Read from Network (MRX)

DS	Used
HPP	N/A

The MODBUS Read from Network (MRX) instruction is used by the DL06 network master to read a block of data from a connected slave device and to write the data into V–memory addresses within the master. The instruction allows the user to specify the MODBUS Function Code, slave station address, starting master and slave memory addresses, number of elements to transfer, MODBUS data format and the Exception Response Buffer.

- **CPU/DCM**: select either CPU or DCM module for communications
- **Slot Number**: select PLC option slot number if using a DCM module.
- **Port Number**: must be DL06 Port 2 (K2)
- **Slave Address**: specify a slave station address (0–247)
- **Function Code**: The following MODBUS function codes are supported by the MRX instruction:

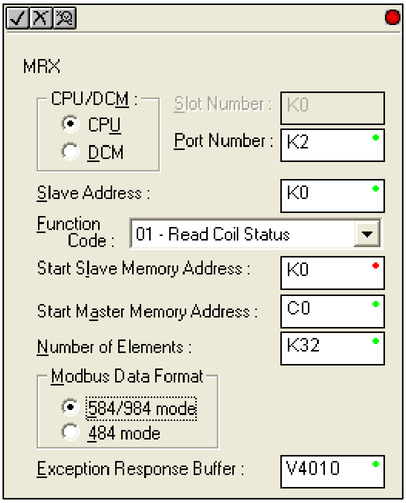
  01 – Read a group of coils

  02 – Read a group of inputs

  03 – Read holding registers

  04 – Read input registers

  07 – Read Exception status



MRX

CPU/DCM :
● CPU
○ DCM

Slot Number : K0

Port Number : K2

Slave Address : K0

Function Code : 01 - Read Coil Status

Start Slave Memory Address : K0

Start Master Memory Address : C0

Number of Elements : K32

Modbus Data Format
● 584/984 mode
○ 484 mode

Exception Response Buffer : V4010

- **Start Slave Memory Address**: specifies the starting slave memory address of the data to be read. See the table on the following page.
- **Start Master Memory Address**: specifies the starting memory address in the master where the data will be placed. See the table on the following page.
- **Number of Elements**: specifies how many coils, inputs, holding registers or input register will be read. See the table on the following page.
- **MODBUS Data Format**: specifies MODBUS 584/984 or 484 data format to be used
- **Exception Response Buffer:** specifies the master memory address where the Exception Response will be placed (6-bytes in length). See the table on the following page.The exception response buffer uses 3 words. These bytes are swapped in the MRX/MWX exception response buffer V-memory so:

  V-Memory 1 Hi Byte = Function Code Byte (Most Significant Bit Set)

  V-Memory 1 Lo Byte = Address Byte

  V-Memory 2 Hi Byte = One of the CRC Bytes

  V-Memory 2 Lo Byte = Exception Code

  V-Memory 3 Hi Byte = 0

  V-Memory 3 Lo Byte = Other CRC Byte

MRX Slave Address Ranges

Function Code	MODBUS Data Format	Slave Address Range(s)
01 – Read Coil	484 Mode	1–999
01 – Read Coil	584/984 Mode	1–65535
02 – Read Input Status	484 Mode	1001–1999
02 – Read Input Status	584/984 Mode	10001–19999 (5 digit) or 100001–165535 (6 digit)
03 – Read Holding Register	484 Mode	4001–4999
03 – Read Holding Register	584/984 Mode	40001–49999 (5 digit) or 4000001–465535 (6 digit)
04 – Read Input Register	484 Mode	3001–3999
04 – Read Input Register	584/984 Mode	30001–39999 (5 digit) or 3000001–365535 (6 digit)
07 – Read Exception Status	484 and 584/984 Mode	N/A

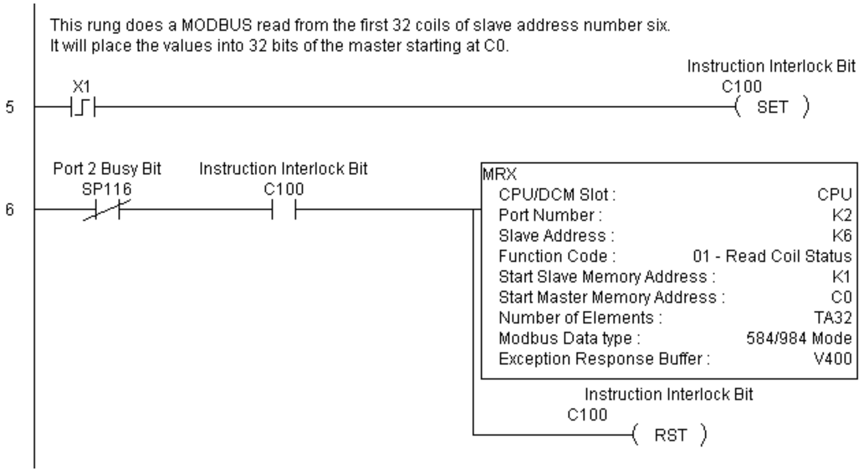
MRX Master Memory Address Ranges		
Operand Data Type		DL06 Range
Inputs	X	0–1777
Outputs	Y	0–1777
Control Relays	C	0–3777
Stage Bits	S	0–1777
Timer Bits	T	0–377
Counter Bits	CT	0–377
Special Relays	SP	0–777
V–memory	V	all
Global Inputs	GX	0–3777
Global Outputs	GY	0–3777

Number of Elements		
Operand Data Type		DL06 Range
V–memory	V	all
Constant	K	Bits: 1–2000 Registers: 1–125

Exception Response Buffer		
Operand Data Type		DL06 Range
V–memory	V	all

MRX Example

DL06 port 2 has two Special Relay contacts associated with it (see Appendix D for comm port special relays). One indicates "Port busy"(SP116), and the other indicates "Port Communication Error"(SP117). The "Port Busy" bit is on while the PLC communicates with the slave. When the bit is off, the program can initiate the next network request. The "Port Communication Error" bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes, since the error bit is reset when an MRX or MWX instruction is executed. Typically, network communications will last longer than 1 CPU scan. The program must wait for the communications to finish before starting the next transaction.



This rung does a MODBUS read from the first 32 coils of slave address number six.
It will place the values into 32 bits of the master starting at C0.

MRX
CPU/DCM Slot : CPU
Port Number : K2
Slave Address : K6
Function Code : 01 - Read Coil Status
Start Slave Memory Address : K1
Start Master Memory Address : C0
Number of Elements : TA32
Modbus Data type : 584/984 Mode
Exception Response Buffer : V400

5
X1
Instruction Interlock Bit
C100
SET
6
Port 2 Busy Bit
SP116
Instruction Interlock Bit
C100
Instruction Interlock Bit
C100
RST

***NOTE**: See Chapter 4, page 4-21, for an RLL example using multiple Read and Write interlocks with MRX/MWX instructions.*

MODBUS Write to Network (MWX)

DS	Used
HPP	N/A

The MODBUS Write to Network (MWX) instruction is used to write a block of data from the network masters's (DL06) memory to MODBUS memory addresses within a slave device on the network. The instruction allows the user to specify the MODBUS Function Code, slave station address, starting master and slave memory addresses, number of elements to transfer, MODBUS data format and the Exception Response Buffer.

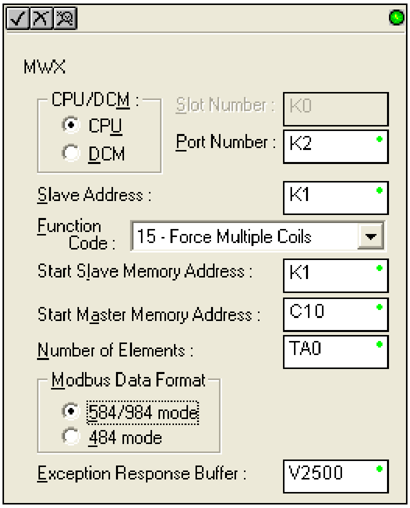
- **CPU/DCM**: select either CPU or DCM module for communications
- **Slot Number**: select PLC option slot number if using a DCM module
- **Port Number**: must be DL06 Port 2 (K2)
- **Slave Address**: specify a slave station address (0–247)
- **Function Code**: MODBUS function codes supported by the MWX instruction:

  05 – Force Single coil

  06 – Preset Single Register

  15 – Force Multiple Coils

  16 – Preset Multiple Registers



MWX

CPU/DCM : ● CPU ○ DCM

Slot Number : K0

Port Number : K2

Slave Address : K1

Function Code : 15 - Force Multiple Coils

Start Slave Memory Address : K1

Start Master Memory Address : C10

Number of Elements : TA0

Modbus Data Format: ● 584/984 mode ○ 484 mode

Exception Response Buffer : V2500

- **Start Slave Memory Address**: specifies the starting slave memory address where the data will be written
- **Start Master Memory Address**: specifies the starting address of the data in the master that is to be written to the slave
- **Number of Elements**: specifies how many consecutive coils or registers will be written to. This field is only active when either function code 15 or 16 is selected.
- **MODBUS Data Format**: specifies MODBUS 584/984 or 484 data format to be used
- **Exception Response Buffer:** specifies the master memory address where the Exception Response will be placed (6-bytes in length). See the table on the following page.The exception response buffer uses 3 words. These bytes are swapped in the MRX/MWX exception response buffer V-memory so:

  V-Memory 1 Hi Byte = Function Code Byte (Most Significant Bit Set)

  V-Memory 1 Lo Byte = Address Byte

  V-Memory 2 Hi Byte = One of the CRC Bytes

  V-Memory 2 Lo Byte = Exception Code

  V-Memory 3 Hi Byte = 0

  V-Memory 3 Lo Byte = Other CRC Byte

MWX Slave Address Ranges

MWX Slave Address Ranges		
Function Code	**MODBUS Data Format**	**Slave Address Range(s)**
05 – Force Single Coil	484 Mode	1–999
05 – Force Single Coil	584/984 Mode	1–65535
06 – Preset Single Register	484 Mode	4001–4999
06 – Preset Single Register	584/984 Mode	40001–49999 (5 digit) or 400001–465535 (6 digit)
15 – Force Multiple Coils	484 Mode	1–999
15 – Force Multiple Coils	585/984 Mode	1–65535
16 – Preset Multiple Registers	484 Mode	4001–4999
16 – Preset Multiple Registers	584/984 Mode	40001–49999 (5 digit) or 4000001–465535 (6 digit)

MWX Master Memory Address Ranges

MWX Master Memory Address Ranges		
Operand Data Type		**DL06 Range**
Inputs	X	0–1777
Outputs	Y	0–1777
Control Relays	C	0–3777
Stage Bits	S	0–1777
Timer Bits	T	0–377
Counter Bits	CT	0–377
Special Relays	SP	0–777
V–memory	V	all
Global Inputs	GX	0–3777
Global Outputs	GY	0–3777

MWX Number of Elements

Number of Elements		
Operand Data Type		**DL06 Range**
V–memory	V	all
Constant	K	Bits: 1–2000 Registers: 1–125

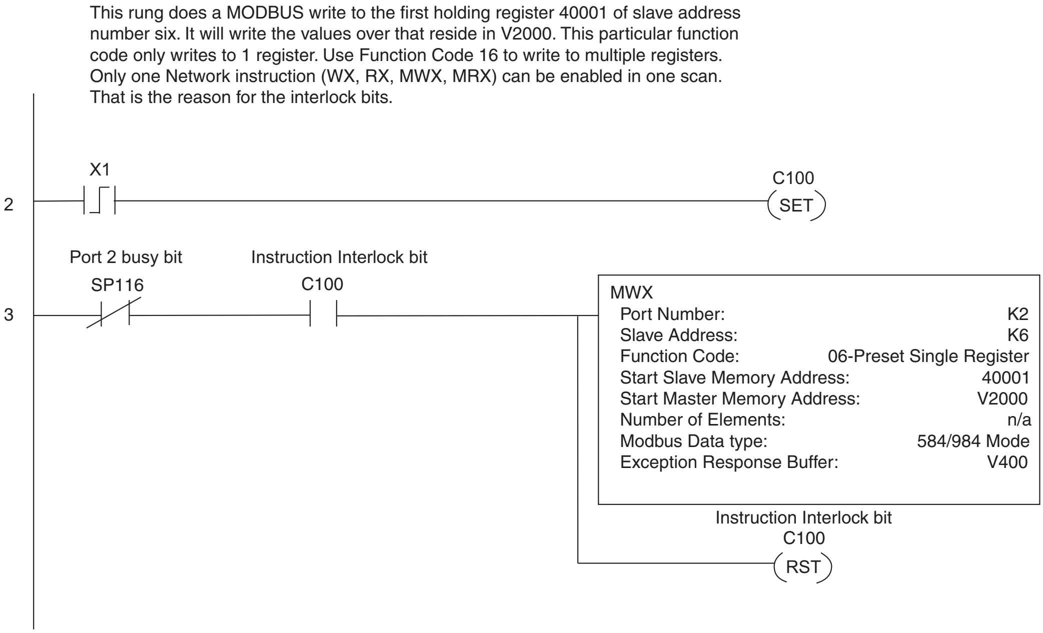
MWX Exception Response Buffer

Number of Elements		
Operand Data Type		**DL06 Range**
V–memory	V	all

MWX Example

DL06 port 2 has two Special Relay contacts associated with it (see Appendix D for comm port special relays). One indicates "Port busy"(SP116), and the other indicates "Port Communication Error"(SP117). The "Port Busy" bit is on while the PLC communicates with the slave. When the bit is off, the program can initiate the next network request. The "Port Communication Error" bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an MRX or MWX instruction is executed.

Typically, network communications will last longer than 1 CPU scan. The program must wait for the communications to finish before starting the next transaction.


This rung does a MODBUS write to the first holding register 40001 of slave address number six. It will write the values over that reside in V2000. This particular function code only writes to 1 register. Use Function Code 16 to write to multiple registers. Only one Network instruction (WX, RX, MWX, MRX) can be enabled in one scan. That is the reason for the interlock bits.
2
X1
C100
SET
3
Port 2 busy bit
SP116
Instruction Interlock bit
C100
MWX
Port Number: K2
Slave Address: K6
Function Code: 06-Preset Single Register
Start Slave Memory Address: 40001
Start Master Memory Address: V2000
Number of Elements: n/a
Modbus Data type: 584/984 Mode
Exception Response Buffer: V400
Instruction Interlock bit
C100
RST

NOTE: *See Chapter 4, page 4-21, for an RLL example using multiple Read and Write interlocks with MRX/MWX instructions.*

ASCII Instructions

The DL06 CPU supports several instructions and methods that allow ASCII strings to be read into and written from the PLC communications ports. Specifically, port 2 on the DL06 can be used for either reading or writing raw ASCII strings, but cannot be used for both at the same time. The DL06 can also decipher ASCII embedded within a supported protocol (K–Sequence, ***Direct***Net, Modbus) via the CPU port.

Reading ASCII Input Strings

There are several methods that the DL06 can use to read ASCII input strings.

1) ASCII IN (AIN) – This instruction configures port 2 for raw ASCII input strings with parameters such as fixed and variable length ASCII strings, termination characters, byte swapping options, and instruction control bits. Use barcode scanners, weight scales, etc. to write raw ASCII input strings into port 2 based on the (AIN) instruction's parameters.

2) Write embedded ASCII strings directly to V–memory from an external HMI or similar master device via a supported communications protocol using the CPU ports. The AIN instruction is not used in this case. 3) If a DL06 PLC is a master on a network, the Network Read instruction (RX) can be used to read embedded ASCII data from a slave device via a supported communications protocol using port 2. The RX instruction places the data directly into V–memory.

Writing ASCII Output Strings

The following instructions can be used to write ASCII output strings:

1) Print from V–memory (PRINTV) – Use this instruction to write raw ASCII strings out of port 2 to a display panel or a serial printer, etc. The instruction features the starting V–memory address, string length, byte swapping options, etc. When the instruction's permissive bit is enabled, the string is written to port 2.

2) Print to V–memory (VPRINT) – Use this instruction to create pre–coded ASCII strings in the PLC (i.e. alarm messages). When the instruction's permissive bit is enabled, the message is loaded into a pre–defined V–memory address location. Then the (PRINTV) instruction may be used to write the pre–coded ASCII string out of port 2. American, European and Asian Time/Date stamps are supported.

Additionally, if a DL06 PLC is a master on a network, the Network Write instruction (WX) can be used to write embedded ASCII data to an HMI or slave device directly from V–memory via a supported communications protocol using port 2.

Managing the ASCII Strings

The following instructions can be helpful in managing the ASCII strings within the CPUs V–memory:

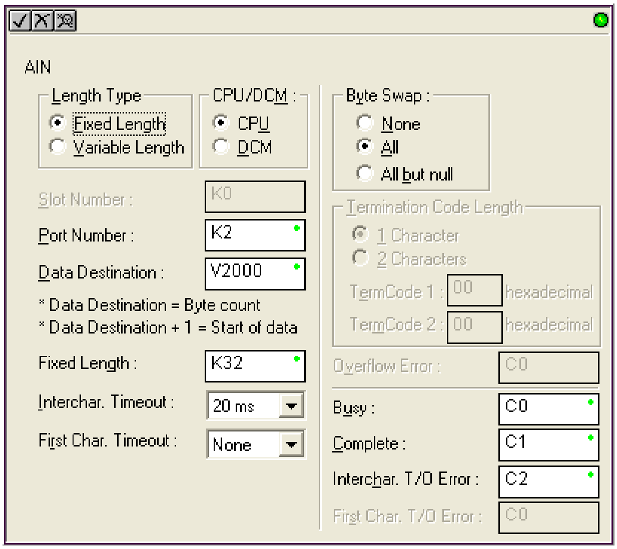
- ASCII Find (AFIND) – Finds where a specific portion of the ASCII string is located in continuous V–memory addresses. Forward and reverse searches are supported.
- ASCII Extract (AEX) – Extracts a specific portion (usually some data value) from the ASCII find location or other known ASCII data location.
- Compare V–memory (CMPV) – This instruction is used to compare two blocks of V–memory addresses and is usually used to detect a change in an ASCII string. Compared data types must be of the same format (i.e., BCD, ASCII, etc.).
- Swap Bytes (SWAPB) – usually used to swap V–memory bytes on ASCII data that was written directly to V–memory from an external HMI or similar master device via a communications protocol. The AIN and AEX instructions have a built–in byte swap feature.

ASCII Input (AIN)

DS	Used
HPP	N/A

The ASCII Input instruction allows the CPU to receive ASCII strings through the specified communications port and places the string into a series of specified V–memory registers. The ASCII data can be received as a fixed number of bytes or as a variable length string with specified termination character(s). Other features include, Byte Swap preferences, Character Timeout, and user defined flag bits for Busy, Complete and Timeout Error.

AIN Fixed Length Configuration



AIN

Length Type: ● Fixed Length ○ Variable Length

CPU/DCM: ● CPU ○ DCM

Byte Swap: ○ None ● All ○ All but null

Slot Number: K0

Port Number: K2

Data Destination: V2000

* Data Destination = Byte count
* Data Destination + 1 = Start of data

Fixed Length: K32

Interchar. Timeout: 20 ms

First Char. Timeout: None

Termination Code Length: ● 1 Character ○ 2 Characters

TermCode 1: 00 hexadecimal

TermCode 2: 00 hexadecimal

Overflow Error: C0

Busy: C0

Complete: C1

Interchar. T/O Error: C2

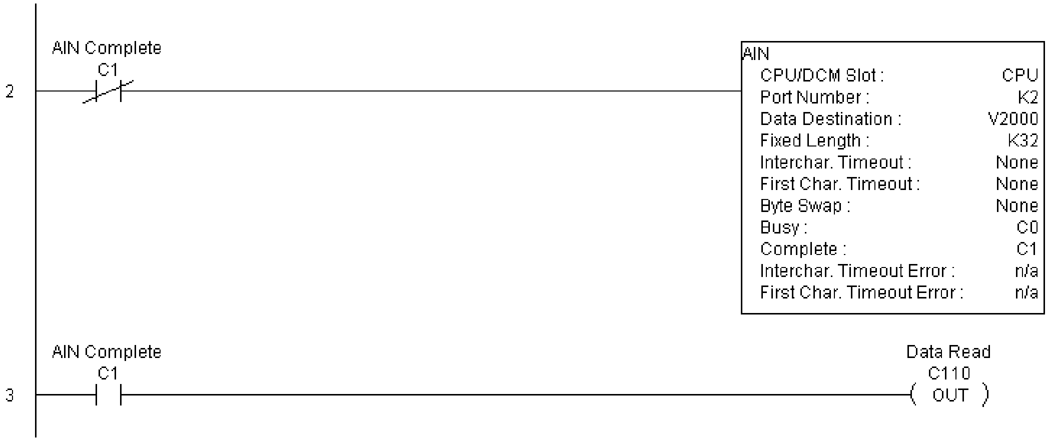
First Char. T/O Error: C0

- **Length Type:** select fixed length based on the length of the ASCII string that will be sent to the CPU port
- **Port Number**: must be DL06 port 2 (K2)
- **Data Destination**: specifies where the ASCII string will be placed in V–memory
- **Fixed Length**: specifies the length, in bytes, of the fixed length ASCII string the port will receive
- **Inter–character Timeout**: if the amount of time between incoming ASCII characters exceeds the set time, the specified Timeout Error bit will be set. No data will be stored at the Data Destination V–memory location. The bit will reset when the AIN instruction permissive bits are disabled. 0ms selection disables this feature.
- **First Character Timeout**: if the amount of time from when the AIN is enabled to the time the first character is received exceeds the set time, the specified First Character Timeout bit will be set. The bit will reset when the AIN instruction permissive bits are disabled. 0ms selection disables this feature.
- **Byte Swap**: swaps the high–byte and low–byte within each V–memory register of the Fixed Length ASCII string. See the SWAPB instruction for details.
- **Busy Bit**: is ON while the AIN instruction is receiving ASCII data
- **Complete Bit**: is set once the ASCII data has been received for the specified fixed length and reset when the AIN instruction permissive bits are disabled.
- **Inter–character Timeout Error Bit**: is set when the Character Timeout is exceeded. See Character Timeout explanation above.
- **First Character Timeout Error Bit**: is set when the First Character Timeout is exceeded. See First Character Timeout explanation above.

Parameter	
Data Destination	All V–memory
Fixed Length	K1–128
Bits: Busy, Complete, Timeout Error, Overflow	C0–3777

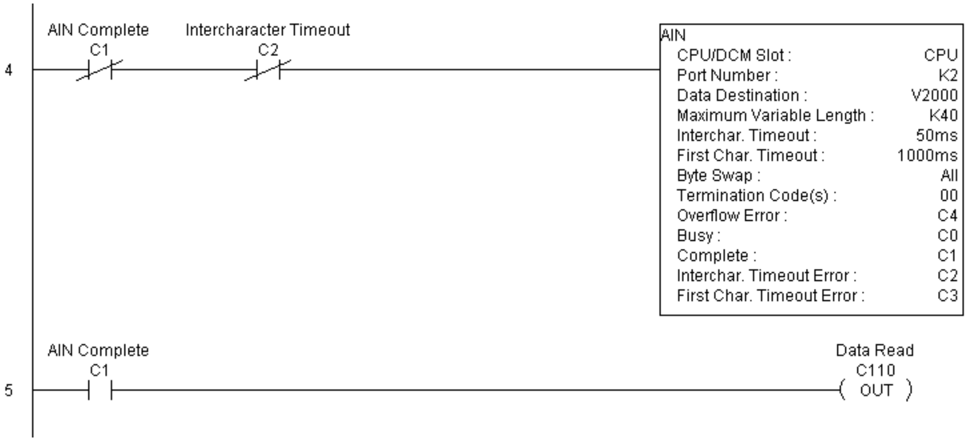
AIN Fixed Length Examples

Fixed Length example when the PLC is reading the port continuously and timing is not critical



AIN
CPU/DCM Slot : CPU
Port Number : K2
Data Destination : V2000
Fixed Length : K32
Interchar. Timeout : None
First Char. Timeout : None
Byte Swap : None
Busy : C0
Complete : C1
Interchar. Timeout Error : n/a
First Char. Timeout Error : n/a

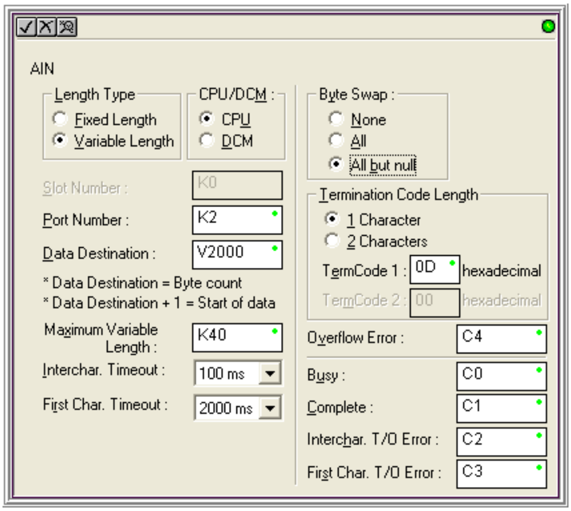
Fixed Length example when character to character timing is critical



AIN
CPU/DCM Slot : CPU
Port Number : K2
Data Destination : V2000
Maximum Variable Length : K40
Interchar. Timeout : 50ms
First Char. Timeout : 1000ms
Byte Swap : All
Termination Code(s) : 00
Overflow Error : C4
Busy : C0
Complete : C1
Interchar. Timeout Error : C2
First Char. Timeout Error : C3

AIN Variable Length Configuration:

- **Length Type**: select Variable Length if the ASCII string length followed by termination characters will vary in length
- **Port Number**: must be DL06 port 2 (K2)
- **Data Destination**: specifies where the ASCII string will be placed in V–memory
- **Maximum Variable Length**: specifies, in bytes, the maximum length of a Variable Length ASCII string the port will receive
- **Inter–character Timeout**: if the amount of time between incoming ASCII characters exceeds the set time, the Timeout Error bit will be set. No data will be stored at the Data Destination V–memory location.



AIN

Length Type: ○ Fixed Length ● Variable Length

CPU/DCM: ● CPU ○ DCM

Byte Swap: ○ None ○ All ● All but null

Slot Number: K0

Port Number: K2

Data Destination: V2000

* Data Destination = Byte count
* Data Destination + 1 = Start of data

Maximum Variable Length: K40

Interchar. Timeout: 100 ms

First Char. Timeout: 2000 ms

Termination Code Length: ● 1 Character ○ 2 Characters

TermCode 1: 0D hexadecimal

TermCode 2: 00 hexadecimal

Overflow Error: C4

Busy: C0

Complete: C1

Interchar. T/O Error: C2

First Char. T/O Error: C3

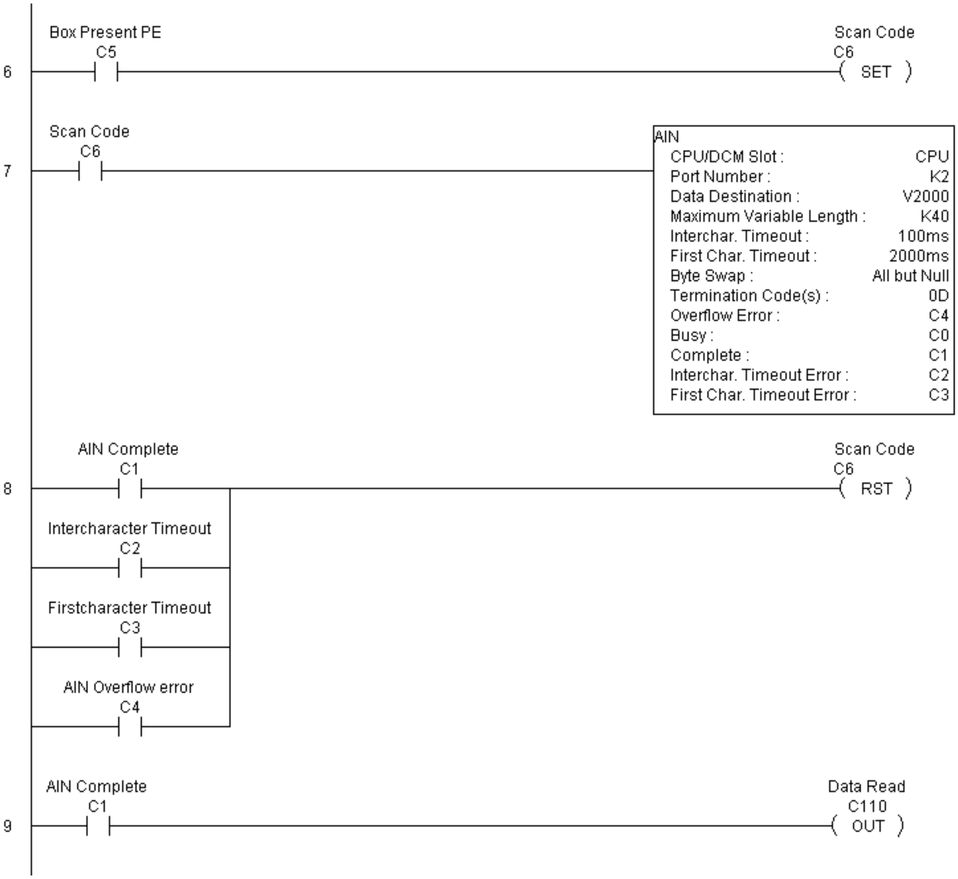
The Timeout Error bit will reset when the AIN instruction permissive bits are disabled. 0ms selection disables this feature.

- **First Character Timeout**: if the amount of time from when the AIN is enabled to the time the first character is received exceeds the set time, the specified First Character Timeout bit will be set. The bit will reset when the AIN instruction permissive bits are disabled. 0ms selection disables this feature.
- **Byte Swap**: swaps the high–byte and low–byte within each V–memory register of the Variable Length ASCII string. See the SWAPB instruction for details.
- **Termination Code Length**: consists of either 1 or 2 characters. Refer to Appendix G, ASCII Table.
- **Busy Bit**: is ON while the AIN instruction is receiving ASCII data
- **Complete Bit**: is set once the ASCII data has been received up to the termination code characters. It will be reset when the AIN instruction permissive bits are disabled.
- **Inter–character Timeout Error Bit**: is set when the Character Timeout is exceeded. See Character Timeout explanation above.
- **First Character Timeout Error Bit**: is set when the First Character Timeout is exceeded. See First Character Timeout explanation above.
- **Overflow Error Bit**: is set when the ASCII data received exceeds the Maximum Variable Length specified.

Parameter	
Data Destination	All V–memory
Fixed Length	K1–128
Bits: Busy, Complete, Timeout Error, Overflow	C0–3777

AIN Variable Length Example

AIN variable length example used to read barcodes on boxes (PE = photoelectric sensor)



AIN
CPU/DCM Slot : CPU
Port Number : K2
Data Destination : V2000
Maximum Variable Length : K40
Interchar. Timeout : 100ms
First Char. Timeout : 2000ms
Byte Swap : All but Null
Termination Code(s) : 0D
Overflow Error : C4
Busy : C0
Complete : C1
Interchar. Timeout Error : C2
First Char. Timeout Error : C3

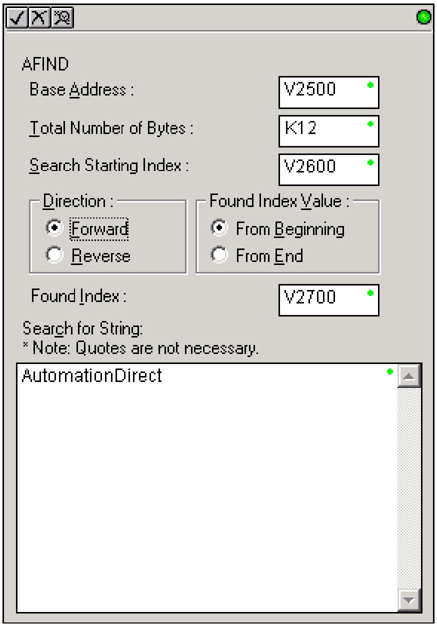
ASCII Find (AFIND)

DS	Used
HPP	N/A

The ASCII Find instruction locates a specific ASCII string or portion of an ASCII string within a range of V–memory registers and places the string's Found Index number (byte number where desired string is found), in Hex, into a specified V–memory register. Other features include, Search Starting Index number for skipping over unnecessary bytes before beginning the FIND operation, Forward or Reverse direction search, and From Beginning and From End selections to reference the Found Index Value.

- **Base Address**: specifies the beginning V–memory register where the entire ASCII string is stored in memory
- **Total Number of Bytes**: specifies the total number of bytes to search for the desired ASCII string
- **Search Starting Index**: specifies which byte to skip to (with respect to the Base Address) before beginning the search
- **Direction**: Forward begins the search from lower numbered V–memory registers to higher numbered V–memory registers. Reverse does the search from higher numbered V–memory registers to lower numbered V–memory registers.
- **Found Index Value**: specifies whether the Beginning or the End byte of the ASCII string found will be loaded into the Found Index register
- **Found Index**: specifies the V–memory register where the Found Index Value will be stored. A value of FFFF will result if the desired string is not located in the memory registers specified.
- **Search for String**: up to 128 characters.

Parameter	DL06 Range
Base Address	All V–memory
Total Number of Bytes	All V–memory or K1–128
Search Starting Index	All V–memory or K0–127
Found Index	All V–memory



AFIND

Base Address : V2500

Total Number of Bytes : K12

Search Starting Index : V2600

Direction : ● Forward ○ Reverse

Found Index Value : ● From Beginning ○ From End

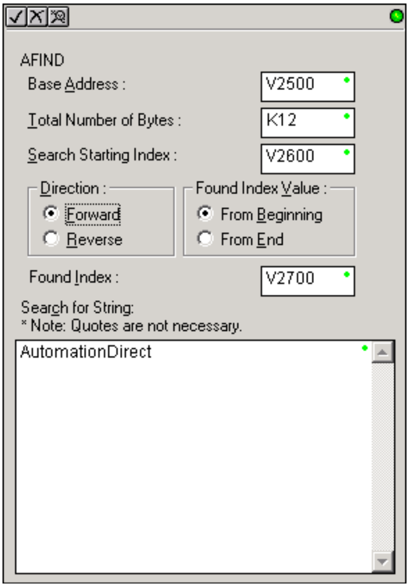
Found Index : V2700

Search for String:
* Note: Quotes are not necessary.

AutomationDirect

AFIND Search Example

In the following example, the AFIND instruction is used to search for the "day" portion of "Friday" in the ASCII string "Today is Friday.", which had previously been loaded into V–memory. Note that a Search Starting Index of constant (K) 5 combined with a Forward Direction Search is used to prevent finding the "day" portion of the word "Today". The Found Index will be placed into V4000.



AFIND

Base Address : V2500

Total Number of Bytes : K12

Search Starting Index : V2600

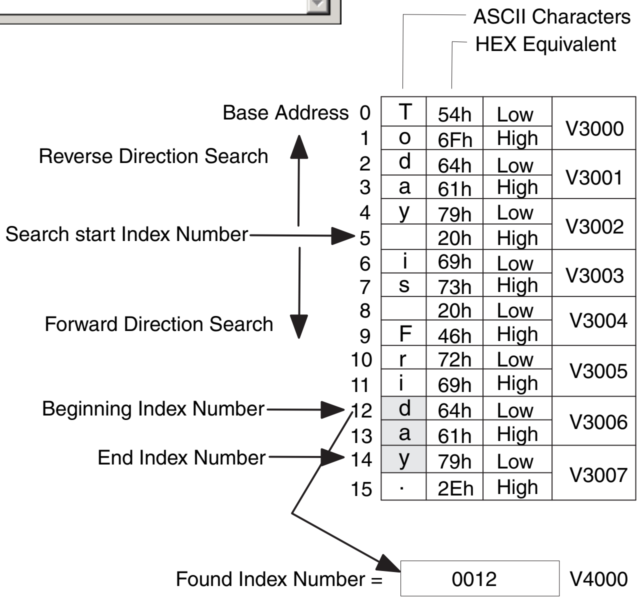
Direction : ● Forward ○ Reverse

Found Index Value : ● From Beginning ○ From End

Found Index : V2700

Search for String:
* Note: Quotes are not necessary.

AutomationDirect

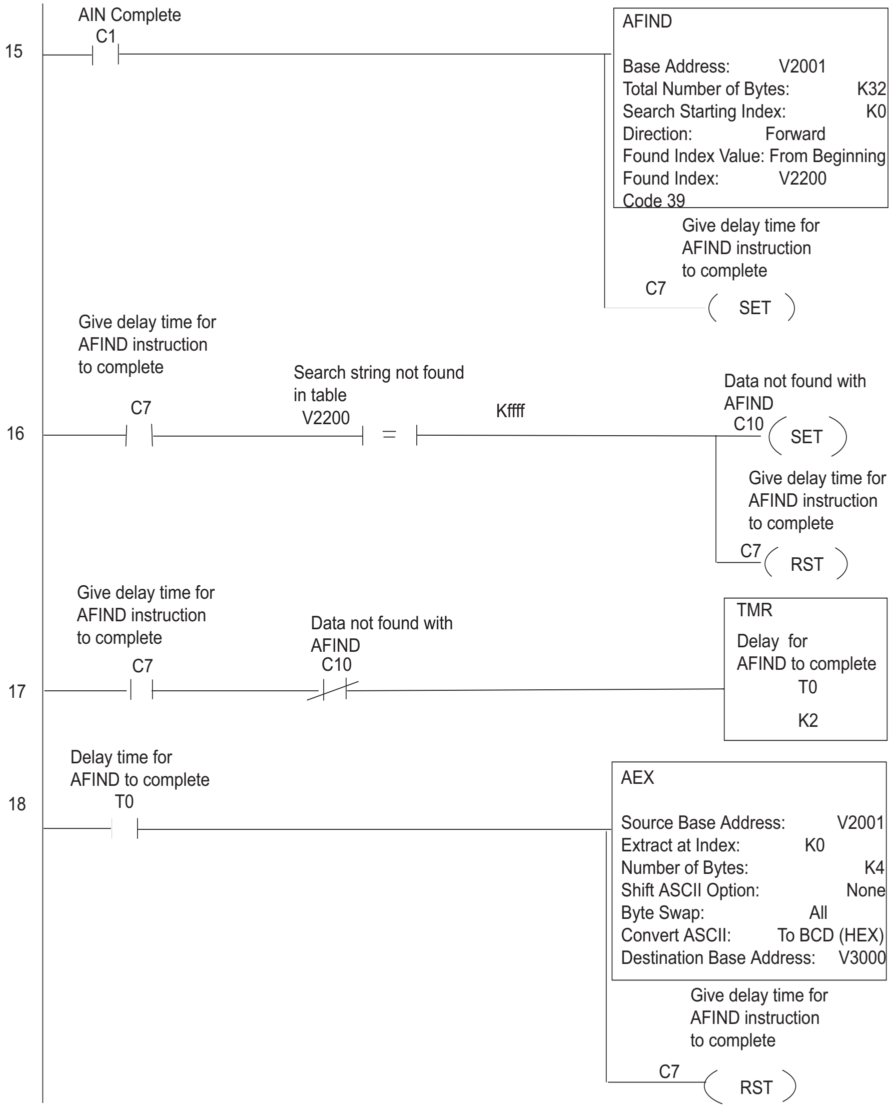


0	T	54h	Low	V3000
1	o	6Fh	High	
2	d	64h	Low	V3001
3	a	61h	High	
4	y	79h	Low	V3002
5		20h	High	
6	i	69h	Low	V3003
7	s	73h	High	
8		20h	Low	V3004
9	F	46h	High	
10	r	72h	Low	V3005
11	i	69h	High	
12	d	64h	Low	V3006
13	a	61h	High	
14	y	79h	Low	V3007
15	.	2Eh	High	

Found Index Number = 0012 V4000

AFIND Example Combined with AEX Instruction

When an AIN instruction has executed, its Complete bit can be used to trigger an AFIND instruction to search for a desired portion of the ASCII string. Once the string is found, the AEX instruction can be used to extract the located string.


15
AIN Complete
C1
AFIND
Base Address: V2001
Total Number of Bytes: K32
Search Starting Index: K0
Direction: Forward
Found Index Value: From Beginning
Found Index: V2200
Code 39
Give delay time for AFIND instruction to complete
C7
SET
16
Give delay time for AFIND instruction to complete
C7
Search string not found in table
V2200
=
Kffff
Data not found with AFIND
C10
SET
Give delay time for AFIND instruction to complete
C7
RST
17
Give delay time for AFIND instruction to complete
C7
Data not found with AFIND
C10
TMR
Delay for AFIND to complete
T0
K2
18
Delay time for AFIND to complete
T0
AEX
Source Base Address: V2001
Extract at Index: K0
Number of Bytes: K4
Shift ASCII Option: None
Byte Swap: All
Convert ASCII: To BCD (HEX)
Destination Base Address: V3000
Give delay time for AFIND instruction to complete
C7
RST

ASCII Extract (AEX)

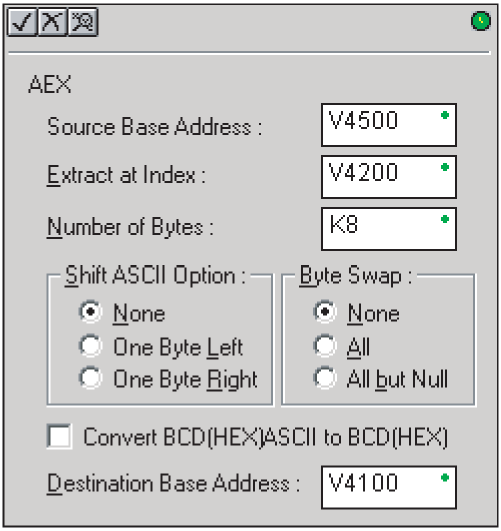
DS	Used
HPP	N/A

The ASCII Extract instruction extracts a specified number of bytes of ASCII data from one series of V–memory registers and places it into another series of V–memory registers. Other features include, Extract at Index for skipping over unnecessary bytes before beginning the Extract operation, Shift ASCII Option, for One Byte Left or One Byte Right, Byte Swap and Convert data to a BCD format number.

- **Source Base Address**: specifies the beginning V–memory register where the entire ASCII string is stored in memory
- **Extract at Index**: specifies which byte to skip to (with respect to the Source Base Address) before extracting the data
- **Number of Bytes**: specifies the number of bytes to be extracted
- **Shift ASCII Option**: shifts all extracted data one byte left or one byte right to displace "unwanted" characters if necessary
- **Byte Swap**: swaps the high–byte and the low–byte within each V–memory register of the extracted data. See the SWAPB instruction for details.
- **Convert BCD(Hex) ASCII to BCD (Hex)**: if enabled, this will convert ASCII numerical characters to Hexadecimal numerical values
- **Destination Base Address**: specifies the V–memory register where the extracted data will be stored

See the previous page for an example using the AEX instruction.

Parameter	DL06 Range	
Source Base Address	All V–memory	
Extract at Index	All V–memory or K0–127	
Number of Bytes "Convert BCD (HEX) ASCII" **not** checked	Constant range: K1–128	V-memory location containing BCD value: 1–128
Number of Bytes "Convert BCD (HEX) ASCII" checked	Constant range: K1–4	V-memory location containing BCD value: 1–4
Destination Base Address	All V–memory	



AEX

Source Base Address : V4500

Extract at Index : V4200

Number of Bytes : K8

Shift ASCII Option :
● None
○ One Byte Left
○ One Byte Right

Byte Swap :
● None
○ All
○ All but Null

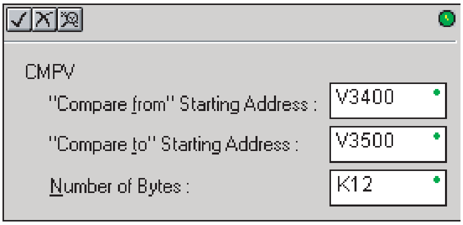
□ Convert BCD(HEX)ASCII to BCD(HEX)

Destination Base Address : V4100

ASCII Compare (CMPV)

DS	Used
HPP	N/A

The ASCII Compare instruction compares two groups of V–memory registers. The CMPV will compare any data type (ASCII to ASCII, BCD to BCD, etc.) of one series (group) of V–memory registers to another series of V–memory registers for a specified byte length.



CMPV

"Compare from" Starting Address : V3400

"Compare to" Starting Address : V3500

Number of Bytes : K12

SP61 = 1, the result is equal

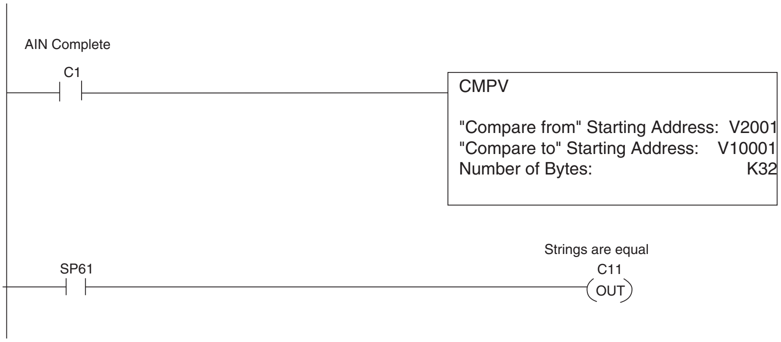
SP61 = 0, the result is not equal

- "Compare from" Starting Address: specifies the beginning V–memory register of the first group of V–memory registers to be compared from.
- "Compare to" Starting Address: specifies the beginning V–memory register of the second group of V–memory registers to be compared to.
- Number of Bytes: specifies the length of each V–memory group to be compared

Parameter	DL06 Range
Compare from Starting Address	All V–memory
Compare to Starting Address	All V–memory
Number of Bytes	K0–127

CMPV Example

The CMPV instruction executes when the AIN instruction is complete. If the compared V–memory tables are equal, SP61 will turn ON.

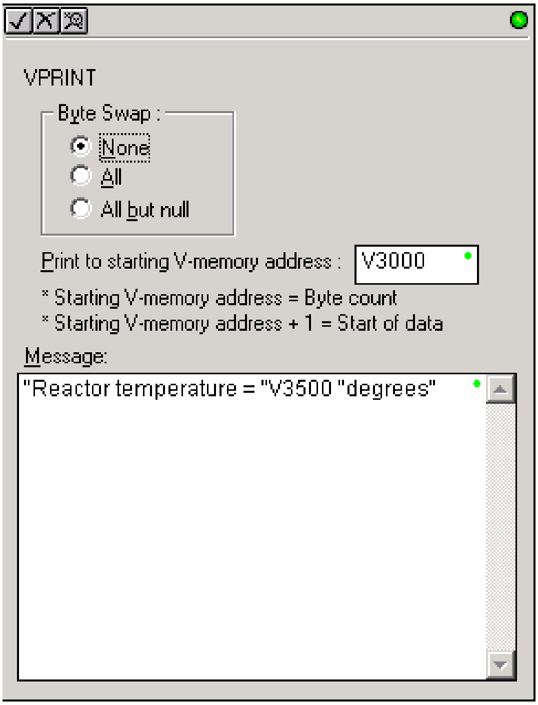

AIN Complete
C1
CMPV
"Compare from" Starting Address: V2001
"Compare to" Starting Address: V10001
Number of Bytes: K32
SP61
Strings are equal
C11
OUT

ASCII Print to V–memory (VPRINT)

DS	Used
HPP	N/A

The ASCII Print to V–memory instruction will write a specified ASCII string into a series of V–memory registers. Other features include Byte Swap, options to suppress or convert leading zeros or spaces, and _Date and _Time options for U.S., European, and Asian date formats and 12 or 24 hour time formats.

- **Byte Swap**: swaps the high–byte and low–byte within each V–memory register the ASCII string is printed to. See the SWAPB instruction for details.
- **Print to Starting V–memory Address**: specifies the beginning of a series of V–memory addresses where the ASCII string will be placed by the VPRINT instruction.
- **Starting V–memory Address**: the first V–memory register of the series of registers specified will contain the ASCII string's length in bytes.



VPRINT

Byte Swap :

● None
○ All
○ All but null

Print to starting V-memory address : V3000

* Starting V-memory address = Byte count
* Starting V-memory address + 1 = Start of data

Message:

"Reactor temperature = "V3500 "degrees"

- **Starting V–memory Address +1**: the 2nd and subsequent registers will contain the ASCII string printed to V–memory.

Parameter	DL06 Range
Print to Starting V–memory Address	All V–memory

VPRINT Time/Date Stamping– the codes in the table below can be used in the VPRINT ASCII string message to "print to V–memory" the current time and/or date.

#	Character code	Date / Time Stamp Options
1	_date:us	American standard (month/day/2 digit year)
2	_date:e	European standard (day/month/2 digit year)
3	_date:a	Asian standard (2 digit year/month/day)
4	_time:12	standard 12 hour clock (0–12 hour:min am/pm)
5	_time:24	standard 24 hour clock (0–23 hour:min am/pm)

VPRINT V-memory element – the following modifiers can be used in the VPRINT ASCII string message to “print to V–memory” register contents in integer format or real format. Use V-memory number or V-memory number with “:” and data type. The data types are shown in the table below. The Character code must be capital letters.

***NOTE**: There must be a space entered before and after the V-memory address to separate it from the text string. Failure to do this will result in an error code 499.*

#	Character code	Description
1	none	16-bit binary (decimal number)
2	: B	4 digit BCD
3	: D	32-bit binary (decimal number)
4	: D B	8 digit BCD
5	: R	Floating point number (real number)
6	: E	Floating point number (real number with exponent)

Examples:

V2000 Print binary data in V2000 for decimal number

V2000 : B Print BCD data in V2000

V2000 : D Print binary number in V2000 and V2001 for decimal number

V2000 : D B Print BCD data in V2000 and V2001

V2000 : R Print floating point number in V2000/V2001 as real number

V2000 : E Print floating point number in V2000/V2001 as real number with exponent

The following modifiers can be added to any of the modifies above to suppress or convert leading zeros or spaces. The character code must be capital letters.

#	Character code	Description
1	S	Suppresses leading spaces
2	C0	Converts leading spaces to zeros
3	0	Suppresses leading zeros

Example with V2000 = 0018 (binary format)

V–memory Register with Modifier	Number of Characters			
	1	2	3	4
V2000	0	0	1	8
V2000:B	0	0	1	2
V2000:B0	1	2		

Example with V2000 = sp sp18 (binary format) where sp = space

V–memory Register with Modifier	Number of Characters			
	1	2	3	4
V2000	sp	sp	1	8
V2000:B	sp	sp	1	2
V2000:BS	1	2		
V2000:BC0	0	0	1	2

VPRINT V-memory text element – the following is used for “printing to V–memory” text stored in registers. Use the % followed by the number of characters after V-memory number for representing the text. If you assign “0” as the number of characters, the function will read the character count from the first location. Then it will start at the next V-memory location and read that number of ASCII codes for the text from memory.

Example:

V2000 % 16 16 characters in V2000 to V2007 are printed.

V2000 % 0 The characters in V2001 to Vxxxx (determined by the number in V2000) will be printed.

VPRINT Bit element – the following is used for “printing to V–memory” the state of the designated bit in V-memory or a control relay bit. The bit element can be assigned by the designating point (.) and bit number preceded by the V-memory number or relay number. The output type is described as shown in the table below.

#	Data format	Description
1	none	Print 1 for an ON state, and 0 for an OFF state
2	: BOOL	Print “TRUE” for an ON state, and “FALSE” for an OFF state
3	: ONOFF	Print “ON” for an ON state, and “OFF” for an OFF state

Example:

V2000 . 15 Prints the status of bit 15 in V2000, in 1/0 format

C100 Prints the status of C100 in 1/0 format

C100 : BOOL Prints the status of C100 in TRUE/FALSE format

C100 : ON/OFF Prints the status of C100 in ON/OFF format

V2000.15 : BOOL Prints the status of bit 15 in V2000 in TRUE/FALSE format

The maximum numbers of characters you can VPRINT is 128. The number of characters required for each element, regardless of whether the :S, :C0 or :0 modifiers are used, is listed in the table below.

Element type	Maximum Characters
Text, 1 character	1
16 bit binary	6
32 bit binary	11
4 digit BCD	4
8 digit BCD	8
Floating point (real number)	3
Floating point (real with exponent)	13
V-memory/text	2
Bit (1/0 format)	1
Bit (TRUE/FALSE format)	5
Bit (ON/OFF format)	3

Text element – the following is used for “printing to V–memory” character strings. The character strings are defined as the character (more than 0) ranged by the double quotation marks. Two hex numbers preceded by the dollar sign means an 8-bit ASCII character code. Also, two characters preceded by the dollar sign is interpreted according to the following table:

#	Character code	Description
1	$$	Dollar sign ($)
2	$”	Double quotation (”)
3	$Lor $l	Line feed (LF)
4	$N or $n	Carriage return line feed (CRLF)
5	$P or $p	Form feed
6	$R or $r	Carriage return (CR)
7	$T or $t	Tab

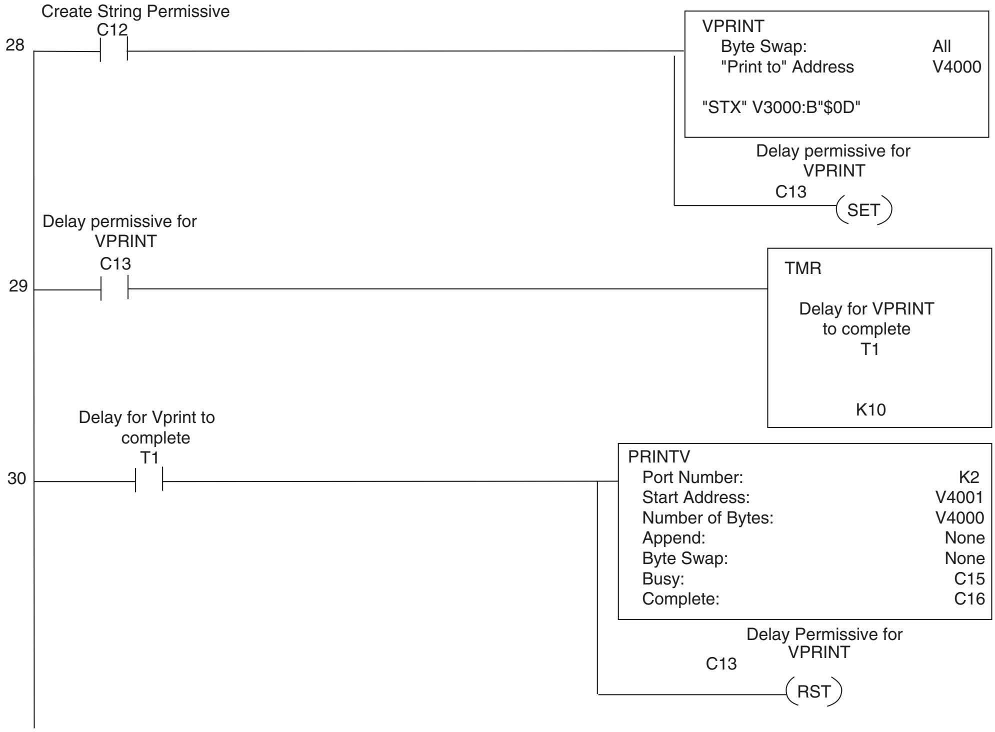
The following examples show various syntax conventions and the length of the output to the printer.

” ”	Length 0 without character
”A”	Length 1 with character A
” ”	Length 1 with blank
” $” ”	Length 1 with double quotation mark
” $ R $ L ”	Length 2 with one CR and one LF
” $ 0 D $ 0 A ”	Length 2 with one CR and one LF
” $ $ ”	Length 1 with one $ mark

In printing an ordinary line of text, you will need to include double quotation marks before and after the text string. Error code 499 will occur in the CPU when the print instruction contains invalid text or no quotations. It is important to test your VPRINT instruction data during the application development.

VPRINT Example Combined with PRINTV Instruction

The VPRINT instruction is used to create a string in V–memory. The PRINTV is used to print the string out of port 2.

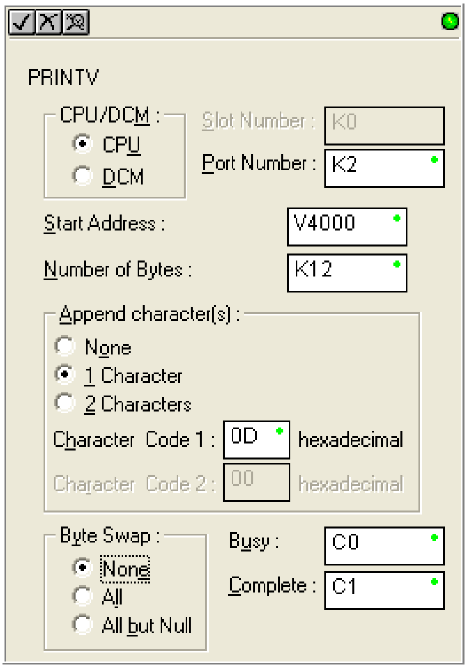


ASCII Print from V–memory (PRINTV)

DS	Used
HPP	N/A

The ASCII Print from V–memory instruction will send an ASCII string out of the designated communications port from a specified series of V–memory registers for a specified length in number of bytes. Other features include user specified Append Characters to be placed after the desired data string for devices that require specific termination character(s), Byte Swap options, and user specified flags for Busy and Complete.

- **Port Number**: must be DL06 port 2 (K2)
- **Start Address**: specifies the beginning of series of V–memory registers that contain the ASCII string to print
- **Number of Bytes**: specifies the length of the string to print
- **Append Characters**: specifies ASCII characters to be added to the end of the string for devices that require specific termination characters
- **Byte Swap**: swaps the high–byte and low–byte within each V–memory register of the string while printing. See the SWAPB instruction for details.
- **Busy Bit**: will be ON while the instruction is printing ASCII data
- **Complete Bit**: will be set once the ASCII data has been printed and reset when the PRINTV instruction permissive bits are disabled.



PRINTV

CPU/DCM : ● CPU ○ DCM

Slot Number : K0

Port Number : K2

Start Address : V4000

Number of Bytes : K12

Append character(s) :
○ None
● 1 Character
○ 2 Characters

Character Code 1 : 0D hexadecimal

Character Code 2 : 00 hexadecimal

Byte Swap : ● None ○ All ○ All but Null

Busy : C0

Complete : C1

See the previous page for an example using the PRINTV instruction.

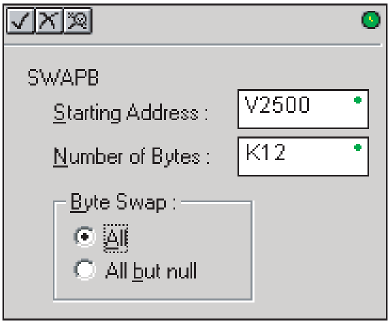
Parameter	DL06 Range
Port Number	port 2 (K2)
Start Address	All V–memory
Number of Bytes	All V–memory or k1–128
Bits: Busy, Complete	C0–3777

ASCII Swap Bytes (SWAPB)

DS	Used
HPP	N/A

The ASCII Swap Bytes instruction swaps byte positions (high–byte to low–byte and low–byte to high–byte) within each V–memory register of a series of V–memory registers for a specified number of bytes.

- **Starting Address:** specifies the beginning of a series of V–memory registers the instruction will use to begin byte swapping
- **Number of Bytes:** specifies the number of bytes, beginning with the Starting Address, to byte swap.
- **Byte Swap:**
  All - swap all bytes specified.
  All but null - swap all bytes specified except the bytes with a null


SWAPB
Starting Address : V2500
Number of Bytes : K12
Byte Swap :
● All
○ All but null

Parameter	DL06 Range
Starting Address	All V–memory
Number of Bytes	All V–memory or K1–128

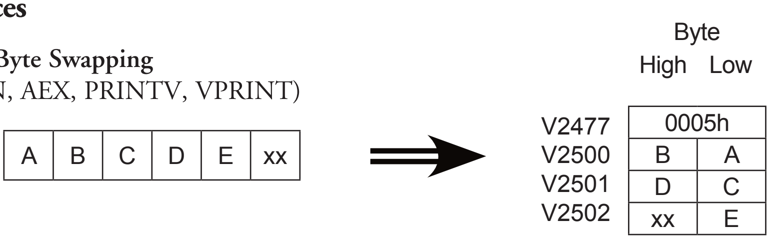
5

Discrete Bit Flags	Description
SP53	On if the CPU cannot execute the instruction.
SP71	On when a value used by the instruction is invalid.

Byte Swap Preferences

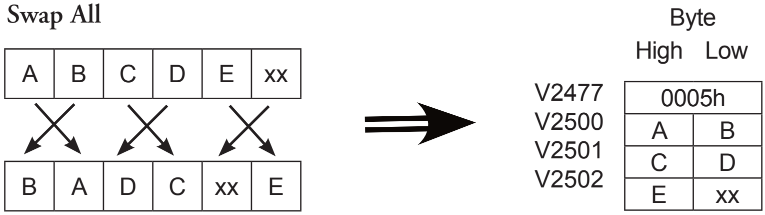
No Byte Swapping

(AIN, AEX, PRINTV, VPRINT)



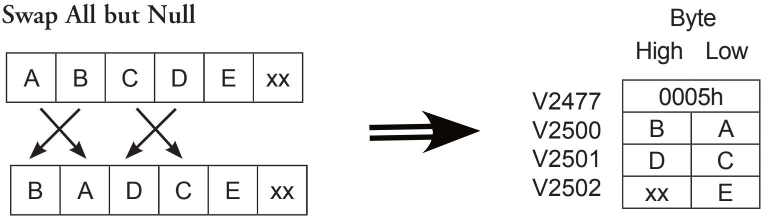
	High	Low
V2477	0005h	
V2500	B	A
V2501	D	C
V2502	xx	E

Byte Swap All



	High	Low
V2477	0005h	
V2500	A	B
V2501	C	D
V2502	E	xx

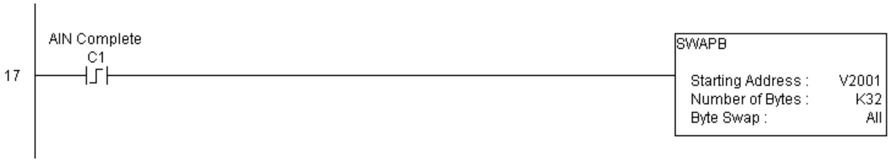
Byte Swap All but Null



	High	Low
V2477	0005h	
V2500	B	A
V2501	D	C
V2502	xx	E

SWAPB Example

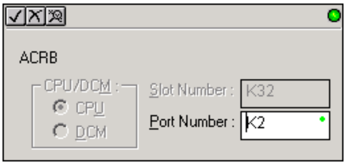
The AIN Complete bit is used to trigger the SWAPB instruction. Use a one–shot so the SWAPB only executes once.



ASCII Clear Buffer (ACRB)

DS	Used
HPP	N/A

The ASCII Clear Buffer instruction will clear the ASCII receive buffer of the specified communications port number. Port Number: must be DL06 port 2 (K2)


ACRB

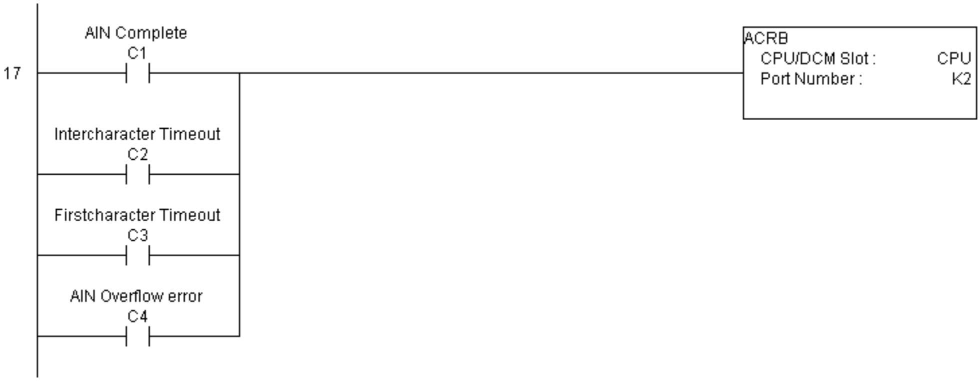
CPU/DCM :
● CPU
○ DCM

Slot Number : K32

Port Number : K2

ACRB Example

The AIN Complete bit or the AIN diagnostic bits are used to clear the ASCII buffer.



This page intentionally left blank.

Intelligent Box (IBox) Instructions

The Intelligent Box Instructions (IBox) listed in this section are additional instructions made available when using *Direct*SOFT to program your DL06 PLC (the DL06 CPU requires firmware version v2.10 or later to use the new features in *Direct*SOFT). For more information on *Direct*SOFT and to download a free demo version, please visit our Web site at: www.automationdirect.com.

Analog Helper IBoxes		
Instruction	**Ibox #**	**Page**
Analog Input / Output Combo Module Pointer Setup (ANLGCMB)	IB-462	5-232
Analog Input Module Pointer Setup (ANLGIN)	IB-460	5-234
Analog Output Module Pointer Setup (ANLGOUT)	IB-461	5-236
Analog Scale 12 Bit BCD to BCD (ANSCL)	IB-423	5-238
Analog Scale 12 Bit Binary to Binary (ANSCLB)	IB-403	5-239
Filter Over Time - BCD (FILTER)	IB-422	5-240
Filter Over Time - Binary (FILTERB)	IB-402	5-242
Hi/Low Alarm - BCD (HILOAL)	IB-421	5-244
Hi/Low Alarm - Binary (HILOALB)	IB-401	5-246

Discrete Helper IBoxes		
Instruction	**Ibox #**	**Page**
Off Delay Timer (OFFDTMR)	IB-302	5-248
On Delay Timer (ONDTMR)	IB-301	5-250
One Shot (ONESHOT)	IB-303	5-252
Push On / Push Off Circuit (PONOFF)	IB-300	5-253

Memory IBoxes		
Instruction	**Ibox #**	**Page**
Move Single Word (MOVEW)	IB-200	5-254
Move Double Word (MOVED)	IB-201	5-255

Math IBoxes		
Instruction	**Ibox #**	**Page**
BCD to Real with Implied Decimal Point (BCDTOR)	IB-560	5-256
Double BCD to Real with Implied Decimal Point (BCDTORD)	IB-562	5-257
Math - BCD (MATHBCD)	IB-521	5-258
Math - Binary (MATHBIN)	IB-501	5-260
Math - Real (MATHR)	IB-541	5-262
Real to BCD with Implied Decimal Point and Rounding (RTOBCD)	IB-561	5-263
Real to Double BCD with Implied Decimal Point and Rounding (RTOBCDD)	IB-563	5-264
Square BCD (SQUARE)	IB-523	5-265
Square Binary (SQUAREB)	IB-503	5-266
Square Real(SQUARER)	IB-543	5-267
Sum BCD Numbers (SUMBCD)	IB-522	5-268
Sum Binary Numbers (SUMBIN)	IB-502	5-269
Sum Real Numbers (SUMR)	IB-542	5-270

Communication IBoxes		
Instruction	**Ibox #**	**Page**
ECOM100 Configuration (ECOM100)	IB-710	5-272
ECOM100 Disable DHCP (ECDHCPD)	IB-736	5-274
ECOM100 Enable DHCP (ECDHCPE)	IB-735	5-276
ECOM100 Query DHCP Setting (ECDHCPQ)	IB-734	5-278
ECOM100 Send E-mail (ECEMAIL)	IB-711	5-280
ECOM100 Restore Default E-mail Setup (ECEMRDS)	IB-713	5-283
ECOM100 E-mail Setup (ECEMSUP)	IB-712	5-286
ECOM100 IP Setup (ECIPSUP)	IB-717	5-290
ECOM100 Read Description (ECRDDES)	IB-726	5-292
ECOM100 Read Gateway Address (ECRDGWA)	IB-730	5-294
ECOM100 Read IP Address (ECRDIP)	IB-722	5-296
ECOM100 Read Module ID (ECRDMID)	IB-720	5-298
ECOM100 Read Module Name (ECRDNAM)	IB-724	5-300
ECOM100 Read Subnet Mask (ECRDSNM)	IB-732	5-302
ECOM100 Write Description (ECWRDES)	IB-727	5-304
ECOM100 Write Gateway Address (ECWRGWA)	IB-731	5-306
ECOM100 Write IP Address (ECWRIP)	IB-723	5-308
ECOM100 Write Module ID (ECWRMID)	IB-721	5-310
ECOM100 Write Name (ECWRNAM)	IB-725	5-312
ECOM100 Write Subnet Mask (ECWRSNM)	IB-733	5-314
ECOM100 RX Network Read (ECRX)	IB-740	5-316
ECOM100 WX Network Write(ECWX)	IB-741	5-319
NETCFG Network Configuration (NETCFG)	IB-700	5-322
Network RX Read (NETRX)	IB-701	5-324
Network WX Write (NETWX)	IB-702	5-327

Counter I/O IBoxes (Works with H0-CTRIO and H0-CTRIO2)		
Instruction	**Ibox #**	**Page**
CTRIO Configuration (CTRIO)	IB-1000	5-330
CTRIO Add Entry to End of Preset Table (CTRADPT)	IB-1005	5-332
CTRIO Clear Preset Table (CTRCLRT)	IB-1007	5-335
CTRIO Edit Preset Table Entry (CTREDPT)	IB-1003	5-338
CTRIO Edit Preset Table Entry and Reload (CTREDRL)	IB-1002	5-342
CTRIO Initialize Preset Table (CTRINPT)	IB-1004	5-346
CTRIO Initialize Preset Table (CTRINTR)	IB-1010	5-350
CTRIO Load Profile (CTRLDPR)	IB-1001	5-354
CTRIO Read Error (CTRRDER)	IB-1014	5-357
CTRIO Run to Limit Mode (CTRRTLM)	IB-1011	5-359
CTRIO Run to Position Mode (CTRRTPM)	IB-1012	5-362
CTRIO Velocity Mode (CTRVELO)	IB-1013	5-365
CTRIO Write File to ROM (CTRWFTR)	IB-1006	5-368

Analog Input/Output Combo Module Pointer Setup (ANLGCMB) (IB-462)

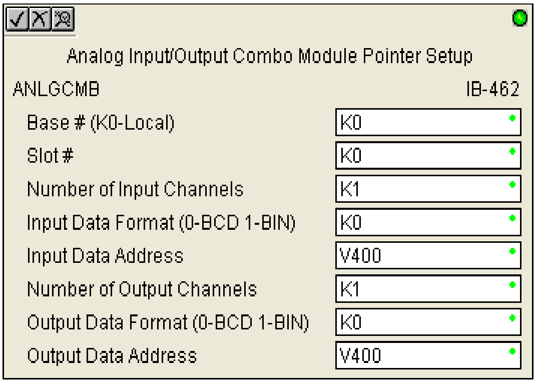
DS	Used
HPP	N/A

The Analog Input/Output Combo Module Pointer Setup instruction generates the logic to configure the pointer method for an analog input/output combination module on the first PLC scan following a Program to Run transition.

The ANLGCMB IBox instruction determines the data format and Pointer addresses based on the CPU type, the Base# and the module Slot#.

The Input Data Address is the starting location in user V-memory where the analog input data values will be stored, one location for each input channel enabled.

The Output Data Address is the starting location in user V-memory where the analog output data values will be placed by ladder code or external device, one location for each output channel enabled



Analog Input/Output Combo Module Pointer Setup

ANLGCMB IB-462

Base # (K0-Local)	K0
Slot #	K0
Number of Input Channels	K1
Input Data Format (0-BCD 1-BIN)	K0
Input Data Address	V400
Number of Output Channels	K1
Output Data Format (0-BCD 1-BIN)	K0
Output Data Address	V400

Since the IBox logic only executes on the first scan, the instruction cannot have any input logic.

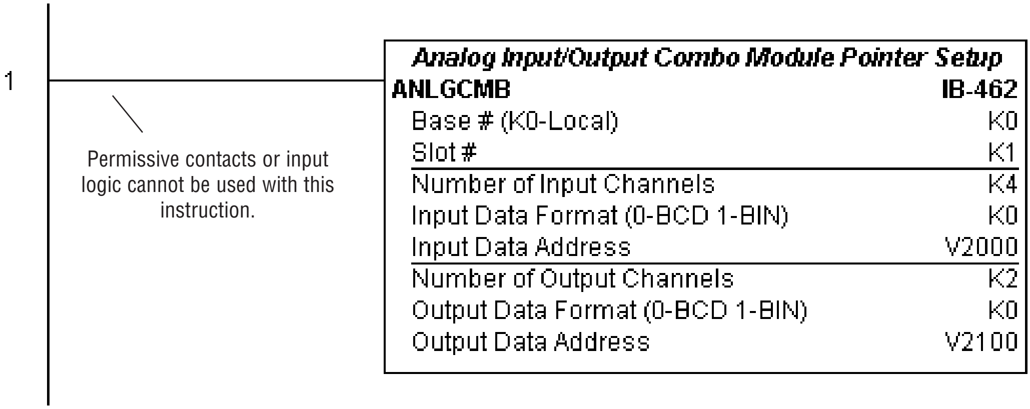
ANLGCMB Parameters

- Base # (K0-Local): must be 0 for DL06 PLC
- Slot #: specifies which PLC option slot is occupied by the analog module (1–4)
- Number of Input Channels: specifies the number of analog input channels to scan
- Input Data Format (0-BCD 1-BIN): specifies the analog input data format (BCD or Binary) - the binary format may be used for displaying data on some OI panels
- Input Data Address: specifies the starting V-memory location that will be used to store the analog input data
- Number of Output Channels: specifies the number of analog output channels that will be used
- Output Data Format (0-BCD 1-BIN): specifies the format of the analog output data (BCD or Binary)
- Output Data Address: specifies the starting V-memory location that will be used to source the analog output data

Parameter		DL06 Range
Base # (K0-Local)	K	K0 (local base only)
Slot #	K	K1-4
Number of Input Channels	K	K1-8
Input Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Input Data Address	V	See DL06 V-memory map - Data Words
Number of Output Channels	K	K1-8
Output Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Output Data Address	V	See DL06 V-memory map - Data Words

ANLGCMB Example

In the following example, the ANLGCMB instruction is used to setup the pointer method for an analog I/O combination module that is installed in option slot 2. Four input channels are enabled and the analog data will be written to V2000 - V2003 in BCD format. Two output channels are enabled and the analog values will be read from V2100 - V2101 in BCD format.



Permissive contacts or input logic cannot be used with this instruction.

Analog Input/Output Combo Module Pointer Setup

ANLGCMB **IB-462**

Base # (K0-Local) K0

Slot # K1

Number of Input Channels K4

Input Data Format (0-BCD 1-BIN) K0

Input Data Address V2000

Number of Output Channels K2

Output Data Format (0-BCD 1-BIN) K0

Output Data Address V2100

Analog Input Module Pointer Setup (ANLGIN) (IB-460)

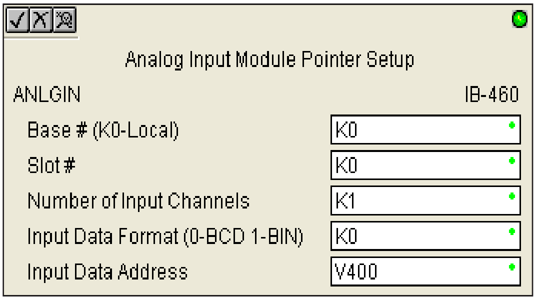
DS	Used
HPP	N/A

Analog Input Module Pointer Setup generates the logic to configure the pointer method for one analog input module on the first PLC scan following a Program to Run transition.

This IBox determines the data format and Pointer addresses based on the CPU type, the Base#, and the Slot#.

The Input Data Address is the starting location in user V-memory where the analog input data values will be stored, one location for each input channel enabled.

Since this logic only executes on the first scan, this IBox cannot have any input logic.


Analog Input Module Pointer Setup

ANLGIN IB-460

Base # (K0-Local) K0

Slot # K0

Number of Input Channels K1

Input Data Format (0-BCD 1-BIN) K0

Input Data Address V400

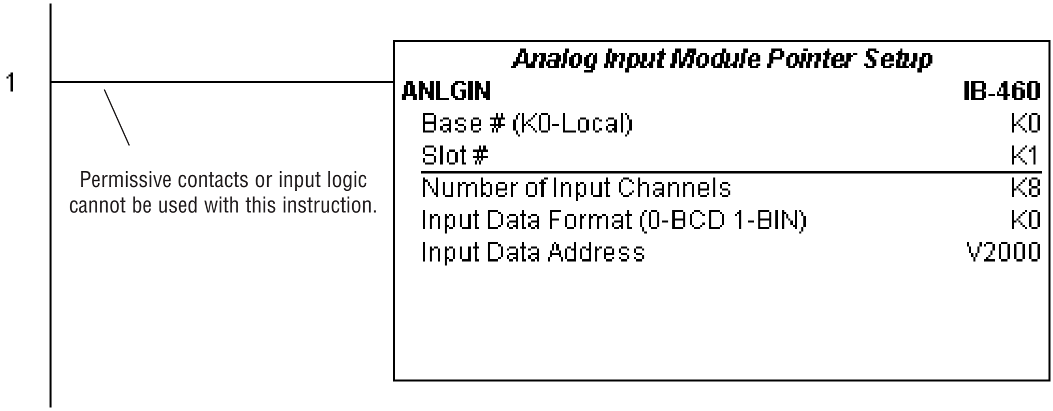
ANLGIN Parameters

- Base # (K0-Local): must be 0 for DL06 PLC
- Slot #: specifies which PLC option slot is occupied by the analog module (1–4)
- Number of Input Channels: specifies the number of input channels to scan
- Input Data Format (0-BCD 1-BIN): specifies the analog input data format (BCD or Binary) - the binary format may be used for displaying data on some OI panels
- Input Data Address: specifies the starting V-memory location that will be used to store the analog input data

Parameter		DL06 Range
Base # (K0-Local)	K	K0 (local base only)
Slot #	K	K1-4
Number of Input Channels	K	K1-8
Input Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Input Data Address	V	See DL06 V-memory map - Data Words

ANLGIN Example

In the following example, the ANLGIN instruction is used to setup the pointer method for an analog input module that is installed in option slot 1. Eight input channels are enabled and the analog data will be written to V2000 - V2007 in BCD format.



Permissive contacts or input logic cannot be used with this instruction.

Analog Input Module Pointer Setup

ANLGIN **IB-460**

Base # (K0-Local) K0

Slot # K1

Number of Input Channels K8

Input Data Format (0-BCD 1-BIN) K0

Input Data Address V2000

Analog Output Module Pointer Setup (ANLGOUT) (IB-461)

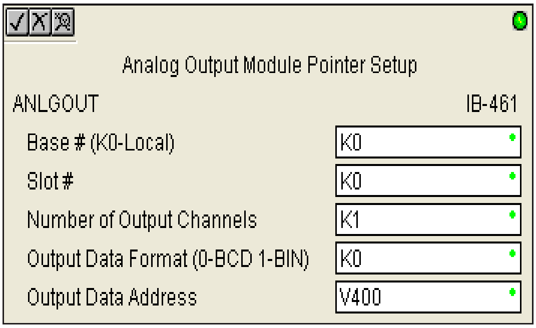
DS	Used
HPP	N/A

Analog Output Module Pointer Setup generates the logic to configure the pointer method for one analog output module on the first PLC scan following a Program to Run transition.

This IBox determines the data format and Pointer addresses based on the CPU type, the Base#, and the Slot#.

The Output Data Address is the starting location in user V-memory where the analog output data values will be placed by ladder code or external device, one location for each output channel enabled.

Since this logic only executes on the first scan, this IBox cannot have any input logic.



Analog Output Module Pointer Setup

ANLGOUT IB-461

Base # (K0-Local) K0

Slot # K0

Number of Output Channels K1

Output Data Format (0-BCD 1-BIN) K0

Output Data Address V400

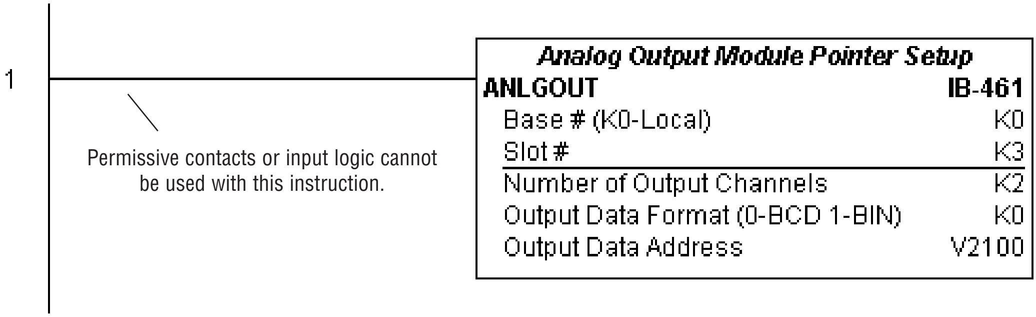
ANLGOUT Parameters

- Base # (K0-Local): must be 0 for DL06 PLC
- Slot #: specifies which PLC option slot is occupied by the analog module (1–4)
- Number of Output Channels: specifies the number of analog output channels that will be used
- Output Data Format (0-BCD 1-BIN): specifies the format of the analog output data (BCD or Binary)
- Output Data Address: specifies the starting V-memory location that will be used to source the analog output data

Parameter		DL06 Range
Base # (K0-Local)	K	K0 (local base only)
Slot #	K	K1-4
Number of Output Channels	K	K1-8
Output Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Output Data Address	V	See DL06 V-memory map - Data Words

ANLGOUT Example

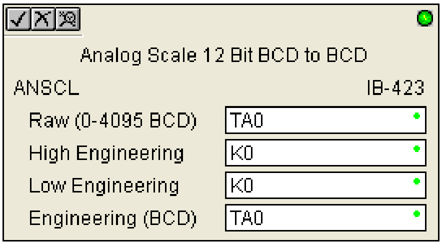
In the following example, the ANLGOUT instruction is used to setup the pointer method for an analog output module that is installed in option slot 3. Two output channels are enabled and the analog data will be read from V2100 - V2101 in BCD format.


1
Permissive contacts or input logic cannot be used with this instruction.
Analog Output Module Pointer Setup
ANLGOUT
IB-461
Base # (K0-Local)
K0
Slot #
K3
Number of Output Channels
K2
Output Data Format (0-BCD 1-BIN)
K0
Output Data Address
V2100

Analog Scale 12 Bit BCD to BCD (ANSCL) (IB-423)

DS	Used
HPP	N/A

Analog Scale 12 Bit BCD to BCD scales a 12 bit BCD analog value (0-4095 BCD) into BCD engineering units. You specify the engineering unit high value (when raw is 4095), and the engineering low value (when raw is 0), and the output V-Memory address you want the to place the scaled engineering unit value. The engineering units are generated as BCD and can be the full range of 0 to 9999 (see ANSCLB - Analog Scale 12 Bit Binary to Binary if your raw units are in Binary format).



Analog Scale 12 Bit BCD to BCD

ANSCL IB-423

Raw (0-4095 BCD) TA0

High Engineering K0

Low Engineering K0

Engineering (BCD) TA0

NOTE: *This IBox only works with unipolar unsigned raw values. It does NOT work with bipolar or sign plus magnitude raw values.*

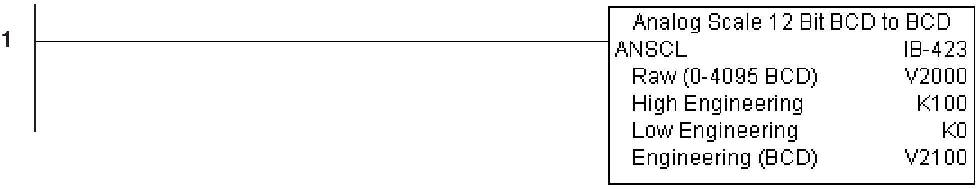
ANSCL Parameters

- Raw (0-4095 BCD): specifies the V-memory location of the unipolar unsigned raw 0-4095 unscaled value
- High Engineering: specifies the high engineering value when the raw input is 4095
- Low Engineering: specifies the low engineering value when the raw input is 0
- Engineering (BCD): specifies the V-memory location where the scaled engineering BCD value will be placed

Parameter		DL06 Range
Raw (0-4095 BCD)	V,P	See DL06 V-memory map - Data Words
High Engineering	K	K0-9999
Low Engineering	K	K0-9999
Engineering (BCD)	V,P	See DL06 V-memory map - Data Words

ANSCL Example

In the following example, the ANSCL instruction is used to scale a raw value (0-4095 BCD) that is in V2000. The engineering scaling range is set 0-100 (low engineering value - high engineering value). The scaled value will be placed in V2100 in BCD format.



Analog Scale 12 Bit BCD to BCD

ANSCL IB-423

Raw (0-4095 BCD) V2000

High Engineering K100

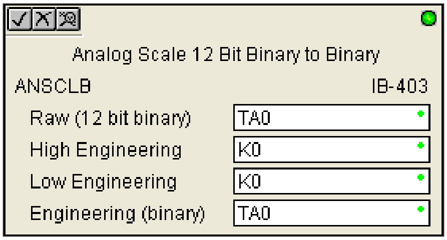
Low Engineering K0

Engineering (BCD) V2100

Analog Scale 12 Bit Binary to Binary (ANSCLB) (IB-403)

DS	Used
HPP	N/A

Analog Scale 12 Bit Binary to Binary scales a 12 bit binary analog value (0-4095 decimal) into binary (decimal) engineering units. You specify the engineering unit high value (when raw is 4095), and the engineering low value (when raw is 0), and the output V-Memory address you want to place the scaled engineering unit value. The engineering units are generated as binary and can be the full range of 0 to 65535 (see ANSCL - Analog Scale 12 Bit BCD to BCD if your raw units are in BCD format).


Analog Scale 12 Bit Binary to Binary
ANSCLB IB-403
Raw (12 bit binary) TA0
High Engineering K0
Low Engineering K0
Engineering (binary) TA0

NOTE*: This IBox only works with unipolar unsigned raw values. It does NOT work with bipolar, sign plus magnitude, or signed 2’s complement raw values.*

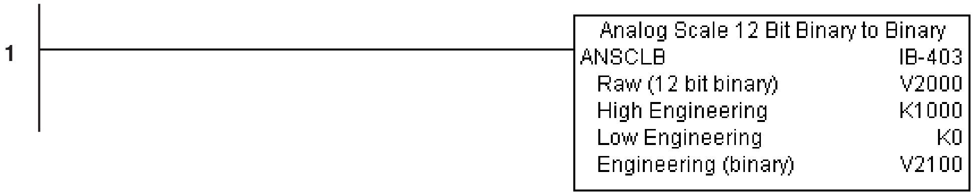
ANSCLB Parameters

- Raw (12 bit binary): specifies the V-memory location of the unipolar unsigned raw decimal unscaled value (12 bit binary = 0-4095 decimal)
- High Engineering: specifies the high engineering value when the raw input is 4095 decimal
- Low Engineering: specifies the low engineering value when the raw input is 0 decimal
- Engineering (binary): specifies the V-memory location where the scaled engineering decimal value will be placed

Parameter		DL06 Range
Raw (12 bit binary)	V,P	See DL06 V-memory map - Data Words
High Engineering	K	K0-65535
Low Engineering	K	K0-65535
Engineering (binary)	V,P	See DL06 V-memory map - Data Words

ANSCLB Example

In the following example, the ANSCLB instruction is used to scale a raw value (0-4095 binary) that is in V2000. The engineering scaling range is set 0-1000 (low engineering value - high engineering value). The scaled value will be placed in V2100 in binary format.



Analog Scale 12 Bit Binary to Binary
ANSCLB IB-403
Raw (12 bit binary) V2000
High Engineering K1000
Low Engineering K0
Engineering (binary) V2100

Filter Over Time - BCD (FILTER) (IB-422)

DS	Used
HPP	N/A

Filter Over Time BCD will perform a first-order filter on the Raw Data on a defined time interval. The equation is:

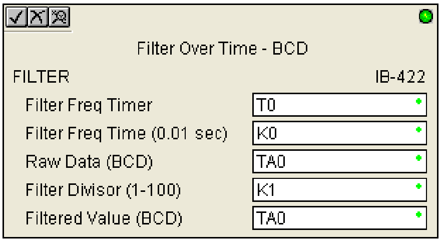
New = Old + [(Raw - Old) / FDC]
where,

New: New Filtered Value

Old: Old Filtered Value

FDC: Filter Divisor Constant

Raw: Raw Data



Filter Over Time - BCD

FILTER IB-422

Filter Freq Timer	T0
Filter Freq Time (0.01 sec)	K0
Raw Data (BCD)	TA0
Filter Divisor (1-100)	K1
Filtered Value (BCD)	TA0

The Filter Divisor Constant is an integer in the range K1 to K100, such that if it equaled K1 then no filtering would be done.

The rate at which the calculation is performed is specified by time in hundredths of a second (0.01 seconds) as the Filter Freq Time parameter. Note that this Timer instruction is embedded in the IBox and must NOT be used anywhere else in your program. Power flow controls whether the calculation is enabled. If it is disabled, the Filter Value is not updated. On the first scan from Program to Run mode, the Filter Value is initialized to 0 to give the calculation a consistent starting point.

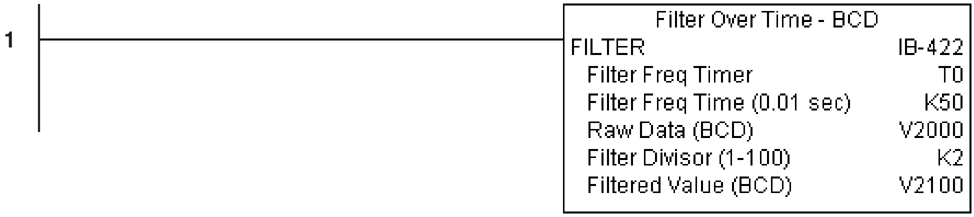
FILTER Parameters

- Filter Frequency Timer: specifies the Timer (T) number which is used by the Filter instruction
- Filter Frequency Time (0.01sec): specifies the rate at which the calculation is performed
- Raw Data (BCD): specifies the V-memory location of the raw unfiltered BCD value
- Filter Divisor (1-100): this constant used to control the filtering effect. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering.
- Filtered Value (BCD): specifies the V-memory location where the filtered BCD value will be placed

Parameter		DL06 Range
Filter Frequency Timer	T	T0-377
Filter Frequency Time (0.01 sec)	K	K0-9999
Raw Data (BCD)	V	See DL06 V-memory map - Data Words
Filter Divisor (1-100)	K	K1-100
Filtered Value (BCD)	V	See DL06 V-memory map - Data Words

FILTER Example

In the following example, the Filter instruction is used to filter a BCD value that is in V2000. Timer(T0) is set to 0.5 sec, the rate at which the filter calculation will be performed. The filter constant is set to 2. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering. The filtered value will be placed in V2100.



```
        Filter Over Time - BCD
FILTER                          IB-422
  Filter Freq Timer                 T0
  Filter Freq Time (0.01 sec)      K50
  Raw Data (BCD)                 V2000
  Filter Divisor (1-100)            K2
  Filtered Value (BCD)           V2100
```

Filter Over Time - Binary (FILTERB) (IB-402)

DS	Used
HPP	N/A

Filter Over Time in Binary (decimal) will perform a first-order filter on the Raw Data on a defined time interval. The equation is

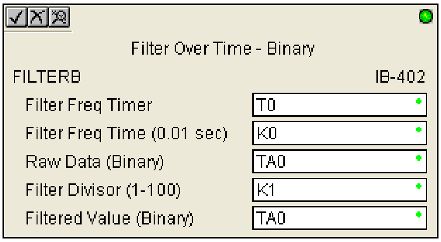
New = Old + [(Raw - Old) / FDC] where

New: New Filtered Value

Old: Old Filtered Value

FDC: Filter Divisor Constant

Raw: Raw Data



Filter Over Time - Binary

FILTERB IB-402

Filter Freq Timer: T0

Filter Freq Time (0.01 sec): K0

Raw Data (Binary): TA0

Filter Divisor (1-100): K1

Filtered Value (Binary): TA0

The Filter Divisor Constant is an integer in the range K1 to K100, such that if it equaled K1 then no filtering would be done.

The rate at which the calculation is performed is specified by time in hundredths of a second (0.01 seconds) as the Filter Freq Time parameter. Note that this Timer instruction is embedded in the IBox and must NOT be used any other place in your program. Power flow controls whether the calculation is enabled. If it is disabled, the Filter Value is not updated. On the first scan from Program to Run mode, the Filter Value is initialized to 0 to give the calculation a consistent starting point.

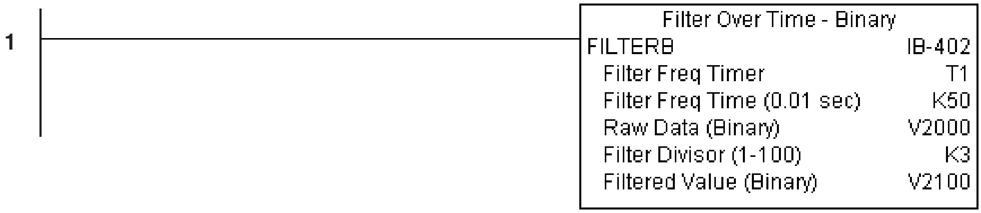
FILTERB Parameters

- Filter Frequency Timer: specifies the Timer (T) number which is used by the Filter instruction
- Filter Frequency Time (0.01sec): specifies the rate at which the calculation is performed
- Raw Data (Binary): specifies the V-memory location of the raw unfiltered binary (decimal) value
- Filter Divisor (1-100): this constant used to control the filtering effect. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering.
- Filtered Value (Binary): specifies the V-memory location where the filtered binary (decimal) value will be placed

Parameter		DL06 Range
Filter Frequency Timer	T	T0-377
Filter Frequency Time (0.01 sec)	K	K0-9999
Raw Data (Binary)	V	See DL06 V-memory map - Data Words
Filter Divisor (1-100)	K	K1-100
Filtered Value (Binary)	V	See DL06 V-memory map - Data Words

FILTERB Example

In the following example, the FILTERB instruction is used to filter a binary value that is in V2000. Timer (T1) is set to 0.5 sec, the rate at which the filter calculation will be performed. The filter constant is set to 3.0. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering. The filtered value will be placed in V2100



Filter Over Time - Binary

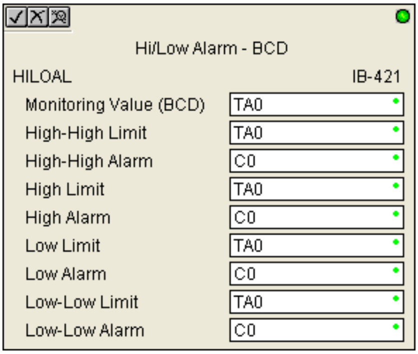
FILTERB	IB-402
Filter Freq Timer	T1
Filter Freq Time (0.01 sec)	K50
Raw Data (Binary)	V2000
Filter Divisor (1-100)	K3
Filtered Value (Binary)	V2100

Hi/Low Alarm - BCD (HILOAL) (IB-421)

DS	Used
HPP	N/A

Hi/Low Alarm - BCD monitors a BCD value V-Memory location and sets four possible alarm states, High-High, High, Low, and Low-Low whenever the IBox has power flow. You enter the alarm thresholds as constant K BCD values (K0-K9999) and/or BCD value V-Memory locations.

You must ensure that threshold limits are valid, that is HH >= H > L >= LL. Note that when the High-High or Low-Low alarm condition is true, that the High and Low alarms will also be set, respectively. This means you may use the same threshold limit and same alarm bit for the High-High and the High alarms in case you only need one “High” alarm. Also note that the boundary conditions are inclusive. That is, if the Low boundary is K50, and the Low-Low boundary is K10, and if the Monitoring Value equals 10, then the Low Alarm AND the Low-Low alarm will both be ON. If there is no power flow to the IBox, then all alarm bits will be turned off regardless of the value of the Monitoring Value parameter.



Hi/Low Alarm - BCD

HILOAL IB-421

Monitoring Value (BCD)	TA0
High-High Limit	TA0
High-High Alarm	C0
High Limit	TA0
High Alarm	C0
Low Limit	TA0
Low Alarm	C0
Low-Low Limit	TA0
Low-Low Alarm	C0

HILOAL Parameters

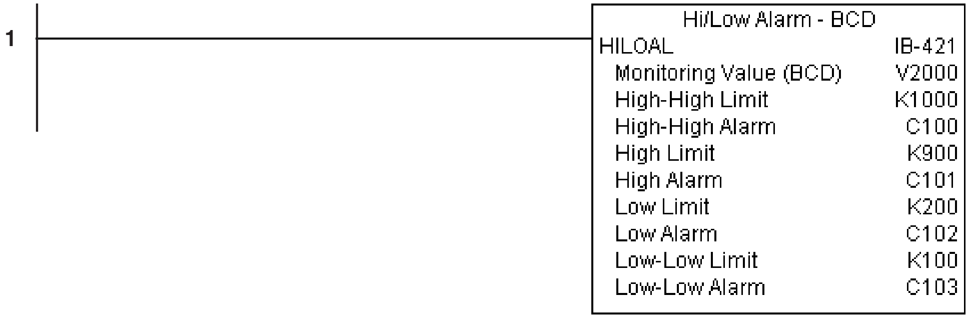
- Monitoring Value (BCD): specifies the V-memory location of the BCD value to be monitored
- High-High Limit: V-memory location or constant specifies the high-high alarm limit
- High-High Alarm: On when the high-high limit is reached
- High Limit: V-memory location or constant specifies the high alarm limit
- High Alarm: On when the high limit is reached
- Low Limit: V-memory location or constant specifies the low alarm limit
- Low Alarm: On when the low limit is reached
- Low-Low Limit: V-memory location or constant specifies the low-low alarm limit
- Low-Low Alarm: On when the low-low limit is reached

Parameter		DL06 Range
Monitoring Value (BCD)	V	See DL06 V-memory map - Data Words
High-High Limit	V, K	K0-9999; or see DL06 V-memory map - Data Words
High-High Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map
High Limit	V, K	K0-9999; or see DL06 V-memory map - Data Words
High Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map
Low Limit	V, K	K0-9999; or see DL06 V-memory map - Data Words
Low Alarm	X, Y, C, GX,GY,B	See DL06 V-memory map
Low-Low Limit	V, K	K0-9999; or see DL06 V-memory map - Data Words
Low-Low Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map

HILOAL Example

In the following example, the HILOAL instruction is used to monitor a BCD value that is in V2000. If the value in V2000 meets/exceeds the high limit of K900, C101 will turn on. If the value continues to increase to meet/exceed the high-high limit, C100 will turn on. Both bits would be on in this case. The high and high-high limits and alarms can be set to the same value if one “high” limit or alarm is desired to be used.

If the value in V2000 meets or falls below the low limit of K200, C102 will turn on. If the value continues to decrease to meet or fall below the low-low limit of K100, C103 will turn on. Both bits would be on in this case. The low and low-low limits and alarms can be set to the same value if one “low” limit or alarm is desired to be used.



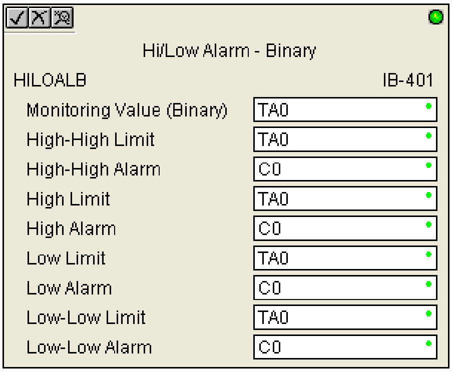
```
            Hi/Low Alarm - BCD
HILOAL                            IB-421
  Monitoring Value (BCD)           V2000
  High-High Limit                  K1000
  High-High Alarm                   C100
  High Limit                        K900
  High Alarm                        C101
  Low Limit                         K200
  Low Alarm                         C102
  Low-Low Limit                     K100
  Low-Low Alarm                     C103
```

Hi/Low Alarm - Binary (HILOALB) (IB-401)

DS	Used
HPP	N/A

Hi/Low Alarm - Binary monitors a binary (decimal) V-Memory location and sets four possible alarm states, High-High, High, Low, and Low-Low whenever the IBox has power flow. You enter the alarm thresholds as constant K decimal values (K0-K65535) and/or binary (decimal) V-Memory locations.

You must ensure that threshold limits are valid, that is HH >= H > L >= LL. Note that when the High-High or Low-Low alarm condition is true, that the High and Low alarms will also be set, respectively. This means you may use the same threshold limit and same alarm bit for the High-High and the High alarms in case you only need one "High" alarm. Also note that the boundary conditions are inclusive. That is, if the Low boundary is K50, and the Low-Low boundary is K10, and if the Monitoring Value equals 10, then the Low Alarm AND the Low-Low alarm will both be ON. If there is no power flow to the IBox, then all alarm bits will be turned off regardless of the value of the Monitoring Value parameter.



Hi/Low Alarm - Binary

HILOALB IB-401

Monitoring Value (Binary)	TA0
High-High Limit	TA0
High-High Alarm	C0
High Limit	TA0
High Alarm	C0
Low Limit	TA0
Low Alarm	C0
Low-Low Limit	TA0
Low-Low Alarm	C0

HILOALB Parameters

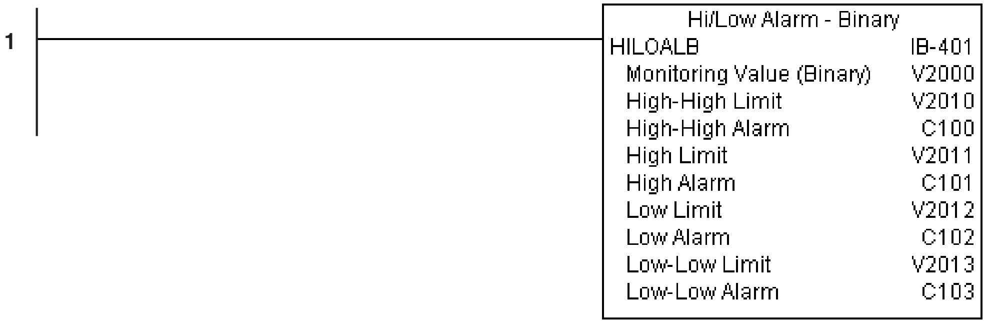
- Monitoring Value (Binary): specifies the V-memory location of the Binary value to be monitored
- High-High Limit: V-memory location or constant specifies the high-high alarm limit
- High-High Alarm: On when the high-high limit is reached
- High Limit: V-memory location or constant specifies the high alarm limit
- High Alarm: On when the high limit is reached
- Low Limit: V-memory location or constant specifies the low alarm limit
- Low Alarm: On when the low limit is reached
- Low-Low Limit: V-memory location or constant specifies the low-low alarm limit
- Low-Low Alarm: On when the low-low limit is reached

Parameter		DL06 Range
Monitoring Value (Binary)	V	See DL06 V-memory map - Data Words
High-High Limit	V, K	K0-65535; or see DL06 V-memory map - Data Words
High-High Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map
High Limit	V, K	K0-65535; or see DL06 V-memory map - Data Words
High Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map
Low Limit	V, K	K0-65535; or see DL06 V-memory map - Data Words
Low Alarm	X, Y, C, GX,GY,B	See DL06 V-memory map
Low-Low Limit	V, K	K0-65535; or see DL06 V-memory map - Data Words
Low-Low Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map

HILOALB Example

In the following example, the HILOALB instruction is used to monitor a binary value that is in V2000. If the value in V2000 meets/exceeds the high limit of the binary value in V2011, C101 will turn on. If the value continues to increase to meet/exceed the high-high limit value in V2010, C100 will turn on. Both bits would be on in this case. The high and high-high limits and alarms can be set to the same V-memory location/value if one “high” limit or alarm is desired to be used.

If the value in V2000 meets or falls below the low limit of the binary value in V2012, C102 will turn on. If the value continues to decrease to meet or fall below the low-low limit in V2013, C103 will turn on. Both bits would be on in this case. The low and low-low limits and alarms can be set to the same V-memory location/value if one “low” limit or alarm is desired to be used.



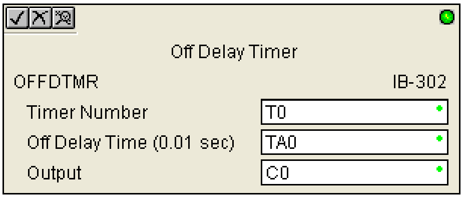
Hi/Low Alarm - Binary

HILOALB	IB-401
Monitoring Value (Binary)	V2000
High-High Limit	V2010
High-High Alarm	C100
High Limit	V2011
High Alarm	C101
Low Limit	V2012
Low Alarm	C102
Low-Low Limit	V2013
Low-Low Alarm	C103

Off Delay Timer (OFFDTMR) (IB-302)

DS	Used
HPP	N/A

Off Delay Timer will delay the "turning off" of the Output parameter by the specified Off Delay Time (in hundredths of a second) based on the power flow into the IBox. Once the IBox receives power, the Output bit will turn on immediately. When the power flow to the IBox turns off, the Output bit WILL REMAIN ON for the specified amount of time (in hundredths of a second).



Timer Number: T0
Off Delay Time (0.01 sec): TA0
Output: C0

Once the Off Delay Time has expired, the output will turn Off. If the power flow to the IBox comes back on BEFORE the Off Delay Time, then the timer is RESET and the Output will remain On - so you must continuously have NO power flow to the IBox for AT LEAST the specified Off Delay Time before the Output will turn Off.

This IBox utilizes a Timer resource (TMRF), which cannot be used anywhere else in your program.

OFFDTMR Parameters

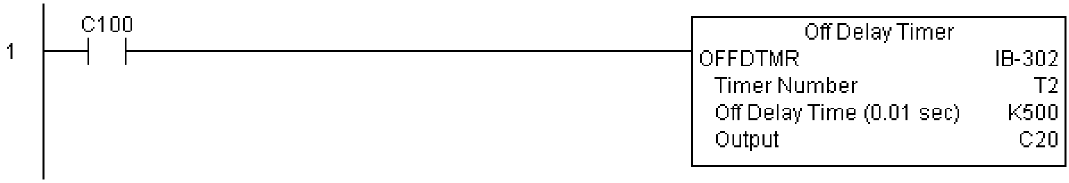
- Timer Number: specifies the Timer(TMRF) number which is used by the OFFDTMR instruction
- Off Delay Time (0.01sec): specifies how long the Output will remain on once power flow to the Ibox is removed
- Output: specifies the output that will be delayed "turning off" by the Off Delay Time.

Parameter		DL06 Range
Timer Number	T	T0-377
Off Delay Time	K,V	K0-9999; See DL06 V-memory map - Data Words
Output	X, Y, C, GX,GY, B	See DL06 V-memory map

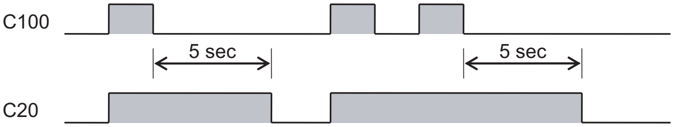
OFFDTMR Example

In the following example, the OFFDTMR instruction is used to delay the “turning off”of output C20. Timer 2 (T2) is set to 5 seconds, the “off-delay” period.

When C100 turns on, C20 turns on and will remain on while C100 is on. When C100 turns off, C20 will remain for the specified Off Delay Time (5s), and then turn off.



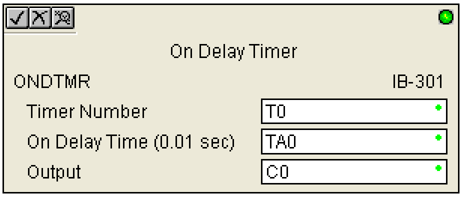
Example timing diagram



On Delay Timer (ONDTMR) (IB-301)

DS	Used
HPP	N/A

On Delay Timer will delay the "turning on" of the Output parameter by the specified amount of time (in hundredths of a second) based on the power flow into the IBox. Once the IBox loses power, the Output is turned off immediately. If the power flow turns off BEFORE the On Delay Time, then the timer is RESET and the Output is never turned on, so you must have continuous power flow to the IBox for at least the specified On Delay Time before the Output turns On.


On Delay Timer

ONDTMR IB-301

Timer Number: T0

On Delay Time (0.01 sec): TA0

Output: C0

This IBox utilizes a Timer resource (TMRF), which cannot be used anywhere else in your program.

ONDTMR Parameters

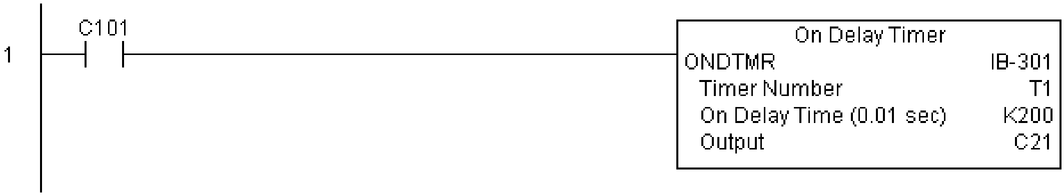
- Timer Number: specifies the Timer(TMRF) number which is used by the ONDTMR instruction
- On Delay Time (0.01sec): specifies how long the Output will remain off once power flow to the Ibox is applied.
- Output: specifies the output that will be delayed "turning on" by the On Delay Time.

Parameter		DL06 Range
Timer Number	T	T0-377
On Delay Time	K,V	K0-9999; See DL06 V-memory map - Data Words
Output	X, Y, C, GX,GY, B	See DL06 V-memory map

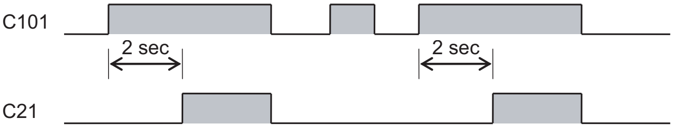
ONDTMR Example

In the following example, the ONDTMR instruction is used to delay the "turning on" of output C21. Timer 1 (T1) is set to 2 seconds, the "on-delay" period.

When C101 turns on, C21 is delayed turning on by 2 seconds. When C101 turns off, C21 turns off immediately.



Example timing diagram



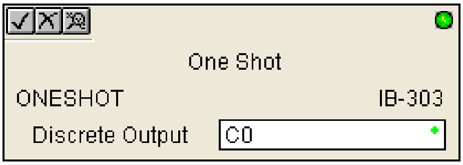
One Shot (ONESHOT) (IB-303)

DS	Used
HPP	N/A

One Shot will turn on the given bit output parameter for one scan on an OFF to ON transition of the power flow into the IBox. This IBox is simply a different name for the PD Coil (Positive Differential).

ONESHOT Parameters

- Discrete Output: specifies the output that will be on for one scan



One Shot

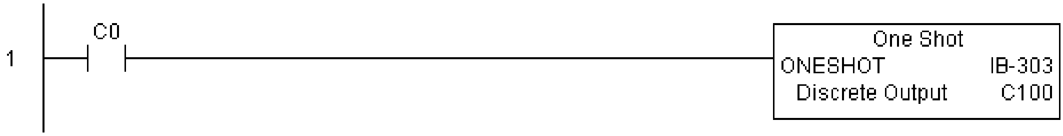
ONESHOT IB-303

Discrete Output C0

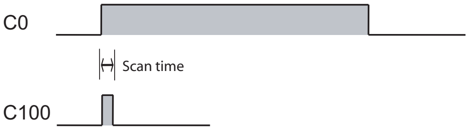
Parameter		DL06 Range
Discrete Output	X, Y, C	See DL06 V-memory map

ONESHOT Example

In the following example, the ONESHOT instruction is used to turn C100 on for one PLC scan after C0 goes from an off to on transition. The input logic must produce an off to on transition to execute the One Shot instruction.



Example timing diagram



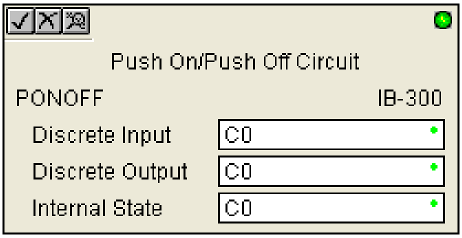
Push On / Push Off Circuit (PONOFF) (IB-300)

DS	Used
HPP	N/A

Push On/Push Off Circuit toggles an output state whenever its input power flow transitions from off to on. Requires an extra bit parameter for scan-to-scan state information. This extra bit must NOT be used anywhere else in the program. This is also known as a “flip-flop circuit”. The PONOFF IBox cannot have any input logic.

PONOFF Parameters

- Discrete Input: specifies the input that will toggle the specified output
- Discrete Output: specifies the output that will be “turned on/off” or toggled
- Internal State: specifies a work bit that is used by the instruction



Push On/Push Off Circuit

PONOFF IB-300

Discrete Input: C0

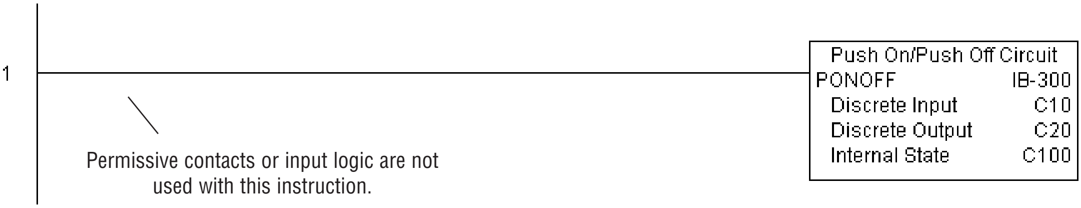
Discrete Output: C0

Internal State: C0

Parameter		DL06 Range
Discrete Input	X,Y,C,S,T,CT,GX,GY,SP,B,PB	See DL06 V-memory map
Discrete Output	X,Y,C,GX,GY,B	See DL06 V-memory map
Internal State	X, Y, C	See DL06 V-memory map

PONOFF Example

In the following example, the PONOFF instruction is used to control the on and off states of the output C20 with a single input C10. When C10 is pressed once, C20 turns on. When C10 is pressed again, C20 turns off. C100 is an internal bit used by the instruction.



Permissive contacts or input logic are not used with this instruction.

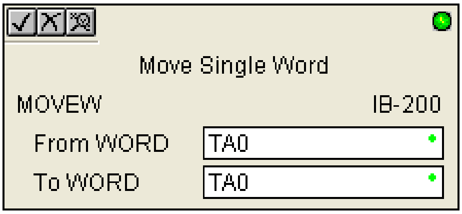
Move Single Word (MOVEW) (IB-200)

DS	Used
HPP	N/A

Move Single Word moves (copies) a word to a memory location directly or indirectly via a pointer, either as a HEX constant, from a memory location, or indirectly through a pointer

MOVEW Parameters

- From WORD: specifies the word that will be moved to another location
- To WORD: specifies the location where the "From WORD" will be move to



Move Single Word

MOVEW IB-200

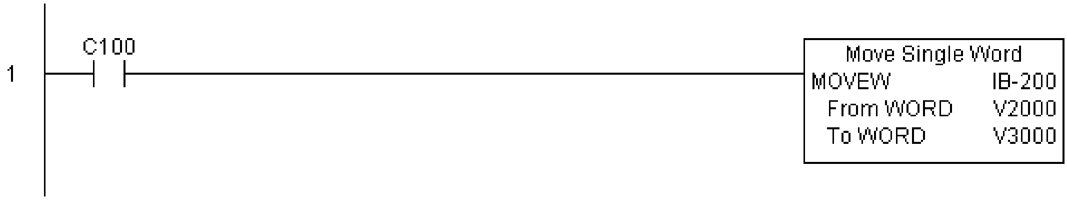
From WORD: TA0

To WORD: TA0

Parameter		DL06 Range
From WORD	V,P,K	K0-FFFF; See DL06 V-memory map - Data Words
To WORD	V,P	See DL06 V-memory map - Data Words

MOVEW Example

In the following example, the MOVEW instruction is used to move 16-bits of data from V2000 to V3000 when C100 turns on.



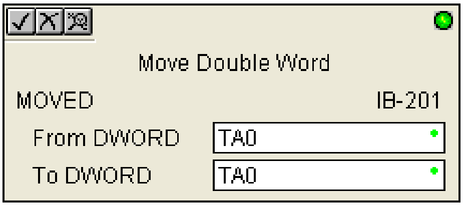
Move Double Word (MOVED) (IB-201)

DS	Used
HPP	N/A

Move Double Word moves (copies) a double word to two consecutive memory locations directly or indirectly via a pointer, either as a double HEX constant, from a double memory location, or indirectly through a pointer to a double memory location.

MOVED Parameters

- From DWORD: specifies the double word that will be moved to another location
- To DWORD: specifies the location where the "From DWORD" will be move to



Move Double Word

MOVED IB-201

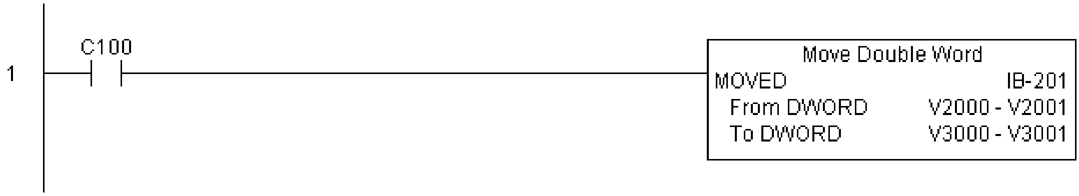
From DWORD: TA0

To DWORD: TA0

Parameter		DL06 Range
From DWORD	V,P,K	K0-FFFFFFFF; See DL06 V-memory map - Data Words
To DWORD	V,P	See DL06 V-memory map - Data Words

MOVED Example

In the following example, the MOVED instruction is used to move 32-bits of data from V2000 and V2001 to V3000 and V3001 when C100 turns on.

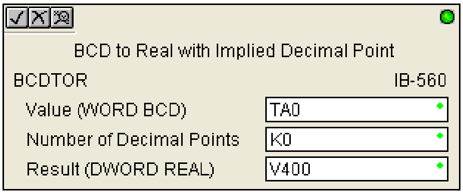


BCD to Real with Implied Decimal Point (BCDTOR) (IB-560)

DS	Used
HPP	N/A

BCD to Real with Implied Decimal Point converts the given 4 digit WORD BCD value to a Real number, with the implied number of decimal points (K0-K4).

For example, BCDTOR K1234 with an implied number of decimal points equal to K1, would yield R123.4



BCD to Real with Implied Decimal Point

BCDTOR IB-560

Value (WORD BCD): TA0

Number of Decimal Points: K0

Result (DWORD REAL): V400

BCDTOR Parameters

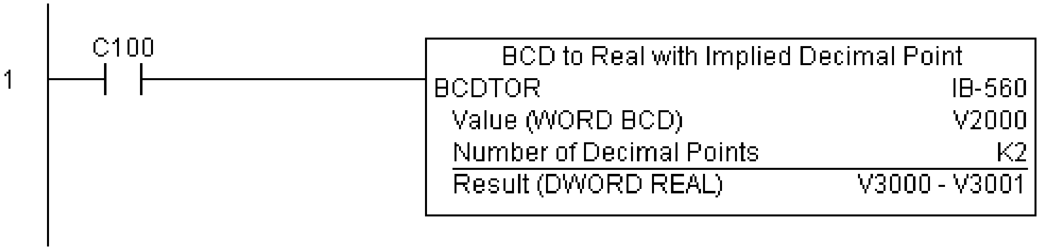
- Value (WORD BCD): specifies the word or constant that will be converted to a Real number
- Number of Decimal Points: specifies the number of implied decimal points in the Result DWORD
- Result (DWORD REAL): specifies the location where the Real number will be placed

Parameter		DL06 Range
Value (WORD BCD)	V,P,K	K0-9999; See DL06 V-memory map - Data Words
Number of Decimal Points	K	K0-4
Result (DWORD REAL)	V	See DL06 V-memory map - Data Words

BCDTOR Example

In the following example, the BCDTOR instruction is used to convert the 16-bit data in V2000 from a 4-digit BCD data format to a 32-bit REAL (floating point) data format and stored into V3000 and V3001 when C100 turns on.

K2 in the Number of Decimal Points implies the data will have two digits to the right of the decimal point.



BCD to Real with Implied Decimal Point

BCDTOR IB-560

Value (WORD BCD) V2000

Number of Decimal Points K2

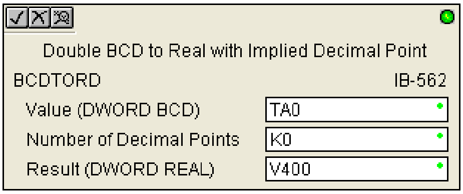
Result (DWORD REAL) V3000 - V3001

Double BCD to Real with Implied Decimal Point (BCDTORD) (IB-562)

DS	Used
HPP	N/A

Double BCD to Real with Implied Decimal Point converts the given 8 digit DWORD BCD value to a Real number, given an implied number of decimal points (K0-K8).

For example, BCDTORD K12345678 with an implied number of decimal points equal to K5, would yield R123.45678



Double BCD to Real with Implied Decimal Point

BCDTORD IB-562

Value (DWORD BCD) TA0

Number of Decimal Points K0

Result (DWORD REAL) V400

BCDTORD Parameters

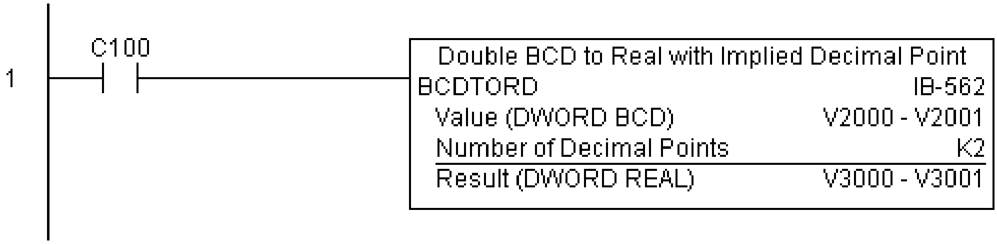
- Value (DWORD BCD): specifies the Dword or constant that will be converted to a Real number
- Number of Decimal Points: specifies the number of implied decimal points in the Result DWORD
- Result (DWORD REAL): specifies the location where the Real number will be placed

BCDTORD Example

Parameter		DL06 Range
Value (DWORD BCD)	V,P,K	K0-99999999; See DL06 V-memory map - Data Words
Number of Decimal Points	K	K0-8
Result (DWORD REAL)	V	See DL06 V-memory map - Data Words

In the following example, the BCDTORD instruction is used to convert the 32-bit data in V2000 from an 8-digit BCD data format to a 32-bit REAL (floating point) data format and stored into V3000 and V3001 when C100 turns on.

K2 in the Number of Decimal Points implies the data will have two digits to the right of the decimal point.



Double BCD to Real with Implied Decimal Point

BCDTORD IB-562

Value (DWORD BCD) V2000 - V2001

Number of Decimal Points K2

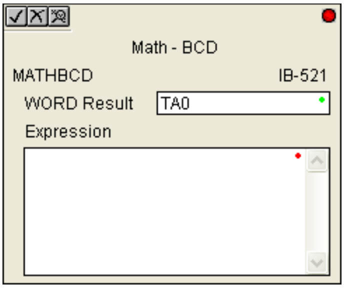
Result (DWORD REAL) V3000 - V3001

Math - BCD (MATHBCD) (IB-521)

DS	Used
HPP	N/A

Math - BCD Format lets you enter complex mathematical expressions like you would in Visual Basic, Excel, or C++ to do complex calculations, nesting parentheses up to 4 levels deep. In addition to + - * /, you can do Modulo (% aka Remainder), Bit-wise And (&) Or (|) Xor (^), and some BCD functions - Convert to BCD (BCD), Convert to Binary (BIN), BCD Complement (BCDCPL), Convert from Gray Code (GRAY), Invert Bits (INV), and BCD/HEX to Seven Segment Display (SEG).

Example: ((V2000 + V2001) / (V2003 - K100)) * GRAY(V3000 & K001F)



Math - BCD

MATHBCD IB-521

WORD Result TA0

Expression

Every V-memory reference MUST be to a single word BCD formatted value. Intermediate results can go up to 32 bit values, but as long as the final result fits in a 16 bit BCD word, the calculation is valid. Typical example of this is scaling using multiply then divide, (V2000 * K1000) / K4095. The multiply term most likely will exceed 9999 but fits within 32 bits. The divide operation will divide 4095 into the 32-bit accumulator, yielding a result that will always fit in 16 bits.

You can reference binary V-memory values by using the BCD conversion function on a V-Memory location but NOT an expression. That is BCD(V2000) is okay and will convert V2000 from Binary to BCD, but BCD(V2000 + V3000) will add V2000 as BCD, to V3000 as BCD, then interpret the result as Binary and convert it to BCD - NOT GOOD.

Also, the final result is a 16 bit BCD number and so you could do BIN around the entire operation to store the result as Binary.

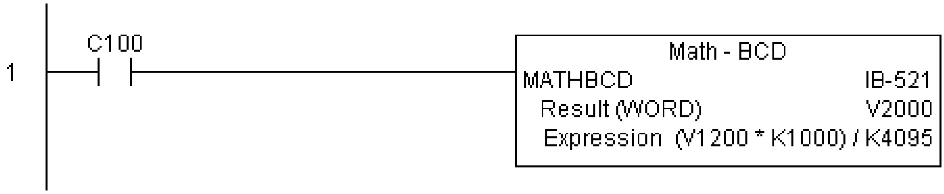
MATHBCD Parameters

- WORD Result: specifies the location where the BCD result of the mathematical expression will be placed (result must fit into 16 bit single V-memory location)
- Expression: specifies the mathematical expression to be executed and the result is stored in specified WORD Result. Each V-memory location used in the expression must be in BCD format.

Parameter		DL06 Range
WORD Result	V	See DL06 V-memory map - Data Words
Expression		Text

MATHBCD Example

In the following example, the MATHBCD instruction is used to calculate the math expression which multiplies the BCD value in V1200 by 1000 then divides by 4095 and loads the resulting value in V2000 when C100 turns on.

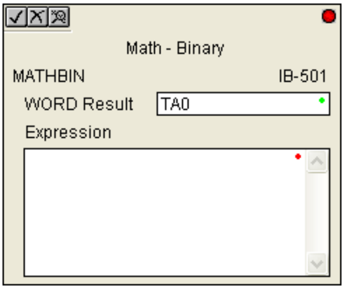


Math - Binary (MATHBIN) (IB-501)

DS	Used
HPP	N/A

Math - Binary Format lets you enter complex mathematical expressions like you would in Visual Basic, Excel, or C++ to do complex calculations, nesting parentheses up to 4 levels deep. In addition to + - * /, you can do Modulo (% aka Remainder), Shift Right (>>) and Shift Left (<<), Bit-wise And (&) Or (|) Xor (^), and some binary functions - Convert to BCD (BCD), Convert to Binary (BIN), Decode Bits (DECO), Encode Bits (ENCO), Invert Bits (INV), HEX to Seven Segment Display (SEG), and Sum Bits (SUM).

Example: ((V2000 + V2001) / (V2003 - K10)) * SUM(V3000 & K001F)



Math - Binary

MATHBIN IB-501

WORD Result: TA0

Expression

Every V-memory reference MUST be to a single word binary formatted value. Intermediate results can go up to 32 bit values, but as long as the final result fits in a 16 bit binary word, the calculation is valid. Typical example of this is scaling using multiply then divide, (V2000 * K1000) / K4095. The multiply term most likely will exceed 65535 but fits within 32 bits. The divide operation will divide 4095 into the 32-bit accumulator, yielding a result that will always fit in 16 bits.

You can reference BCD V-Memory values by using the BIN conversion function on a V-memory location but NOT an expression. That is, BIN(V2000) is okay and will convert V2000 from BCD to Binary, but BIN(V2000 + V3000) will add V2000 as Binary, to V3000 as Binary, then interpret the result as BCD and convert it to Binary - NOT GOOD.

Also, the final result is a 16 bit binary number and so you could do BCD around the entire operation to store the result as BCD.

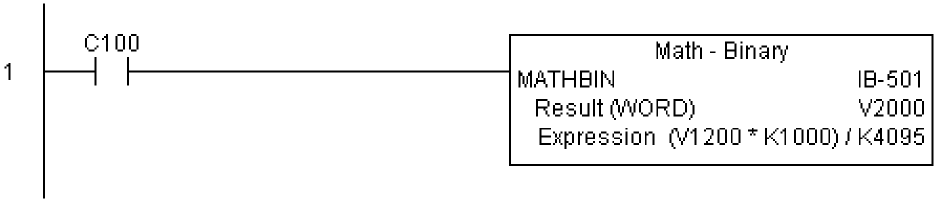
MATHBIN Parameters

- WORD Result: specifies the location where the binary result of the mathematical expression will be placed (result must fit into 16 bit single V-memory location)
- Expression: specifies the mathematical expression to be executed and the result is stored in specified WORD Result. Each V-memory location used in the expression must be in binary format.

Parameter		DL06 Range
WORD Result	V	See DL06 V-memory map - Data Words
Expression		Text

MATHBIN Example

In the following example, the MATHBIN instruction is used to calculate the math expression which multiplies the Binary value in V1200 by 1000 then divides by 4095 and loads the resulting value in V2000 when C100 turns on.



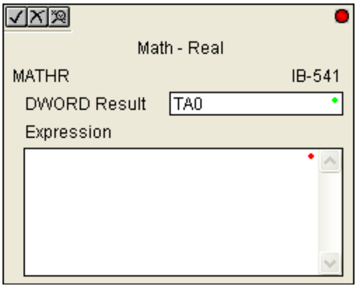
Math - Real (MATHR) (IB-541)

DS	Used
HPP	N/A

Math - Real Format lets you enter complex mathematical expressions like you would in Visual Basic, Excel, or C++ to do complex calculations, nesting parentheses up to 4 levels deep. In addition to + - * /, you can do Bit-wise And (&) Or (|) Xor (^), and many Real functions - Arc Cosine (ACOSR), Arc Sine (ASINR), Arc Tangent (ATANR), Cosine (COSR), Convert Radians to Degrees (DEGR), Invert Bits (INV), Convert Degrees to Radians (RADR), HEX to Seven Segment Display (SEG), Sine (SINR), Square Root (SQRTR), Tangent (TANR).

Example: ((V2000 + V2002) / (V2004 - R2.5)) * SINR(RADR(V3000 / R10.0))

Every V-memory reference MUST be able to fit into a double word Real formatted value.


Math - Real

MATHR IB-541

DWORD Result TA0

Expression

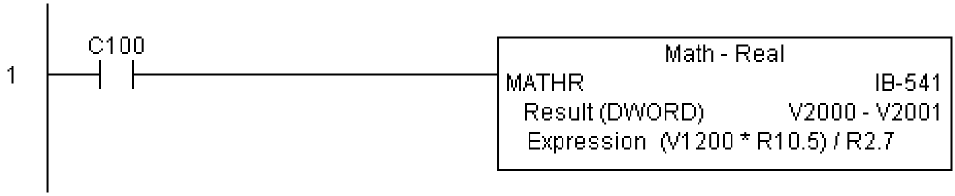
MATHR Parameters

- DWORD Result: specifies the location where the Real result of the mathematical expression will be placed (result must fit into a double word Real formatted location)
- Expression: specifies the mathematical expression to be executed and the result is stored in specified DWORD Result location. Each V-memory location used in the expression must be in Real format.

Parameter	DL06 Range
DWORD Result V	See DL06 V-memory map - Data Words
Expression	Text

MATHR Example

In the following example, the MATHR instruction is used to calculate the math expression which multiplies the REAL (floating point) value in V1200 by 10.5 then divides by 2.7 and loads the resulting 32-bit value in V2000 and V2001 when C100 turns on.



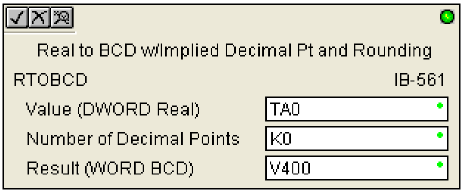
Real to BCD with Implied Decimal Point and Rounding (RTOBCD) (IB-561)

DS	Used
HPP	N/A

Real to BCD with Implied Decimal Point and Rounding converts the absolute value of the given Real number to a 4 digit BCD number, compensating for an implied number of decimal points (K0-K4) and performs rounding.

For example, RTOBCD R56.74 with an implied number of decimal points equal to K1, would yield 567 BCD. If the implied number of decimal points was 0, then the function would yield 57 BCD (note that it rounded up).

If the Real number is negative, the Result will equal its positive, absolute value.



Real to BCD w/Implied Decimal Pt and Rounding

RTOBCD IB-561

Value (DWORD Real): TA0

Number of Decimal Points: K0

Result (WORD BCD): V400

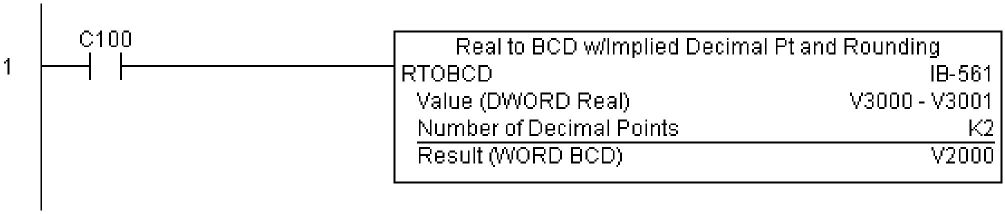
RTOBCD Parameters

- Value (DWORD Real): specifies the Real Dword location or number that will be converted and rounded to a BCD number with decimal points
- Number of Decimal Points: specifies the number of implied decimal points in the Result WORD
- Result (WORD BCD): specifies the location where the rounded/implied decimal points BCD value will be placed

Parameter		DL06 Range
Value (DWORD Real)	V,P,R	R ; See DL06 V-memory map - Data Words
Number of Decimal Points	K	K0-4
Result (WORD BCD)	V	See DL06 V-memory map - Data Words

RTOBCD Example

In the following example, the RTOBCD instruction is used to convert the 32-bit REAL (floating point) data format in V3000 and V3001 to the 4-digit BCD data format and stored in V2000 when C100 turns on.



Real to BCD w/Implied Decimal Pt and Rounding

RTOBCD IB-561

Value (DWORD Real) V3000 - V3001

Number of Decimal Points K2

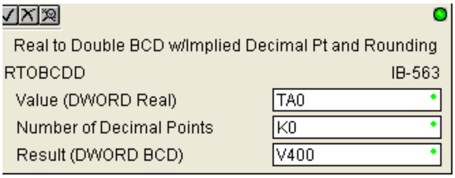
Result (WORD BCD) V2000

K2 in the Number of Decimal Points implies the data will have two implied decimal points.

Real to Double BCD with Implied Decimal Point and Rounding (RTOBCDD) (IB-563)

DS	Used
HPP	N/A

Real to Double BCD with Implied Decimal Point and Rounding converts the absolute value of the given Real number to an 8 digit DWORD BCD number, compensating for an implied number of decimal points (K0-K8) and performs rounding.



Real to Double BCD w/Implied Decimal Pt and Rounding
RTOBCDD IB-563
Value (DWORD Real) TA0
Number of Decimal Points K0
Result (DWORD BCD) V400

For example, RTOBCDD R38156.74 with an implied number of decimal points equal to K1, would yield 381567 BCD. If the implied number of decimal points was 0, then the function would yield 38157 BCD (note that it rounded up).

If the Real number is negative, the Result will equal its positive, absolute value.

RTOBCDD Parameters

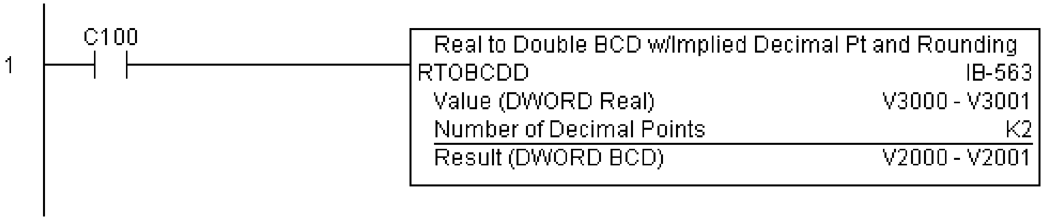
- Value (DWORD Real): specifies the Dword Real number that will be converted and rounded to a BCD number with decimal points
- Number of Decimal Points: specifies the number of implied decimal points in the Result DWORD
- Result (DWORD BCD): specifies the location where the rounded/implied decimal points DWORD BCD value will be placed

Parameter		DL06 Range
Value (DWORD Real)	V,P,R	R ; See DL06 V-memory map - Data Words
Number of Decimal Points	K	K0-8
Result (DWORD BCD)	V	See DL06 V-memory map - Data Words

RTOBCDD Example

In the following example, the RTOBCDD instruction is used to convert the 32-bit REAL (floating point) data format in V3000 and V3001 to the 8-digit BCD data format and stored in V2000 and V2001 when C100 turns on.

K2 in the Number of Decimal Points implies the data will have two implied decimal points.



Real to Double BCD w/Implied Decimal Pt and Rounding
RTOBCDD IB-563
Value (DWORD Real) V3000 - V3001
Number of Decimal Points K2
Result (DWORD BCD) V2000 - V2001

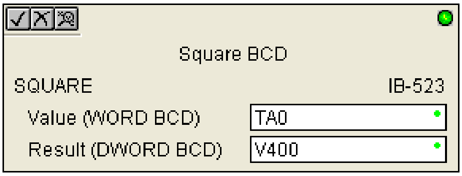
Square BCD (SQUARE) (IB-523)

DS	Used
HPP	N/A

Square BCD squares the given 4-digit WORD BCD number and writes it in as an 8-digit DWORD BCD result.

SQUARE Parameters

- Value (WORD BCD): specifies the BCD Word or constant that will be squared
- Result (DWORD BCD): specifies the location where the squared DWORD BCD value will be placed



Square BCD

SQUARE IB-523

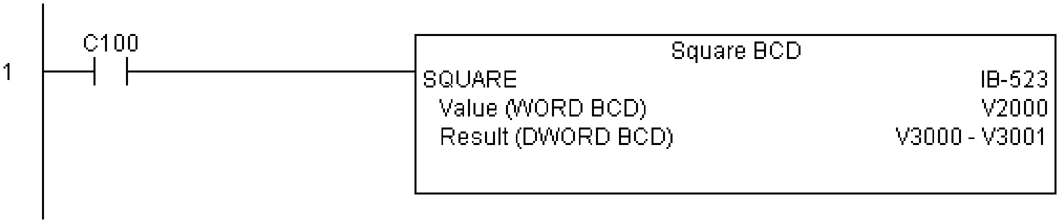
Value (WORD BCD) TA0

Result (DWORD BCD) V400

Parameter		DL06 Range
Value (WORD BCD)	V,P,K	K0-9999 ; See DL06 V-memory map - Data Words
Result (DWORD BCD)	V	See DL06 V-memory map - Data Words

SQUARE Example

In the following example, the SQUARE instruction is used to square the 4-digit BCD value in V2000 and store the 8-digit double word BCD result in V3000 and V3001 when C100 turns on.



Square BCD

SQUARE IB-523

Value (WORD BCD) V2000

Result (DWORD BCD) V3000 - V3001

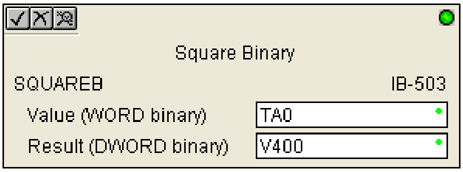
Square Binary (SQUAREB) (IB-503)

DS	Used
HPP	N/A

Square Binary squares the given 16-bit WORD Binary number and writes it as a 32-bit DWORD Binary result.

SQUAREB Parameters

- Value (WORD Binary): specifies the binary Word or constant that will be squared
- Result (DWORD Binary): specifies the location where the squared DWORD binary value will be placed



Square Binary

SQUAREB IB-503

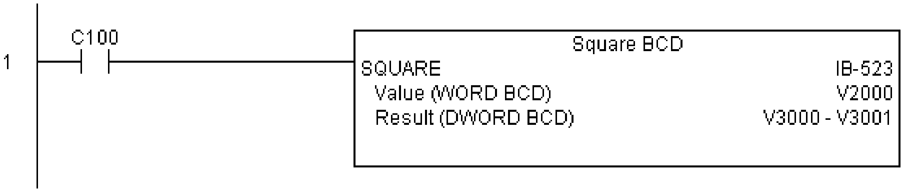
Value (WORD binary) TA0

Result (DWORD binary) V400

Parameter		DL06 Range
Value (WORD Binary)	V,P,K	K0-65535; See DL06 V-memory map - Data Words
Result (DWORD Binary)	V	See DL06 V-memory map - Data Words

SQUAREB Example

In the following example, the SQUAREB instruction is used to square the single word Binary value in V2000 and store the 8-digit double word Binary result in V3000 and V3001 when C100 turns on.



Square BCD

SQUARE IB-523

Value (WORD BCD) V2000

Result (DWORD BCD) V3000 - V3001

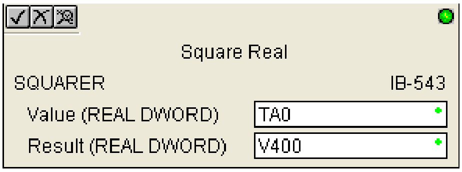
Square Real (SQUARER) (IB-543)

DS	Used
HPP	N/A

Square Real squares the given REAL DWORD number and writes it to a REAL DWORD result.

SQUARER Parameters

- Value (REAL DWORD): specifies the Real DWORD location or number that will be squared
- Result (REAL DWORD): specifies the location where the squared Real DWORD value will be placed

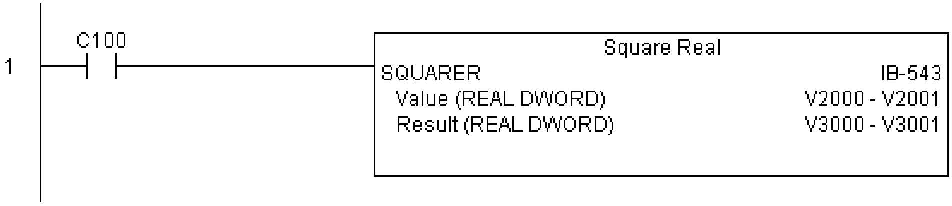


Square Real
SQUARER IB-543
Value (REAL DWORD) TA0
Result (REAL DWORD) V400

Parameter		DL06 Range
Value (REAL DWORD)	V,P,R	R ; See DL06 V-memory map - Data Words
Result (REAL DWORD)	V	See DL06 V-memory map - Data Words

SQUARER Example

In the following example, the SQUARER instruction is used to square the 32-bit floating point REAL value in V2000 and V2001 and store the REAL value result in V3000 and V3001 when C100 turns on.

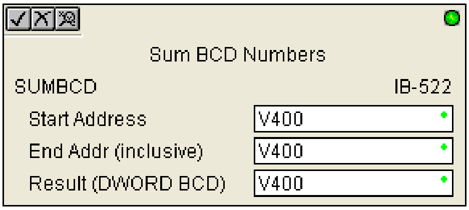


Square Real
SQUARER IB-543
Value (REAL DWORD) V2000 - V2001
Result (REAL DWORD) V3000 - V3001

Sum BCD Numbers (SUMBCD) (IB-522)

DS	Used
HPP	N/A

Sum BCD Numbers sums up a list of consecutive 4-digit WORD BCD numbers into an 8-digit DWORD BCD result.



Sum BCD Numbers

SUMBCD IB-522

Start Address: V400

End Addr (inclusive): V400

Result (DWORD BCD): V400

You specify the group's starting and ending V-memory addresses (inclusive). When enabled, this instruction will add up all the numbers in the group (so you may want to place a differential contact driving the enable).

SUMBCD could be used as the first part of calculating an average.

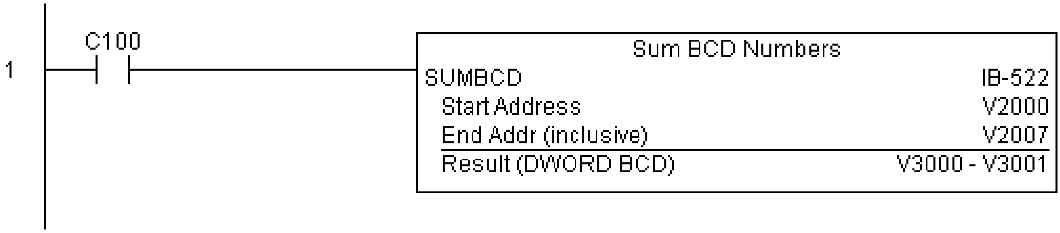
SUMBCD Parameters

- Start Address: specifies the starting address of a block of V-memory location values to be added together (BCD)
- End Addr (inclusive): specifies the ending address of a block of V-memory location values to be added together (BCD)
- Result (DWORD BCD): specifies the location where the sum of the block of V-memory BCD values will be placed

Parameter		DL06 Range
Start Address	V	See DL06 V-memory map - Data Words
End Address (inclusive)	V	See DL06 V-memory map - Data Words
Result (DWORD BCD)	V	See DL06 V-memory map - Data Words

SUMBCD Example

In the following example, the SUMBCD instruction is used to total the sum of all BCD values in words V2000 thru V2007 and store the resulting 8-digit double word BCD value in V3000 and V3001 when C100 turns on.



Sum BCD Numbers

SUMBCD IB-522

Start Address V2000

End Addr (inclusive) V2007

Result (DWORD BCD) V3000 - V3001

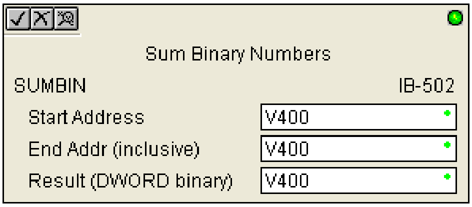
Sum Binary Numbers (SUMBIN) (IB-502)

DS	Used
HPP	N/A

Sum Binary Numbers sums up a list of consecutive 16-bit WORD Binary numbers into a 32-bit DWORD binary result.

You specify the group's starting and ending V-memory addresses (inclusive). When enabled, this instruction will add up all the numbers in the group (so you may want to place a differential contact driving the enable).

SUMBIN could be used as the first part of calculating an average.


Sum Binary Numbers
SUMBIN IB-502
Start Address V400
End Addr (inclusive) V400
Result (DWORD binary) V400

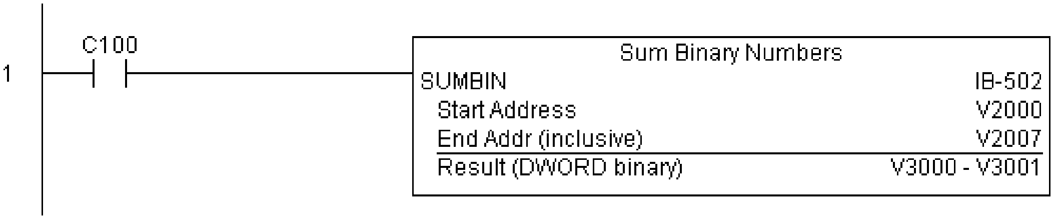
SUMBIN Parameters

- Start Address: specifies the starting address of a block of V-memory location values to be added together (Binary)
- End Addr (inclusive): specifies the ending address of a block of V-memory location values to be added together (Binary)
- Result (DWORD Binary): specifies the location where the sum of the block of V-memory binary values will be placed

Parameter		DL06 Range
Start Address	V	See DL06 V-memory map - Data Words
End Address (inclusive)	V	See DL06 V-memory map - Data Words
Result (DWORD Binary)	V	See DL06 V-memory map - Data Words

SUMBIN Example

In the following example, the SUMBIN instruction is used to total the sum of all Binary values in words V2000 thru V2007 and store the resulting 8-digit double word Binary value in V3000 and V3001 when C100 turns on.


1
C100
Sum Binary Numbers
SUMBIN IB-502
Start Address V2000
End Addr (inclusive) V2007
Result (DWORD binary) V3000 - V3001

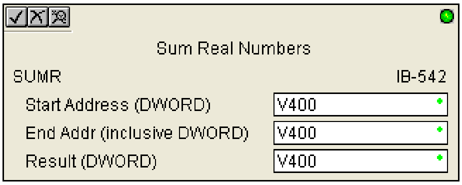
Sum Real Numbers (SUMR) (IB-542)

DS	Used
HPP	N/A

Sum Real Numbers sums up a list of consecutive REAL DWORD numbers into a REAL DWORD result.

You specify the group's starting and ending V- memory addresses (inclusive).

Remember that Real numbers are DWORDs and occupy 2 words of V-Memory each, so the number of Real values summed up is equal to half the number of memory locations. Note that the End Address can be EITHER word of the 2 word ending address, for example, if you wanted to add the 4 Real numbers stored in V2000 thru V2007 (V2000, V2002, V2004, and V2006), you can specify V2006 OR V2007 for the ending address and you will get the same result.


Sum Real Numbers

SUMR IB-542

Start Address (DWORD) V400

End Addr (inclusive DWORD) V400

Result (DWORD) V400

When enabled, this instruction will add up all the numbers in the group (so you may want to place a differential contact driving the enable).

SUMR could be used as the first part of calculating an average.

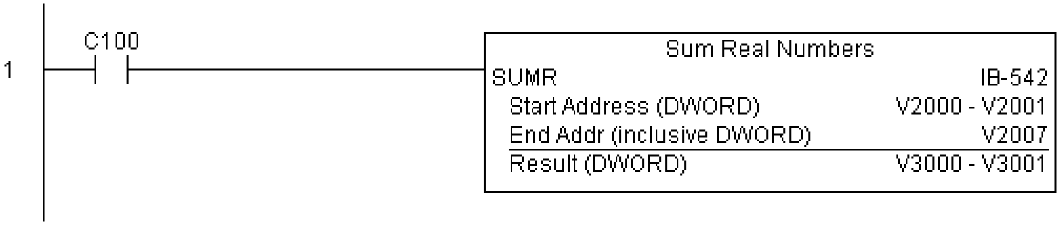
SUMR Parameters

- Start Address (DWORD): specifies the starting address of a block of V-memory location values to be added together (Real)
- End Addr (inclusive) (DWORD): specifies the ending address of a block of V-memory location values to be added together (Real)
- Result (DWORD): specifies the location where the sum of the block of V-memory Real values will be placed

Parameter		DL06 Range
Start Address (inclusive DWORD)	V	See DL06 V-memory map - Data Words
End Address (inclusive DWORD)	V	See DL06 V-memory map - Data Words
Result (DWORD)	V	See DL06 V-memory map - Data Words

SUMR Example

In the following example, the SUMR instruction is used to total the sum of all floating point REAL number values in words V2000 thru V2007 and store the resulting 32-bit floating point REAL number value in V3000 and V3001 when C100 turns on.



```
1   C100
  ──┤ ├──────────────  Sum Real Numbers
                       SUMR                                  IB-542
                         Start Address (DWORD)        V2000 - V2001
                         End Addr (inclusive DWORD)           V2007
                         Result (DWORD)               V3000 - V3001
```

ECOM100 Configuration (ECOM100) (IB-710)

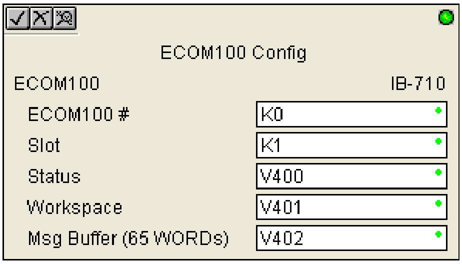
DS	Used
HPP	N/A

ECOM100 Configuration defines the parameters other ECOM100 IBoxes will use when working with this specific ECOM100 module. Each ECOM100 module that will be used with IBox instructions will require a unique ECOM1000 Configuration instruction. The addresses used become workspaces for the IBox instruction to use.

The addresses used in this instruction must not be used elsewhere in the program.

The instructions must be placed at the top of ladder, without a contact. The instruction will inherently run only once, on the first scan.

IBoxes ECEMAIL, ECRX, ECIPSUP and others require an ECOM100 Configuration before they will operate properly.



ECOM100 Config

ECOM100 IB-710

ECOM100 # K0

Slot K1

Status V400

Workspace V401

Msg Buffer (65 WORDs) V402

5

In order for MOST ECOM100 IBoxes to function, DIP switch 7 on the ECOM100 circuit board must be ON DIP switch 7 can remainOFF if ECOM100 Network Read and Write IBoxes (ECRX, ECWX) are the only IBoxes that will be used.

ECOM100 Configuration Parameters

- ECOM100#: specify a logical number to be associated with this particular ECOM100 module. All other ECxxxx IBoxes that need to reference this ECOM1000 module must reference this logical number
- Slot: specifies the option slot the module occupies
- Status: specifies a V-memory location that will be used by the instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Msg Buffer: specifies the starting address of a 65 word buffer that will be used by the module for configuration

Parameter		DL06 Range
ECOM100#	K	K0-255
Slot	K	K1-4
Status	V	See DL06 V-memory map - Data Words
Workspace	V	See DL06 V-memory map - Data Words
Msg Buffer (65 words used)	V	See DL06 V-memory map - Data Words

ECOM100 Example

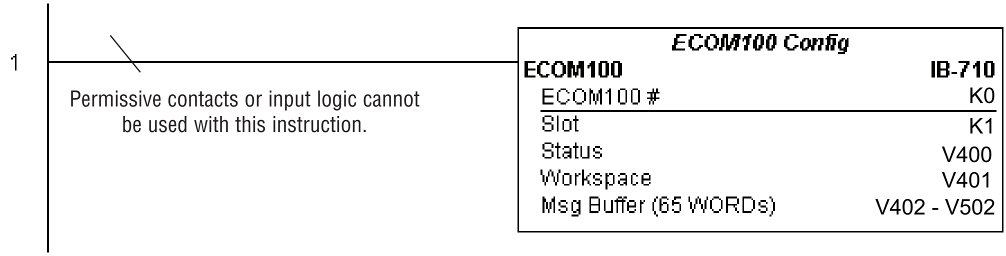
The ECOM100 Config IBox coordinates all of the interaction with other ECOM100 based IBoxes (ECxxxx). You must have an ECOM100 Config IBox for each ECOM100 module in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines ECOM100# K0 to be in slot 3. Any ECOM100 IBoxes that need to reference this specific module (such as ECEMAIL, ECRX, ...) would enter K0 for their ECOM100# parameter.

The Status register is for reporting any completion or error information to other ECOM100 IBoxes. This V-Memory register must not be used anywhere else in the entire program.

The Workspace register is used to maintain state information about the ECOM100, along with proper sharing and interlocking with the other ECOM100 IBoxes in the program. This V-Memory register must not be used anywhere else in the entire program.

The Message Buffer of 65 words (130 bytes) is a common pool of memory that is used by other ECOM100 IBoxes (such as ECEMAIL). This way, you can have a bunch of ECEMAIL IBoxes, but only need 1 common buffer for generating and sending each EMail. These V-Memory registers must not be used anywhere else in your entire program.



Permissive contacts or input logic cannot be used with this instruction.

ECOM100 Config

ECOM100 IB-710

ECOM100# K0

Slot K1

Status V400

Workspace V401

Msg Buffer (65 WORDs) V402 - V502

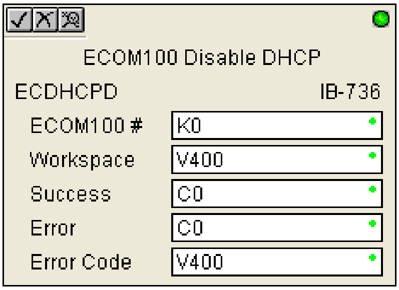
ECOM100 Disable DHCP (ECDHCPD) (IB-736)

DS	Used
HPP	N/A

ECOM100 Disable DHCP will setup the ECOM100 to use its internal TCP/IP settings on a leading edge transition to the IBox. To configure the ECOM100's TCP/IP settings manually, use the NetEdit3 utility, or you can do it programmatically from your PLC program using the ECOM100 IP Setup (ECIPSUP), or the individual ECOM100 IBoxes: ECOM Write IP Address (ECWRIP), ECOM Write Gateway Address (ECWRGWA), and ECOM100 Write Subnet Mask (ECWRSNM).

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



ECOM100 Disable DHCP

ECDHCPD IB-736

ECOM100 #	K0
Workspace	V400
Success	C0
Error	C0
Error Code	V400

The "Disable DHCP" setting is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE, on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

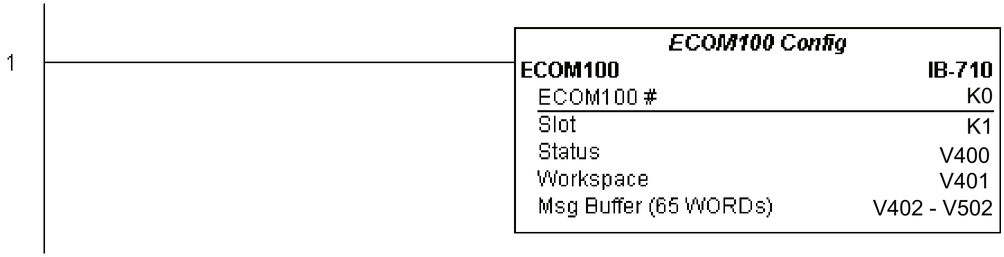
ECDHCPD Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words

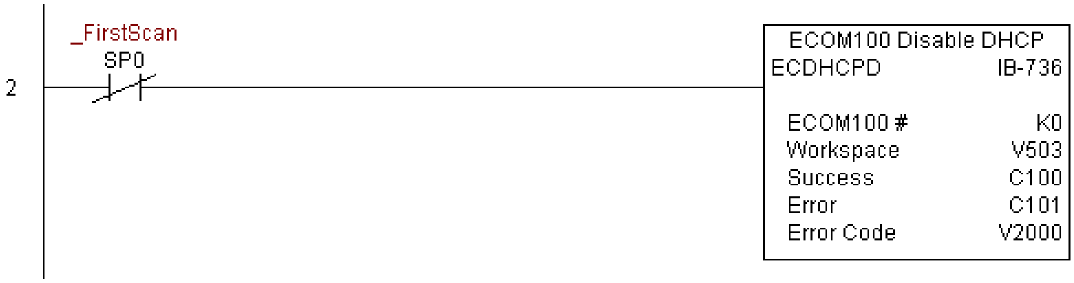
ECDHCPD Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



ECOM100 Config
ECOM100 IB-710
ECOM100 # K0
Slot K1
Status V400
Workspace V401
Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, disable DHCP in the ECOM100. DHCP is the same protocol used by PCs for using a DHCP Server to automatically assign the ECOM100's IP Address, Gateway Address, and Subnet Mask. Typically disabling DHCP is done by assigning a hard-coded IP Address either in NetEdit or using one of the ECOM100 IP Setup IBoxes, but this IBox allows you to disable DHCP in the ECOM100 using your ladder program. The ECDHCPD is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to disable DHCP will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON. If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 Disable DHCP
ECDHCPD IB-736
ECOM100 # K0
Workspace V503
Success C100
Error C101
Error Code V2000

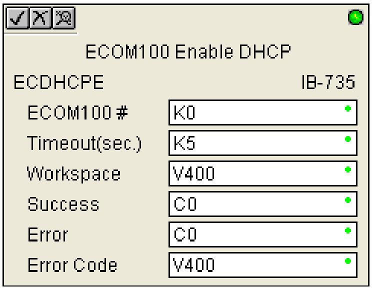
ECOM100 Enable DHCP (ECDHCPE) (IB-735)

DS	Used
HPP	N/A

ECOM100 Enable DHCP will tell the ECOM100 to obtain its TCP/IP setup from a DHCP Server on a leading edge transition to the IBox.

The IBox will be successful once the ECOM100 has received its TCP/IP settings from the DHCP server. Since it is possible for the DHCP server to be unavailable, a Timeout parameter is provided so that the IBox can complete, but with an Error (Error Code = 1004 decimal).

See also the ECOM100 IP Setup (ECIPSUP) IBox 717 to directly setup ALL of the TCP/IP parameters in a single instruction - IP Address, Subnet Mask, and Gateway Address.



ECOM100 Enable DHCP

ECDHCPE IB-735

ECOM100 #	K0
Timeout(sec.)	K5
Workspace	V400
Success	C0
Error	C0
Error Code	V400

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

The “Enable DHCP” setting is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE, on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

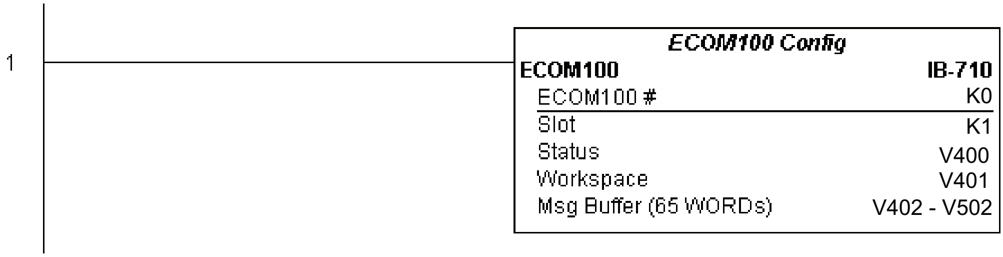
ECDHCPE Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Timeout(sec): specifies a timeout period so that the instruction may have time to complete
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written

Parameter		DL06 Range
ECOM100#	K	K0-255
Timeout (sec)	K	K5-127
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words

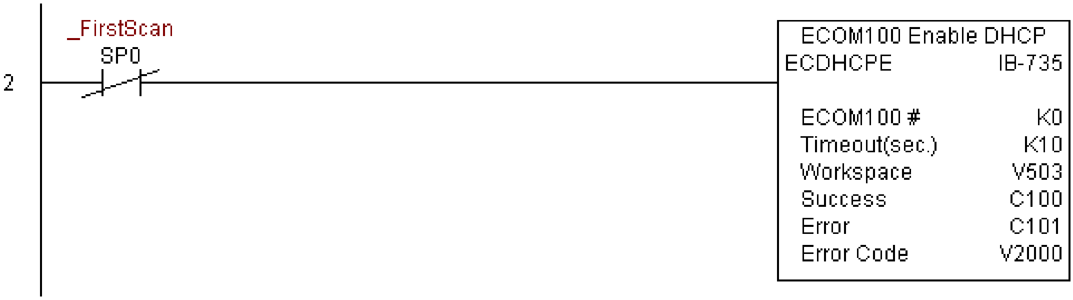
ECDHCPE Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



ECOM100 Config
ECOM100 IB-710
ECOM100 # K0
Slot K1
Status V400
Workspace V401
Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, enable DHCP in the ECOM100. DHCP is the same protocol used by PCs for using a DHCP Server to automatically assign the ECOM100's IP Address, Gateway Address, and Subnet Mask. Typically this is done using NetEdit, but this IBox allows you to enable DHCP in the ECOM100 using your ladder program. The ECDHCPE is leading edge triggered, not power-flow driven (similar to a counter input leg). The commands to enable DHCP will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON. The ECDHCPE does more than just set the bit to enable DHCP in the ECOM100, but it then polls the ECOM100 once every second to see if the ECOM100 has found a DHCP server and has a valid IP Address. Therefore, a timeout parameter is needed in case the ECOM100 cannot find a DHCP server. If a timeout does occur, the Error bit will turn on and the error code will be 1005 decimal. The Success bit will turn on only if the ECOM100 finds a DHCP Server and is assigned a valid IP Address. If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

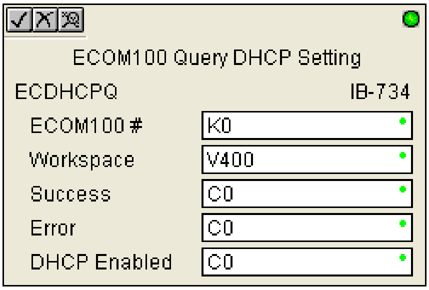


ECOM100 Enable DHCP
ECDHCPE IB-735
ECOM100 # K0
Timeout(sec.) K10
Workspace V503
Success C100
Error C101
Error Code V2000

ECOM100 Query DHCP Setting (ECDHCPQ) (IB-734)

DS	Used
HPP	N/A

ECOM100 Query DHCP Setting will determine if DHCP is enabled in the ECOM100 on a leading edge transition to the IBox. The DHCP Enabled bit parameter will be ON if DHCP is enabled, OFF if disabled.


ECOM100 Query DHCP Setting
ECDHCPQ IB-734
ECOM100 # K0
Workspace V400
Success C0
Error C0
DHCP Enabled C0

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

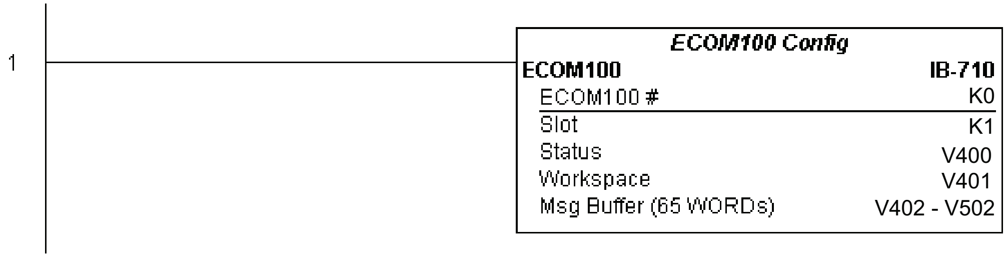
ECDHCPQ Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- DHCP Enabled: specifies a bit that will turn on if the ECOM100's DHCP is enabled or remain off if disabled - after instruction query, be sure to check the state of the Success/Error bit state along with DHCP Enabled bit state to confirm a successful module query

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
DHCP Enabled	X,Y,C,GX,GY,B	See DL06 V-memory map

ECDHCPQ Example

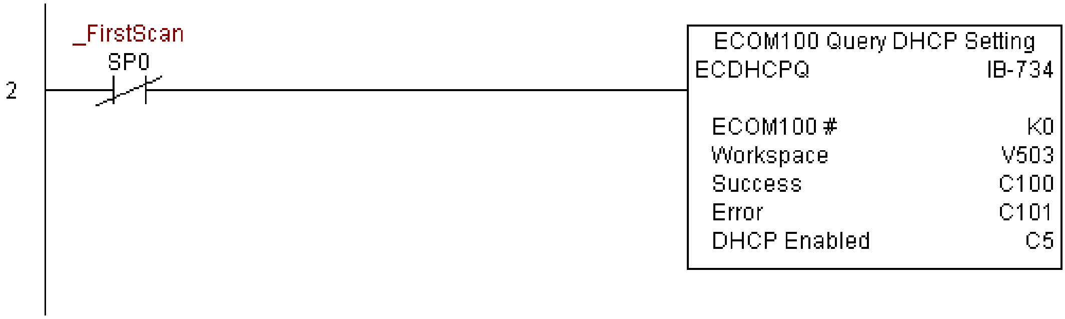
Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



ECOM100 Config

ECOM100	IB-710
ECOM100 #	K0
Slot	K1
Status	V400
Workspace	V401
Msg Buffer (65 WORDs)	V402 - V502

Rung 2: On the 2nd scan, read whether DHCP is enabled or disabled in the ECOM100 and store it in C5. DHCP is the same protocol used by PCs for using a DHCP Server to automatically assign the ECOM100's IP Address, Gateway Address, and Subnet Mask. The ECDHCPQ is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read (Query) whether DHCP is enabled or not will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON. If successful, turn on C100. If there is a failure, turn on C101.



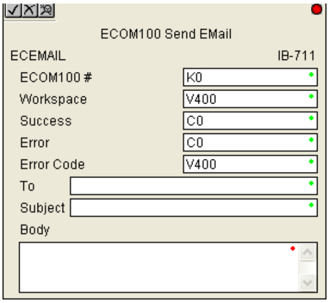
ECOM100 Query DHCP Setting

ECDHCPQ	IB-734
ECOM100 #	K0
Workspace	V503
Success	C100
Error	C101
DHCP Enabled	C5

ECOM100 Send E-mail (ECEMAIL) (IB-711)

DS	Used
HPP	N/A

ECOM100 Send EMail, on a leading edge transition, will behave as an EMail client and send an SMTP request to your SMTP Server to send the EMail message to the EMail addresses in the To: field and also to those listed in the Cc: list hard coded in the ECOM100. It will send the SMTP request based on the specified ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) at the top of your program.



ECOM100 Send EMail

ECEMAIL IB-711

ECOM100 #	K0
Workspace	V400
Success	C0
Error	C0
Error Code	V400
To	
Subject	
Body	

The Body: field supports what the PRINT and VPRINT instructions support for text and embedded variables, allowing you to embed real-time data in your EMail (e.g. “V2000 = “ V2000:B).

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the request is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), an SMPT protocol error (between 100 and 999), or a PLC logic error (greater than 1000).

Since the ECOM100 is only an EMail Client and requires access to an SMTP Server, you MUST have the SMTP parameters configured properly in the ECOM100 via the ECOM100’s Home Page and/or the EMail Setup instruction (ECEMSUP). To get to the ECOM100’s Home Page, use your favorite Internet browser and browse to the ECOM100’s IP Address, e.g. http://192.168.12.86

You are limited to approximately 100 characters of message data for the entire instruction, including the To: Subject: and Body: fields. To save space, the ECOM100 supports a hard coded list of EMail addresses for the Carbon Copy field (cc:) so that you can configure those in the ECOM100, and keep the To: field small (or even empty), to leave more room for the Subject: and Body: fields.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

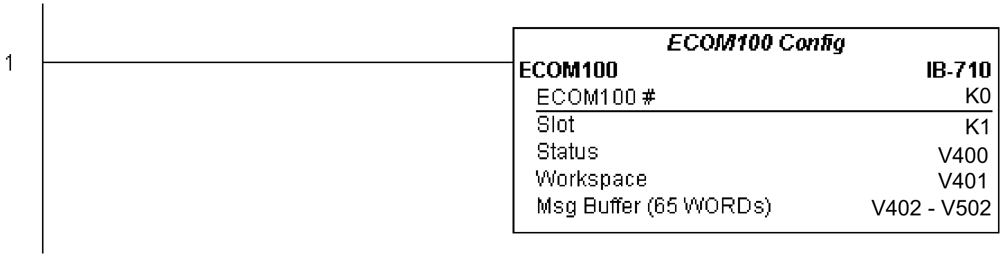
ECEMAIL Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- To: specifies an E-mail address that the message will be sent to
- Subject: subject of the e-mail message
- Body: supports what the PRINT and VPRINT instructions support for text and embedded variables, allowing you to embed real-time data in the EMail message

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map
To:		Text
Subject:		Text
Body:		See PRINT and VPRINT instructions

ECEMAIL Example

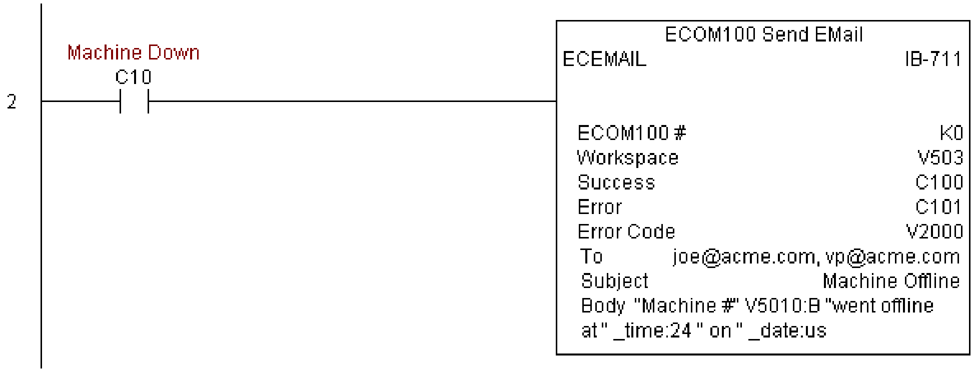
Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: When a machine goes down, send an email to Joe in maintenance and to the VP over production showing what machine is down along with the date/time stamp of when it went down.

The ECEMAIL is leading edge triggered, not power-flow driven (similar to a counter input leg). An email will be sent whenever the power flow into the IBox goes from OFF to ON. This helps prevent self inflicted spamming.

If the EMail is sent, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the SMTP error code or other possible error codes.



ECEMAIL Decimal Status Codes

This list of status codes is based on the list in the *ECOM100 Mock Slave Address 89 Command Specification*.

ECOM100 Status codes can be classified into four different areas based on its **decimal** value.

ECOM100 Status Code Areas	
0-1	Normal Status - no error
2-99	Internal ECOM100 errors
100-999	Standard TCP/IP protocol errors (SMTP, HTTP, etc)
1000+	IBox ladder logic assigned errors (SP Slot Error, etc)

For the ECOM100 Send EMail IBOX, the status codes below are specific to this IBox.

Normal Status 0 - 1

ECOMM100 Send EMAIL IBOX Status Codes	
0-1	Success - ECEMAIL completed successfully.
1	Busy - ECEMAIL IBOX logic sets the Error register to this value when the ECEMAIL starts a new request.

Internal ECOM100 Errors (2-99)

Internal ECOM 100 Errors (2-99)	
10-19	Timeout Errors - last digit shows where in ECOM100’s SMTP state logic the timeout occurred; *regardless of the last digit*, the SMTP conversation with the SMTP Server timed out.
	SMTP Internal Errors (20-29)
20	TCP Write Error
21	No Sendee
22	Invalid State
23	Invalid Data
24	Invalid SMTP Configuration
25	Memory Allocation Error

ECEMAIL IBox Ladder Logic Assigned Errors (1000+)

ECEMAIL IBox Ladder Logic Assigned Errors (1000+)	
101	SP Slot Error - the SP error bit for the ECOM100’s slot turned on. Possibly using RX or WX instructions on the ECOM100 and walking on the ECEMAIL execution. Use should use ECRX and ECX IBoxes.

ECEMAIL Decimal Status Codes

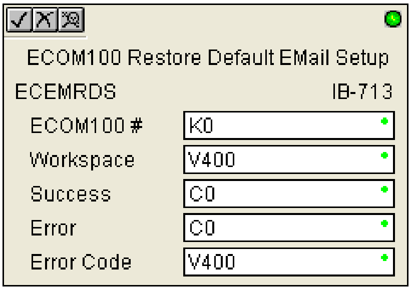
SMTP Protocol Errors - SMTP (100-999)

SMTP Protocol Errors - SMTP (100-999)	
Error	**Description**
1xx	Informational replies
2xx	Success replies
200	(Non-standard success response)
211	System status or system help reply
214	Help message
220	<domain> Service ready. Ready to start TLS
221	<domain> Service closing transmission channel
250	Ok, queuing for node <node> started. Requested mail action okay, completed
251	Ok, no messages waiting for node <node>. User not local; will forward to <forward-path>
252	OK, pending messages for node <node> started. Cannot VRFY user (e.g., info is not local), but will take message for this user and attempt delivery.
253	OK, messages pending messages for node <node> started
3xx	(re) direction replies
354	Start mail input; end with <CRLF>.<CRLF>
355	Octet-offset is the transaction offset.
4xx	client/request error replies
421	<domain> Service not available, closing transmission channel
432	A password transition is needed
450	Requested mail action not taken: mailbox unavailable. ATRN request refused.
451	Requested action aborted; local error in processing. Unable to process ATRN request now.
452	Requested action not taken: insufficient system storage
453	You have no mail
454	TLS is not available due to temporary reason. Encryption required for requested authentication mechanism.
458	Unable to queue messages for node <node>
459	Node <node> not allowed: <reason>
5xx	Server/process error replies
500	Syntax error, command unrecognized. Syntax error.
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command parameter not implemented
521	<domain> does not accept mail
530	Access denied. Must issue a STARTTLS command first. Encryption required for requested authentication mechanism.

ECOM100 Restore Default E-mail Setup (ECEMRDS) (IB-713)

DS	Used
HPP	N/A

ECOM100 Restore Default EMail Setup, on a leading edge transition, will restore the original EMail Setup data stored in the ECOM100 back to the working copy based on the specified ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) at the top of your program.


ECOM100 Restore Default EMail Setup
ECEMRDS IB-713
ECOM100 # K0
Workspace V400
Success C0
Error C0
Error Code V400

When the ECOM100 is first powered up, it copies the EMail setup data stored in ROM to the working copy in RAM. You can then modify this working copy from your program using the ECOM100 EMail Setup (ECEMSUP) IBox. After modifying the working copy, you can later restore the original setup data via your program by using this IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

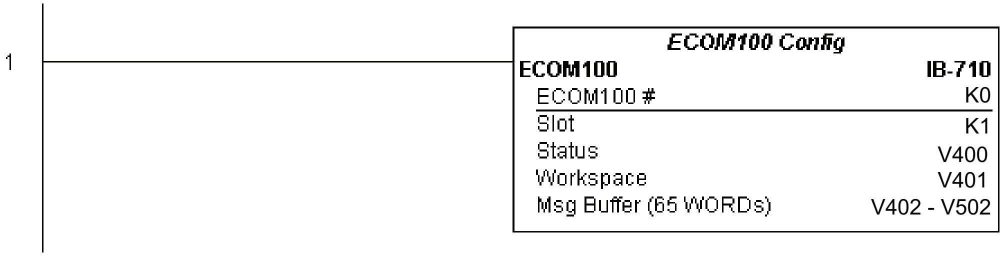
ECEMRDS Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words

ECEMRDS Example

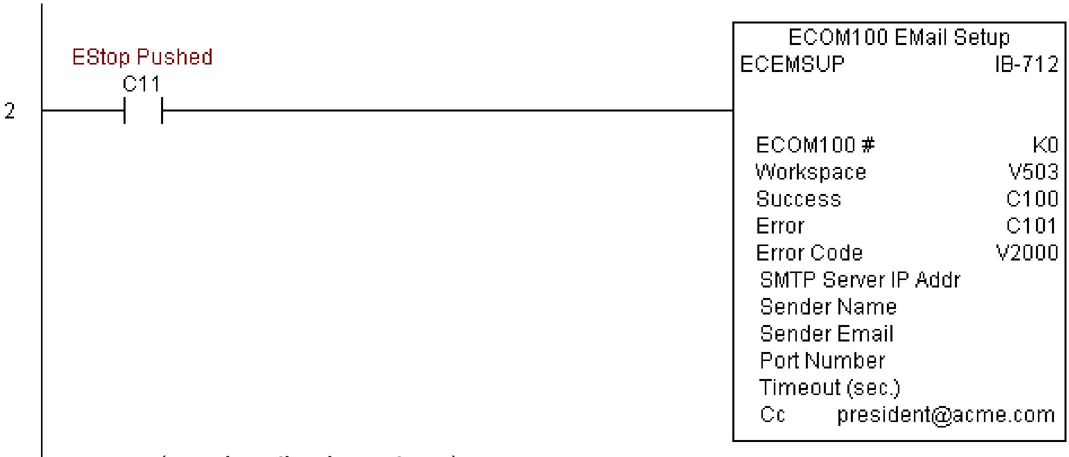
Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



```
1
                              ECOM100 Config
ECOM100                                             IB-710
  ECOM100 #                                             K0
  Slot                                                  K1
  Status                                              V400
  Workspace                                           V401
  Msg Buffer (65 WORDs)                        V402 - V502
```

Rung 2: Whenever an EStop is pushed, ensure that president of the company gets copies of all EMails being sent.

The ECOM100 EMail Setup IBox allows you to set/change the SMTP EMail settings stored in the ECOM100.



```
    EStop Pushed
2       C11                     ECOM100 EMail Setup
                                ECEMSUP                   IB-712

                                  ECOM100 #                   K0
                                  Workspace                 V503
                                  Success                   C100
                                  Error                     C101
                                  Error Code               V2000
                                  SMTP Server IP Addr
                                  Sender Name
                                  Sender Email
                                  Port Number
                                  Timeout (sec.)
                                  Cc       president@acme.com
```

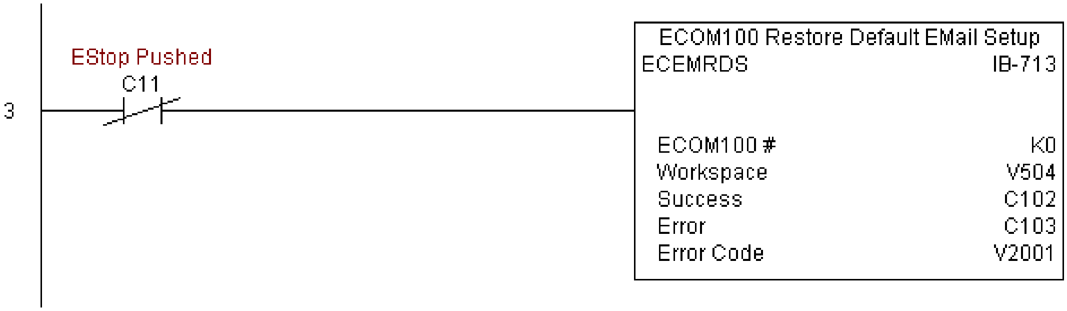
(example continued on next page)

ECEMRDS Example (cont'd)

Rung 3: Once the EStop is pulled out, take the president off the cc: list by restoring the default EMail setup in the ECOM100.

The ECEMRDS is leading edge triggered, not power-flow driven (similar to a counter input leg). The ROM based EMail configuration stored in the ECOM100 will be copied over the "working copy" whenever the power flow into the IBox goes from OFF to ON (the working copy can be changed by using the ECEMSUP IBox).

If successful, turn on C102. If there is a failure, turn on C103. If it fails, you can look at V2001 for the specific error code.

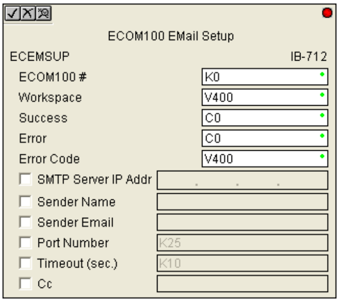


ECOM100 E-mail Setup (ECEMSUP) (IB-712)

DS	Used
HPP	N/A

ECOM100 EMail Setup, on a leading edge transition, will modify the working copy of the EMail setup currently in the ECOM100 based on the specified ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) at the top of your program.

You may pick and choose any or all fields to be modified using this instruction. Note that these changes are cumulative: if you execute multiple ECOM100 EMail Setup IBoxes, then all of the changes are made in the order they are executed. Also note that you can restore the original ECOM100 EMail Setup that is stored in the ECOM100 to the working copy by using the ECOM100 Restore Default EMail Setup (ECEMRDS) IBox.



ECOM100 EMail Setup

ECEMSUP IB-712

ECOM100 # K0

Workspace V400

Success C0

Error C0

Error Code V400

□ SMTP Server IP Addr . . .

□ Sender Name

□ Sender Email

□ Port Number K25

□ Timeout (sec.) K10

□ Cc

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

You are limited to approximately 100 characters/bytes of setup data for the entire instruction. So if needed, you could divide the entire setup across multiple ECEMSUP IBoxes on a field-by-field basis, for example do the Carbon Copy (cc:) field in one ECEMSUP IBox and the remaining setup parameters in another.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECEMSUP Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- SMTP Server IP Addr: optional parameter that specifies the IP Address of the SMTP Server on the ECOM100's network
- Sender Name: optional parameter that specifies the sender name that will appear in the "From:" field to those who receive the e-mail
- Sender EMail: optional parameter that specifies the sender EMail address that will appear in the "From:" field to those who receive the e-mail

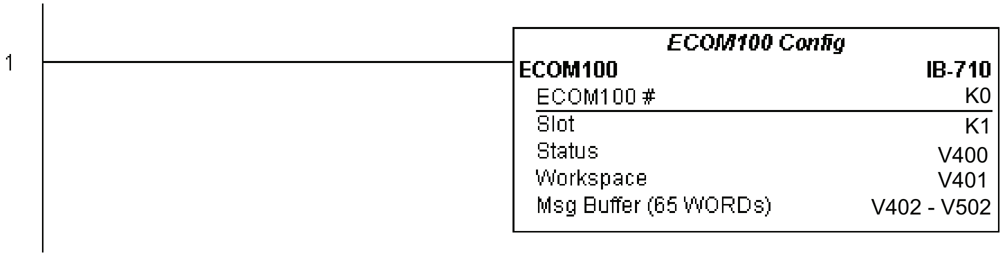
ECEMSUP Parameters (cont'd)

- Port Number: optional parameter that specifies the TCP/IP Port Number to send SMTP requests; usually this does not to be configured (see your network administrator for information on this setting)
- Timeout (sec): optional parameter that specifies the number of seconds to wait for the SMTP Server to send the EMail to all the recipients
- Cc: optional parameter that specifies a list of "carbon copy" Email addresses to send all EMails to

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words

ECEMSUP Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



ECOM100 Config

ECOM100 **IB-710**

ECOM100 # K0

Slot K1

Status V400

Workspace V401

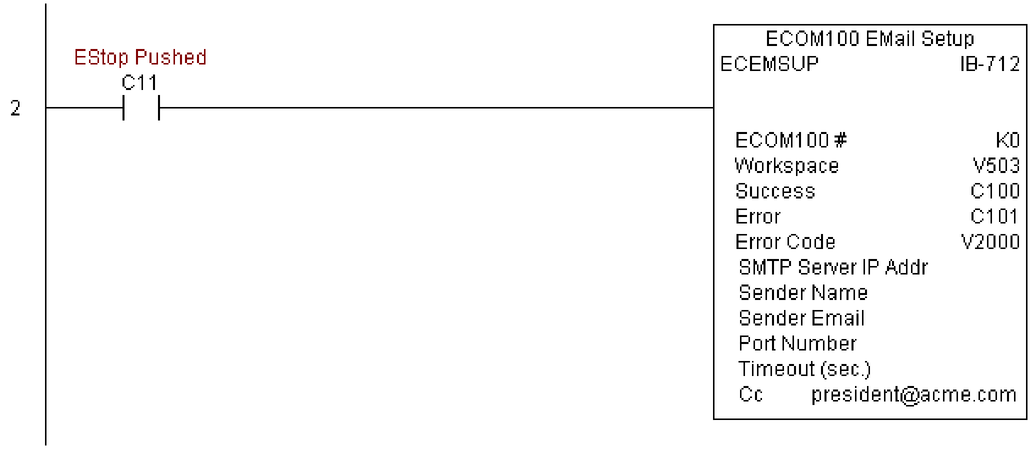
Msg Buffer (65 WORDs) V402 - V502

(example continued on next page)

ECEMSUP Example (cont'd)

Rung 2: Whenever an EStop is pushed, ensure that president of the company gets copies of all EMails being sent.The ECOM100 EMail Setup IBox allows you to set/change the SMTP EMail settings stored in the ECOM100. The ECEMSUP is leading edge triggered, not power-flow driven (similar to a counter input leg). At power-up, the ROM based EMail configuration stored in the ECOM100 is copied to a RAM based "working copy". You can change this working copy by using the ECEMSUP IBox. To restore the original ROM based configuration, use the Restore Default EMail Setup ECEMRDS IBox.

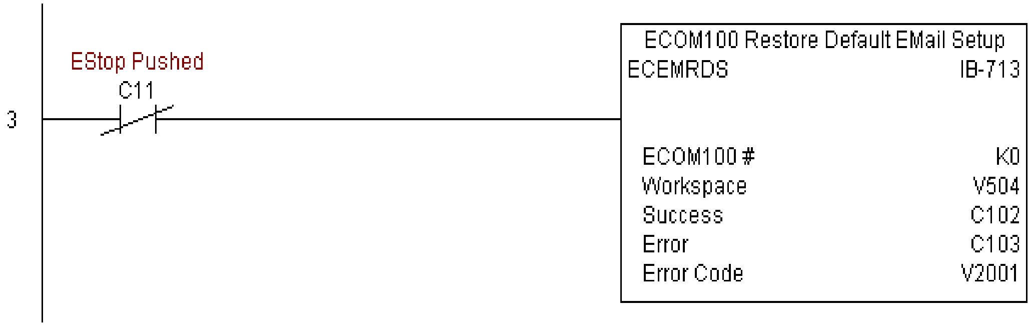
If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 EMail Setup
ECEMSUP IB-712

ECOM100 #	K0
Workspace	V503
Success	C100
Error	C101
Error Code	V2000
SMTP Server IP Addr	
Sender Name	
Sender Email	
Port Number	
Timeout (sec.)	
Cc	president@acme.com

Rung 3: Once the EStop is pulled out, take the president off the cc: list by restoring the default EMail setup in the ECOM100.



ECOM100 Restore Default EMail Setup
ECEMRDS IB-713

ECOM100 #	K0
Workspace	V504
Success	C102
Error	C103
Error Code	V2001

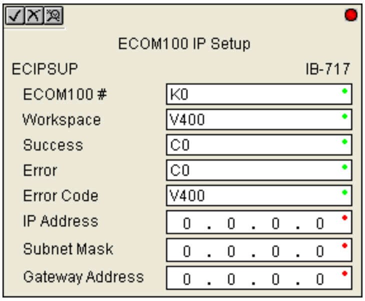
ECOM100 IP Setup (ECIPSUP) (IB-717)

DS	Used
HPP	N/A

ECOM100 IP Setup will configure the three TCP/IP parameters in the ECOM100: IP Address, Subnet Mask, and Gateway Address, on a leading edge transition to the IBox. The ECOM100 is specified by the ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) IBox at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



ECOM100 IP Setup

ECIPSUP IB-717

ECOM100 # K0

Workspace V400

Success C0

Error C0

Error Code V400

IP Address 0 . 0 . 0 . 0

Subnet Mask 0 . 0 . 0 . 0

Gateway Address 0 . 0 . 0 . 0

This setup data is stored in Flash-ROM in the ECOM100 and will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

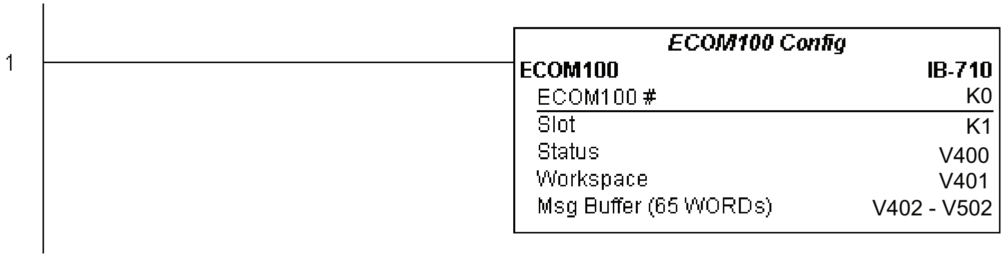
ECIPSUP Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- IP Address: specifies the module's IP Address
- Subnet Mask: specifies the Subnet Mask for the module to use
- Gateway Address: specifies the Gateway Address for the module to use

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
IP Address	IP Address	0.0.0.1. to 255.255.255.254
Subnet Mask Address	IP Address Mask	0.0.0.1. to 255.255.255.254
Gateway Address	IP Address	0.0.0.1. to 255.255.255.254

ECIPSUP Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



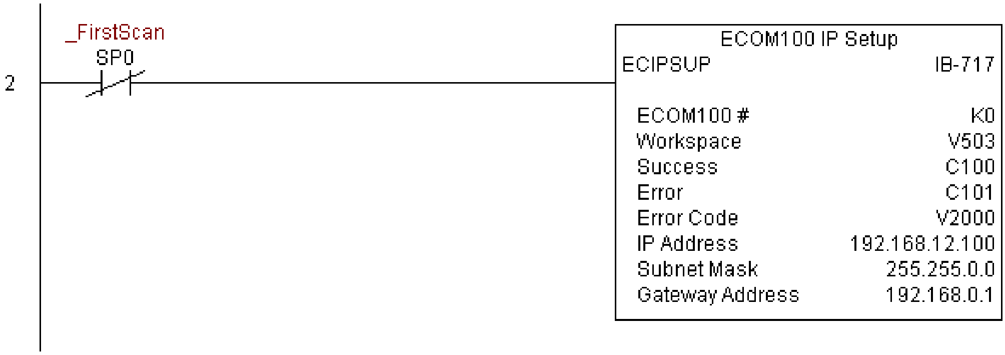
ECOM100 # K0
Slot K1
Status V400
Workspace V401
Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, configure all of the TCP/IP parameters in the ECOM100:

IP Address:	192.168. 12.100
Subnet Mask:	255.255. 0. 0
Gateway Address:	192.168. 0. 1

The ECIPSUP is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the TCP/IP configuration parameters will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

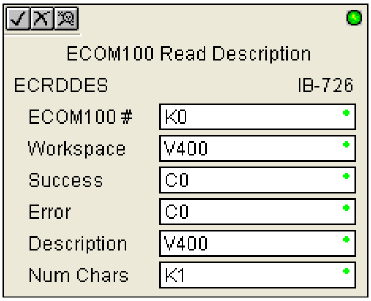


ECOM100 # K0
Workspace V503
Success C100
Error C101
Error Code V2000
IP Address 192.168.12.100
Subnet Mask 255.255.0.0
Gateway Address 192.168.0.1

ECOM100 Read Description (ECRDDES) (IB-726)

DS	Used
HPP	N/A

ECOM100 Read Description will read the ECOM100’s Description field up to the number of specified characters on a leading edge transition to the IBox.



ECOM100 Read Description

ECRDDES IB-726

ECOM100 # K0

Workspace V400

Success C0

Error C0

Description V400

Num Chars K1

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

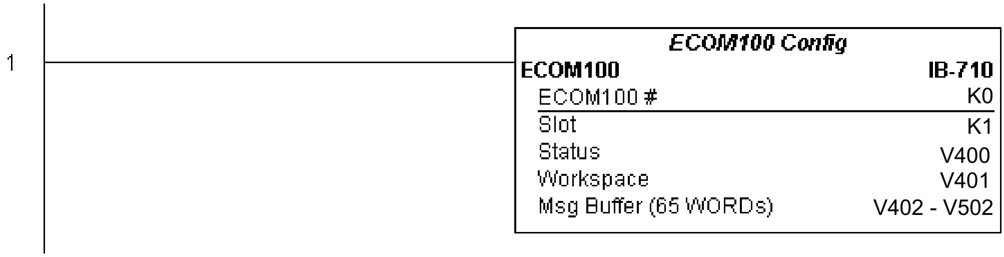
ECRDDES Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Description: specifies the starting buffer location where the ECOM100’s Module Name will be placed
- Num Char: specifies the number of characters (bytes) to read from the ECOM100’s Description field

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Description	V	See DL06 V-memory map - Data Words
Num Chars	K	K1-128

ECRDDES Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

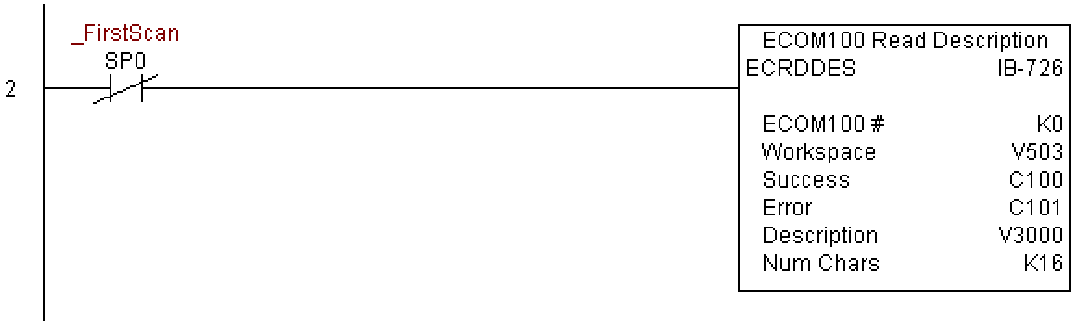


ECOM100 Config
ECOM100 **IB-710**
ECOM100 # K0
Slot K1
Status V400
Workspace V401
Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, read the Module Description of the ECOM100 and store it in V3000 thru V3007 (16 characters). This text can be displayed by an HMI.

The ECRDDES is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the module description will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.

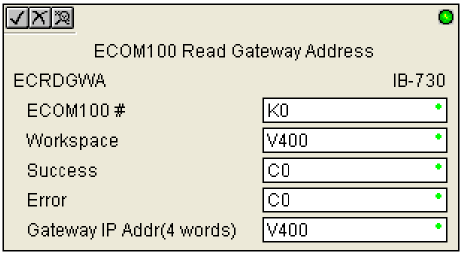


ECOM100 Read Description
ECRDDES IB-726
ECOM100 # K0
Workspace V503
Success C100
Error C101
Description V3000
Num Chars K16

ECOM100 Read Gateway Address (ECRDGWA) (IB-730)

DS	Used
HPP	N/A

ECOM100 Read Gateway Address will read the 4 parts of the Gateway IP address and store them in 4 consecutive V-Memory locations in decimal format, on a leading edge transition to the IBox.



ECOM100 #	K0
Workspace	V400
Success	C0
Error	C0
Gateway IP Addr(4 words)	V400

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

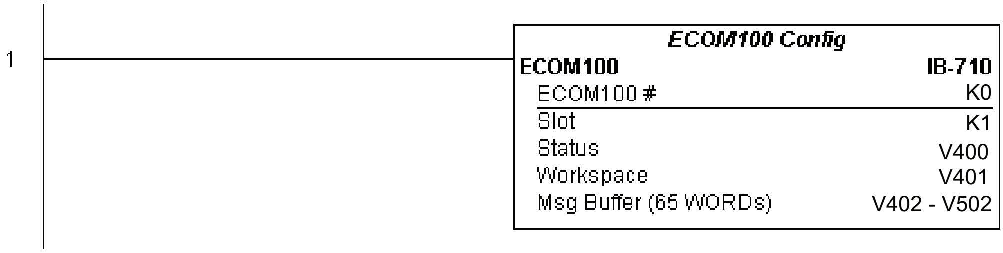
ECRDGWA Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Gateway IP Addr: specifies the starting address where the ECOM100's Gateway Address will be placed in 4 consecutive V-memory locations

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Gateway IP Address (4 Words)	V	See DL06 V-memory map - Data Words

ECRDGWA Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

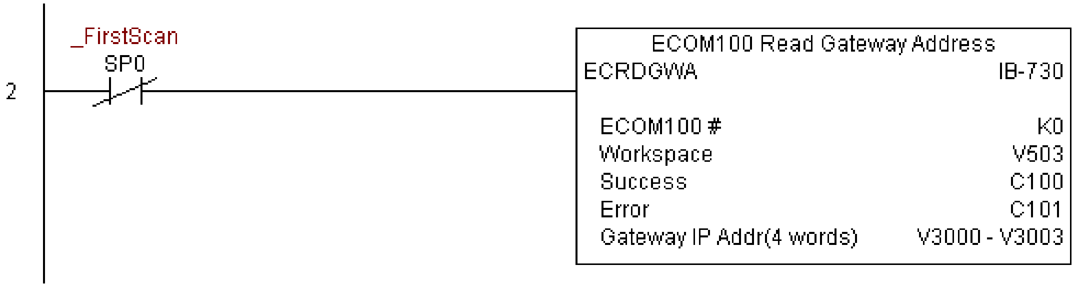


ECOM100 Config
ECOM100 IB-710
ECOM100 # K0
Slot K1
Status V400
Workspace V401
Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, read the Gateway Address of the ECOM100 and store it in V3000 thru V3003 (4 decimal numbers). The ECOM100's Gateway Address could be displayed by an HMI.

The ECRDGWA is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the Gateway Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.

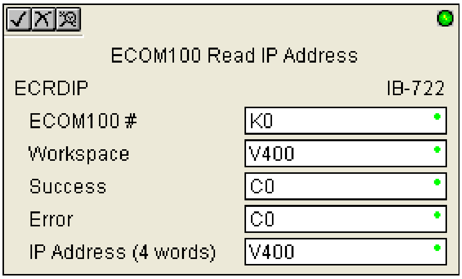


ECOM100 Read Gateway Address
ECRDGWA IB-730
ECOM100 # K0
Workspace V503
Success C100
Error C101
Gateway IP Addr(4 words) V3000 - V3003

ECOM100 Read IP Address (ECRDIP) (IB-722)

DS	Used
HPP	N/A

ECOM100 Read IP Address will read the 4 parts of the IP address and store them in 4 consecutive V-Memory locations in decimal format, on a leading edge transition to the IBox.



ECOM100 Read IP Address

ECRDIP IB-722

ECOM100 #	K0
Workspace	V400
Success	C0
Error	C0
IP Address (4 words)	V400

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

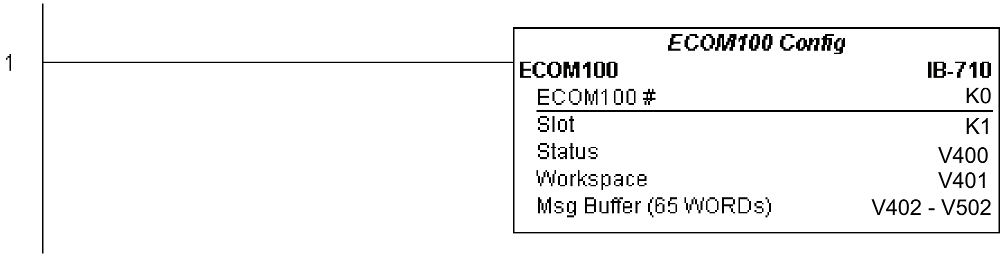
ECRDIP Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- IP Address: specifies the starting address where the ECOM100's IP Address will be placed in 4 consecutive V-memory locations

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
IP Address (4 Words)	V	See DL06 V-memory map - Data Words

ECRDIP Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



ECOM100 Config

ECOM100 **IB-710**

ECOM100 # K0

Slot K1

Status V400

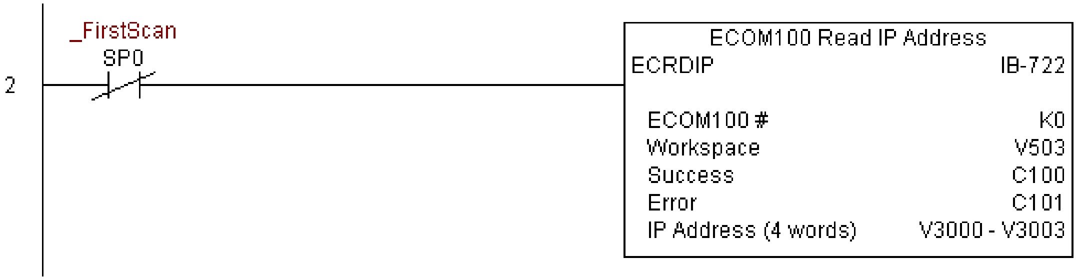
Workspace V401

Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, read the IP Address of the ECOM100 and store it in V3000 thru V3003 (4 decimal numbers). The ECOM100's IP Address could be displayed by an HMI.

The ECRDIP is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the IP Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Read IP Address

ECRDIP IB-722

ECOM100 # K0

Workspace V503

Success C100

Error C101

IP Address (4 words) V3000 - V3003

ECOM100 Read Module ID (ECRDMID) (IB-720)

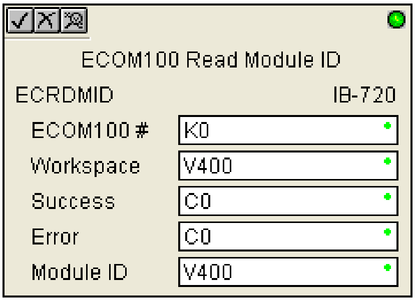
DS	Used
HPP	N/A

ECOM100 Read Module ID will read the binary (decimal) WORD sized Module ID on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.



ECOM100 Read Module ID

ECRDMID IB-720

ECOM100 # K0

Workspace V400

Success C0

Error C0

Module ID V400

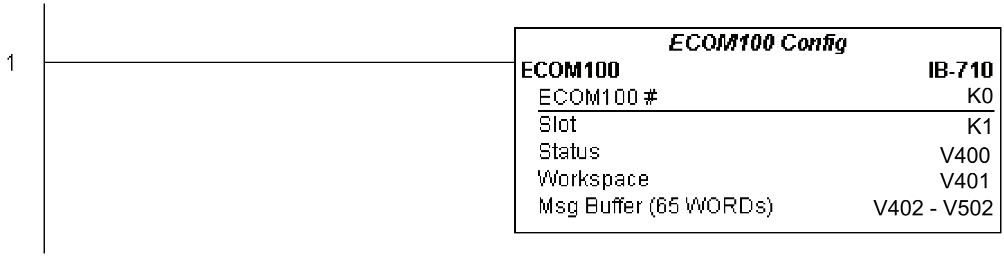
ECRDMID Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Module ID: specifies the location where the ECOM100's Module ID (decimal) will be placed

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Module ID	V	See DL06 V-memory map - Data Words

ECRDMID Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

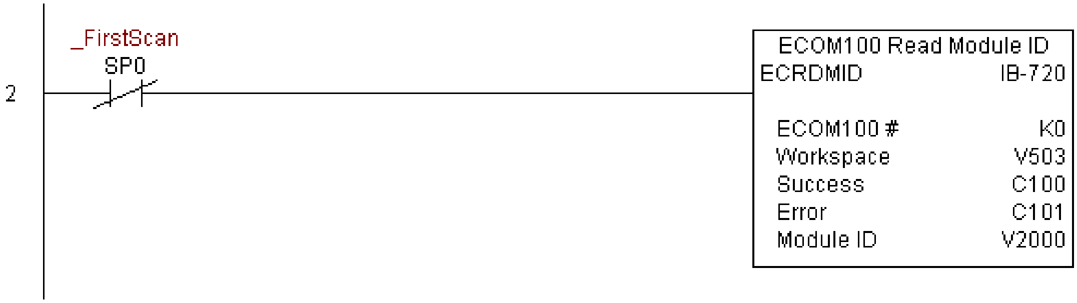


ECOM100 Config
ECOM100 **IB-710**
ECOM100 # K0
Slot K1
Status V400
Workspace V401
Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, read the Module ID of the ECOM100 and store it in V2000.

The ECRDMID is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the module ID will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Read Module ID
ECRDMID IB-720
ECOM100 # K0
Workspace V503
Success C100
Error C101
Module ID V2000

ECOM100 Read Module Name (ECRDNAM) (IB-724)

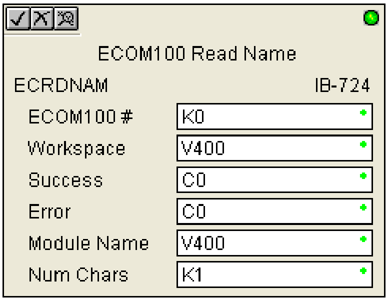
DS	Used
HPP	N/A

ECOM100 Read Name will read the Module Name up to the number of specified characters on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.



ECOM100 Read Name

ECRDNAM IB-724

ECOM100 # K0

Workspace V400

Success C0

Error C0

Module Name V400

Num Chars K1

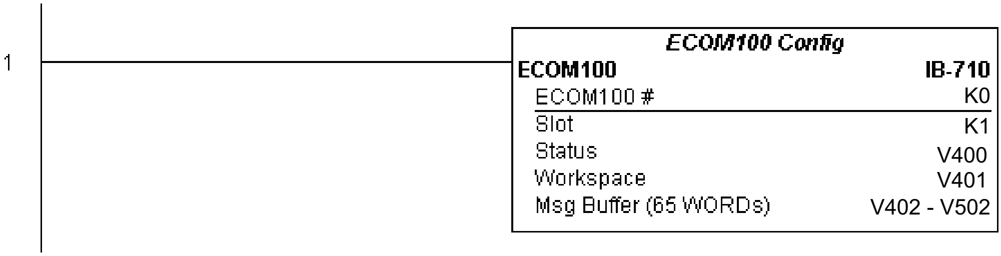
ECRDNAM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Module Name: specifies the starting buffer location where the ECOM100's Module Name will be placed
- Num Chars: specifies the number of characters (bytes) to read from the ECOM100's Name field

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Module Name	V	See DL06 V-memory map - Data Words
Num Chars	K	K1-128

ECRDNAM Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

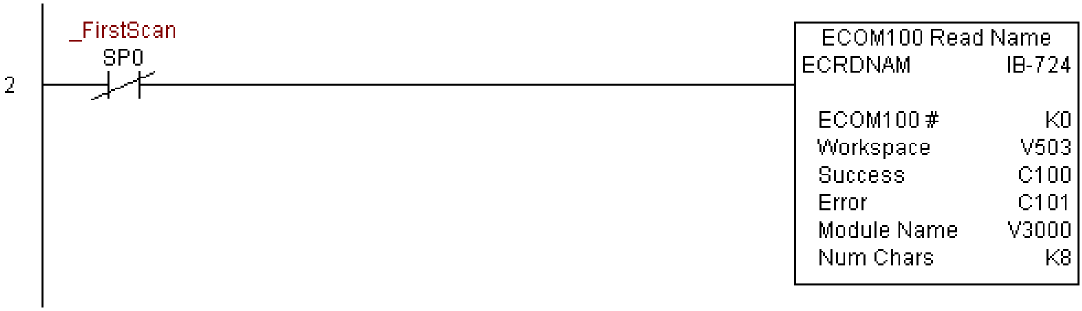


```
ECOM100 Config
ECOM100                     IB-710
  ECOM100 #                     K0
  Slot                          K1
  Status                      V400
  Workspace                   V401
  Msg Buffer (65 WORDs) V402 - V502
```

Rung 2: On the 2nd scan, read the Module Name of the ECOM100 and store it in V3000 thru V3003 (8 characters). This text can be displayed by an HMI.

The ECRDNAM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the module name will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



```
 ECOM100 Read Name
ECRDNAM         IB-724

 ECOM100 #          K0
 Workspace        V503
 Success          C100
 Error            C101
 Module Name     V3000
 Num Chars          K8
```

ECOM100 Read Subnet Mask (ECRDSNM) (IB-732)

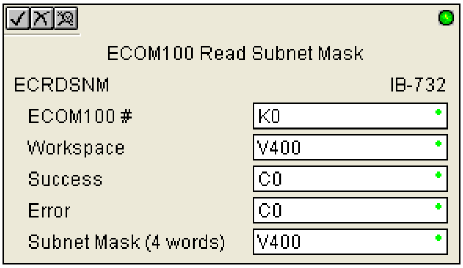
DS	Used
HPP	N/A

ECOM100 Read Subnet Mask will read the 4 parts of the Subnet Mask and store them in 4 consecutive V-Memory locations in decimal format, on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.



ECOM100 Read Subnet Mask

ECRDSNM IB-732

ECOM100 # K0

Workspace V400

Success C0

Error C0

Subnet Mask (4 words) V400

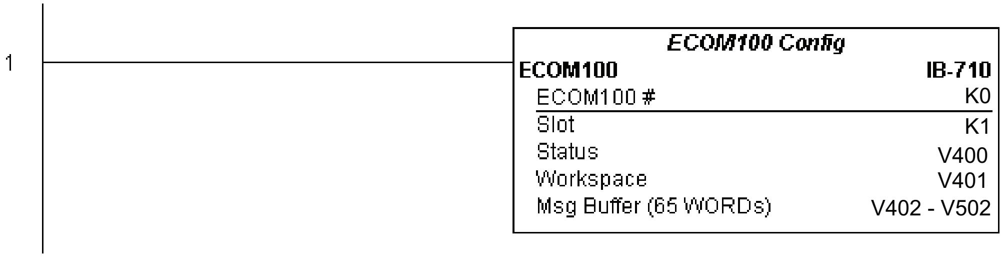
ECRDSNM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Subnet Mask: specifies the starting address where the ECOM100's Subnet Mask will be placed in 4 consecutive V-memory locations

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Subnet Mask (4 Words)	V	See DL06 V-memory map - Data Words

ECRDSNM Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



ECOM100 Config

ECOM100 IB-710

ECOM100 # K0

Slot K1

Status V400

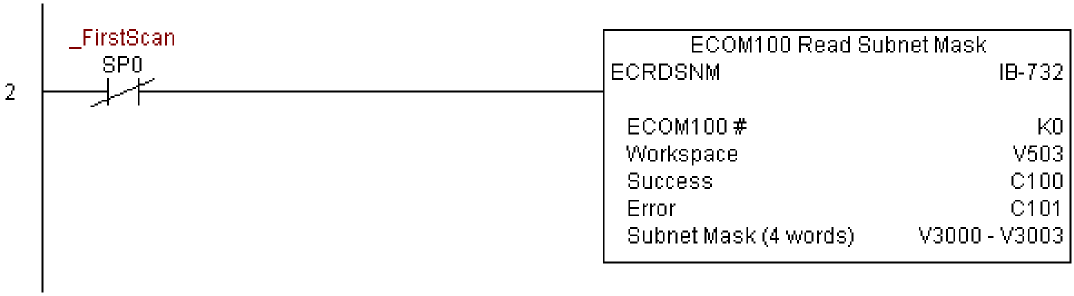
Workspace V401

Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, read the Subnet Mask of the ECOM100 and store it in V3000 thru V3003 (4 decimal numbers). The ECOM100's Subnet Mask could be displayed by an HMI.

The ECRDSNM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the Subnet Mask will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Read Subnet Mask

ECRDSNM IB-732

ECOM100 # K0

Workspace V503

Success C100

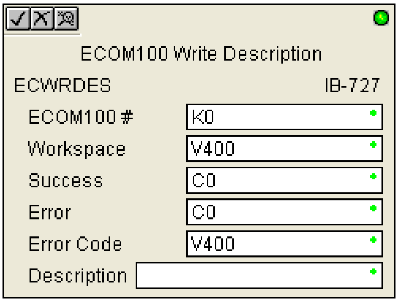
Error C101

Subnet Mask (4 words) V3000 - V3003

ECOM100 Write Description (ECWRDES) (IB-727)

DS	Used
HPP	N/A

ECOM100 Write Description will write the given Description to the ECOM100 module on a leading edge transition to the IBox. If you use a dollar sign ($) or double quote ("), use the PRINT/VPRINT escape sequence of TWO dollar signs ($$) for a single dollar sign or dollar sign-double quote ($") for a double quote character.



ECOM100 # K0
Workspace V400
Success C0
Error C0
Error Code V400
Description

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

The Description is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

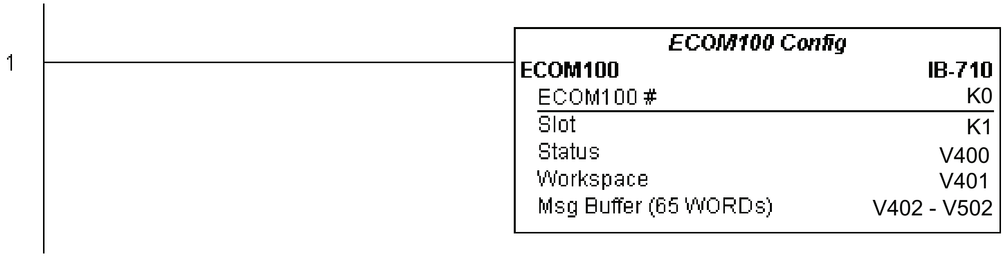
ECWRDES Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- Description: specifies the Description that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
Description		Text

ECWRDES Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



ECOM100 Config

ECOM100 **IB-710**

ECOM100 # K0

Slot K1

Status V400

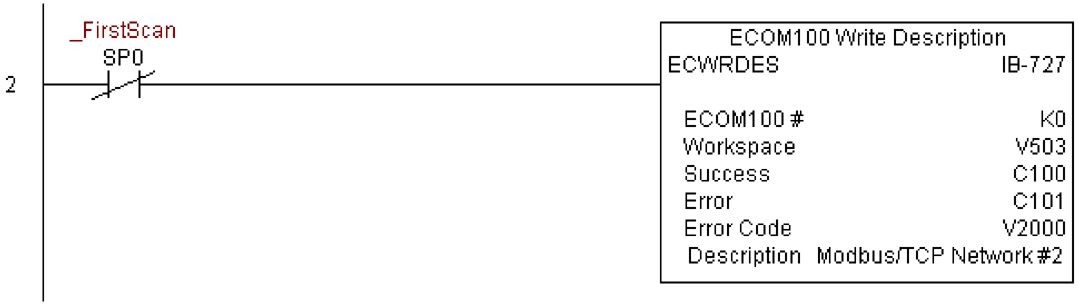
Workspace V401

Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, set the Module Description of the ECOM100. Typically this is done using NetEdit, but this IBox allows you to configure the module description in the ECOM100 using your ladder program.

The EWRDES is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the module description will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 Write Description

ECWRDES IB-727

ECOM100 # K0

Workspace V503

Success C100

Error C101

Error Code V2000

Description Modbus/TCP Network #2

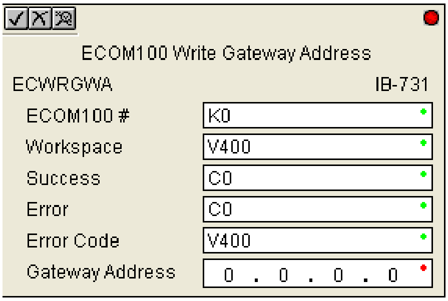
ECOM100 Write Gateway Address (ECWRGWA) (IB-731)

DS	Used
HPP	N/A

ECOM100 Write Gateway Address will write the given Gateway IP Address to the ECOM100 module on a leading edge transition to the IBox. See also ECOM100 IP Setup (ECIPSUP) IBox 717 to setup ALL of the TCP/IP parameters in a single instruction - IP Address, Subnet Mask, and Gateway Address.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



ECOM100 Write Gateway Address

ECWRGWA IB-731

ECOM100 # K0

Workspace V400

Success C0

Error C0

Error Code V400

Gateway Address 0 . 0 . 0 . 0

The Gateway Address is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE, on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

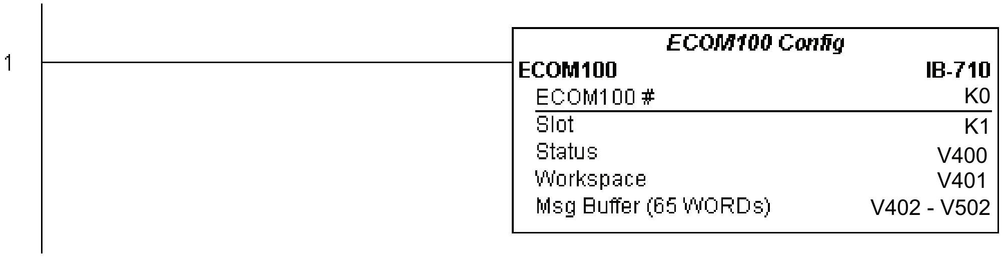
ECWRGWA Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- Gateway Address: specifies the Gateway IP Address that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
Gateway Address		0.0.0.1. to 255.255.255.254

ECWRGWA Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



1

ECOM100 Config

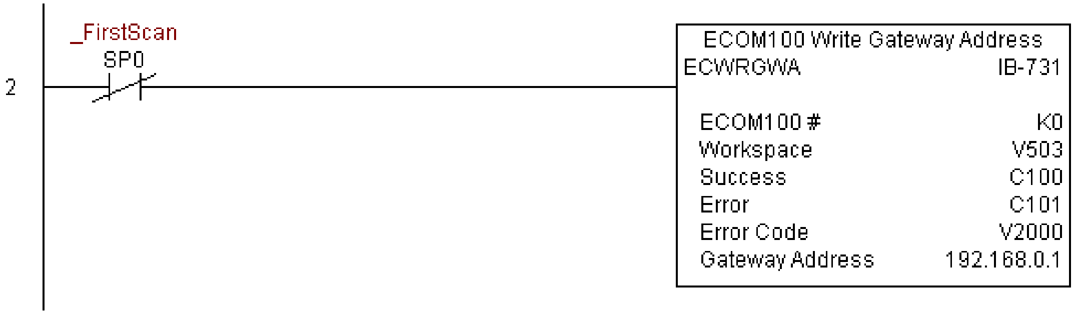
ECOM100	IB-710
ECOM100 #	K0
Slot	K1
Status	V400
Workspace	V401
Msg Buffer (65 WORDs)	V402 - V502

Rung 2: On the 2nd scan, assign the Gateway Address of the ECOM100 to 192.168.0.1

The ECWRGWA is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the Gateway Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

To configure all of the ECOM100 TCP/IP parameters in one IBox, see the ECOM100 IP Setup (ECIPSUP) IBox.



2

_FirstScan
SP0

ECOM100 Write Gateway Address

ECWRGWA	IB-731
ECOM100 #	K0
Workspace	V503
Success	C100
Error	C101
Error Code	V2000
Gateway Address	192.168.0.1

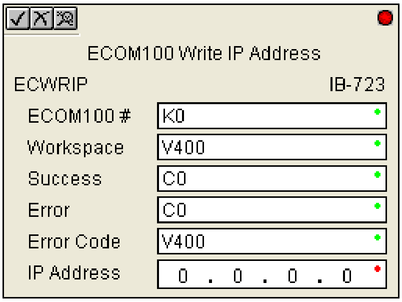
ECOM100 Write IP Address (ECWRIP) (IB-723)

DS	Used
HPP	N/A

ECOM100 Write IP Address will write the given IP Address to the ECOM100 module on a leading edge transition to the IBox. See also ECOM100 IP Setup (ECIPSUP) IBox 717 to setup ALL of the TCP/IP parameters in a single instruction - IP Address, Subnet Mask, and Gateway Address.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



ECOM100 Write IP Address

ECWRIP IB-723

ECOM100 # K0

Workspace V400

Success C0

Error C0

Error Code V400

IP Address 0 . 0 . 0 . 0

The IP Address is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

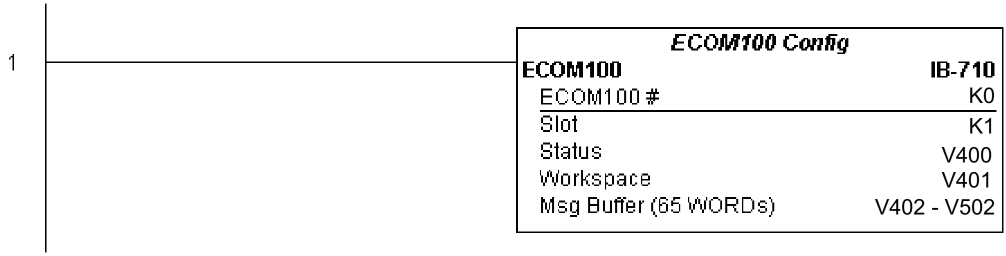
ECWRIP Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- IP Address: specifies the IP Address that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
IP Address		0.0.0.1. to 255.255.255.254

ECWRIP Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



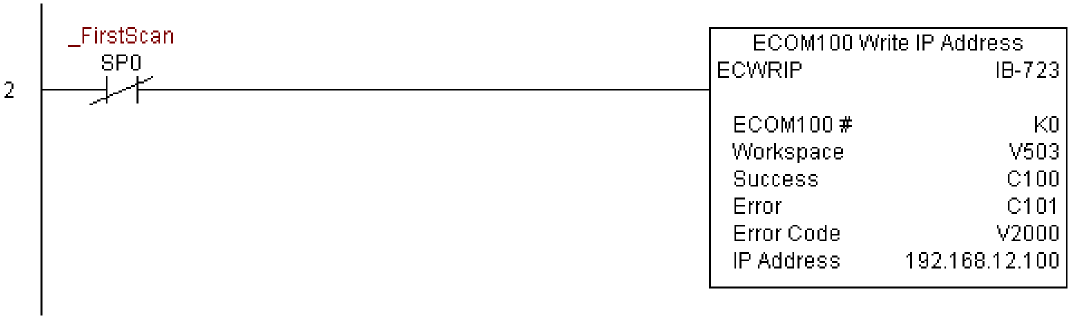
ECOM100 Config
ECOM100 IB-710
ECOM100 # K0
Slot K1
Status V400
Workspace V401
Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, assign the IP Address of the ECOM100 to 192.168.12.100

The ECWRIP is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the IP Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

To configure all of the ECOM100 TCP/IP parameters in one IBox, see the ECOM100 IP Setup (ECIPSUP) IBox.



ECOM100 Write IP Address
ECWRIP IB-723
ECOM100 # K0
Workspace V503
Success C100
Error C101
Error Code V2000
IP Address 192.168.12.100

ECOM100 Write Module ID (ECWRMID) (IB-721)

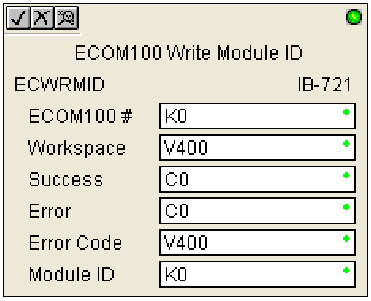
DS	Used
HPP	N/A

ECOM100 Write Module ID will write the given Module ID on a leading edge transition to the IBox

If the Module ID is set in the hardware using the dipswitches, this IBox will fail and return error code 1005 (decimal).

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



ECOM100 Write Module ID

ECWRMID IB-721

ECOM100 #	K0
Workspace	V400
Success	C0
Error	C0
Error Code	V400
Module ID	K0

The Module ID is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

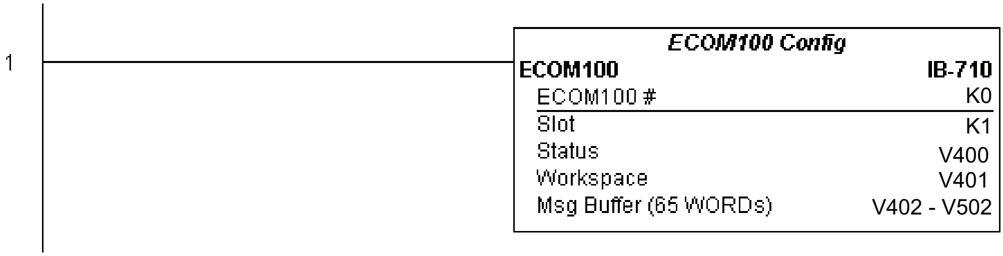
ECWRMID Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- Module ID: specifies the Module ID that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
Module ID		K0-65535

ECWRMID Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



ECOM100 Config

ECOM100 IB-710

ECOM100 # K0

Slot K1

Status V400

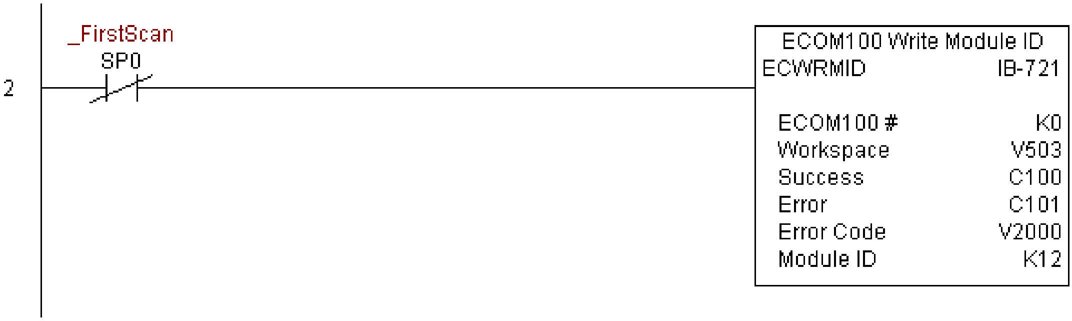
Workspace V401

Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, set the Module ID of the ECOM100. Typically this is done using NetEdit, but this IBox allows you to configure the module ID of the ECOM100 using your ladder program.

The EWRMID is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the module ID will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 Write Module ID

ECWRMID IB-721

ECOM100 # K0

Workspace V503

Success C100

Error C101

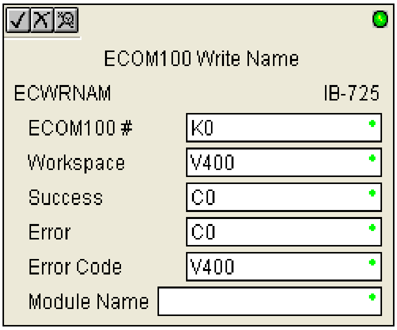
Error Code V2000

Module ID K12

ECOM100 Write Name (ECWRNAM) (IB-725)

DS	Used
HPP	N/A

ECOM100 Write Name will write the given Name to the ECOM100 module on a leading edge transition to the IBox. If you use a dollar sign ($) or double quote ("), use the PRINT/VPRINT escape sequence of TWO dollar signs ($$) for a single dollar sign or dollar sign-double quote ($") for a double quote character.



ECOM100 Write Name

ECWRNAM IB-725

ECOM100 # K0

Workspace V400

Success C0

Error C0

Error Code V400

Module Name

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

The Name is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

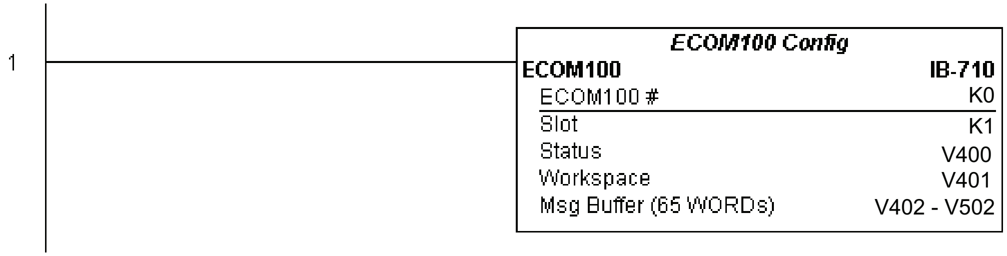
ECWRNAM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- Module Name: specifies the Name that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
Module Name		Text

ECWRNAM Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

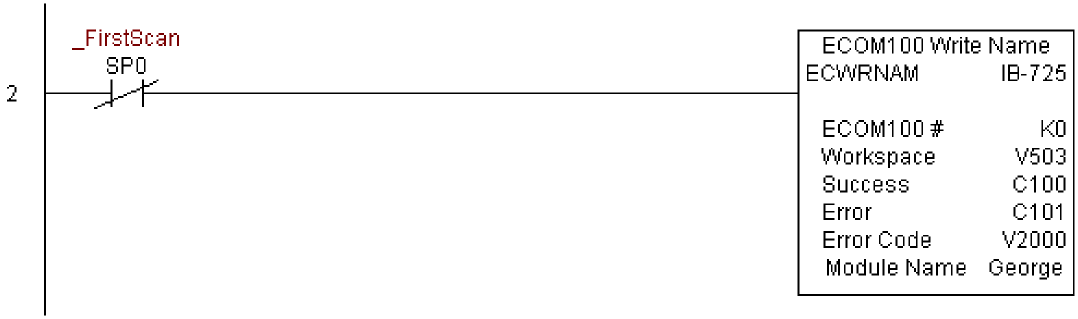


ECOM100 Config
ECOM100 IB-710
ECOM100 # K0
Slot K1
Status V400
Workspace V401
Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, set the Module Name of the ECOM100. Typically this is done using NetEdit, but this IBox allows you to configure the module name of the ECOM100 using your ladder program.

The EWRNAM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the module name will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 Write Name
ECWRNAM IB-725
ECOM100 # K0
Workspace V503
Success C100
Error C101
Error Code V2000
Module Name George

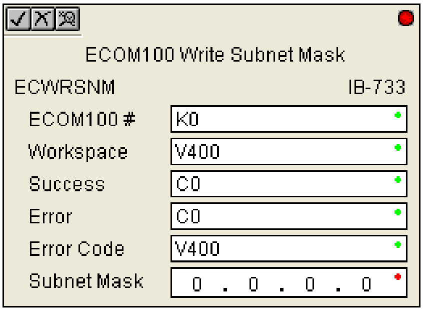
ECOM100 Write Subnet Mask (ECWRSNM) (IB-733)

DS	Used
HPP	N/A

ECOM100 Write Subnet Mask will write the given Subnet Mask to the ECOM100 module on a leading edge transition to the IBox. See also ECOM100 IP Setup (ECIPSUP) IBox 717 to setup ALL of the TCP/IP parameters in a single instruction - IP Address, Subnet Mask, and Gateway Address.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



ECOM100 Write Subnet Mask

ECWRSNM IB-733

ECOM100 # K0

Workspace V400

Success C0

Error C0

Error Code V400

Subnet Mask 0 . 0 . 0 . 0

The Subnet Mask is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

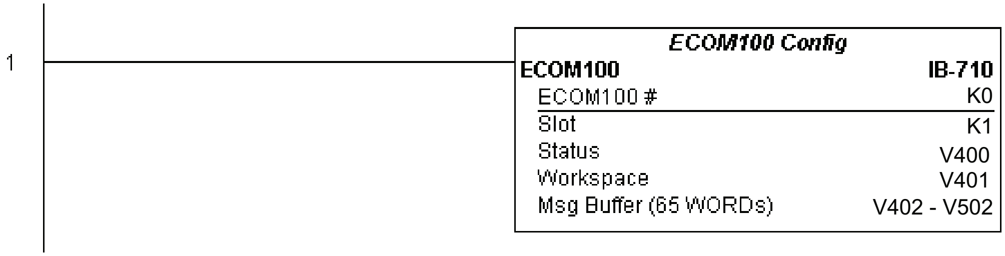
ECWRSNM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- Subnet Mask: specifies the Subnet Mask that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
Subnet Mask		Masked IP Address

ECWRSNM Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



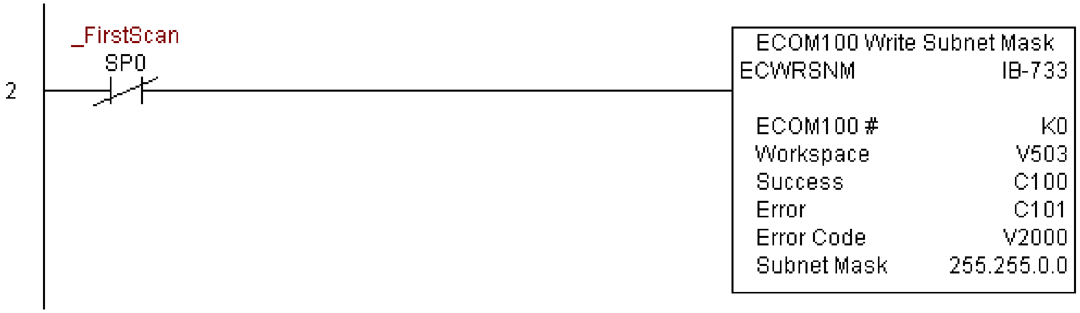
ECOM100 Config
ECOM100 IB-710
ECOM100 # K0
Slot K1
Status V400
Workspace V401
Msg Buffer (65 WORDs) V402 - V502

Rung 2: On the 2nd scan, assign the Subnet Mask of the ECOM100 to 255.255.0.0

The ECWRSNM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the Subnet Mask will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

To configure all of the ECOM100 TCP/IP parameters in one IBox, see the ECOM100 IP Setup (ECIPSUP) IBox.



ECOM100 Write Subnet Mask
ECWRSNM IB-733
ECOM100 # K0
Workspace V503
Success C100
Error C101
Error Code V2000
Subnet Mask 255.255.0.0

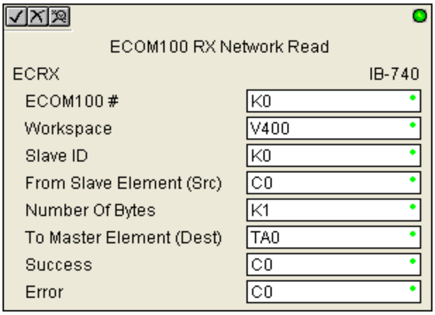
ECOM100 RX Network Read (ECRX) (IB-740)

DS	Used
HPP	N/A

ECOM100 RX Network Read performs the RX instruction with built-in interlocking with all other ECOM100 RX (ECRX) and ECOM100 WX (ECWX) IBoxes in your program to simplify communications networking. It will perform the RX on the specified ECOM100#'s network, which corresponds to a specific unique ECOM100 Configuration (ECOM100) IBox at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Whenever this IBox has power, it will read element data from the specified slave into the given destination V-Memory buffer, giving other ECOM100 RX and ECOM100 WX IBoxes on that ECOM100# network a chance to execute.



ECOM100 RX Network Read

ECRX	IB-740
ECOM100 #	K0
Workspace	V400
Slave ID	K0
From Slave Element (Src)	C0
Number Of Bytes	K1
To Master Element (Dest)	TA0
Success	C0
Error	C0

For example, if you wish to read and write data continuously from 5 different slaves, you can have all of these ECRX and ECWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically.

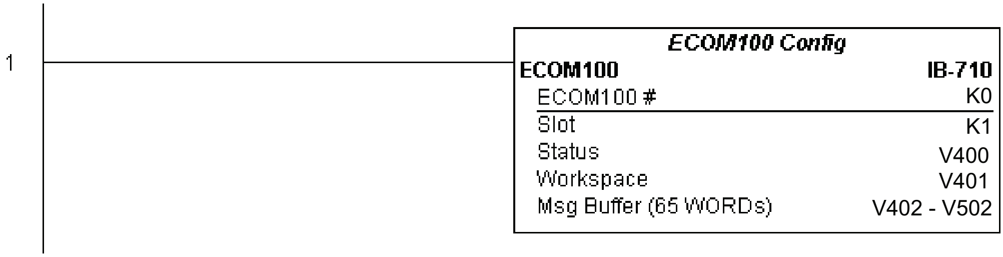
ECRX Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Slave ID: specifies the slave ECOM(100) PLC that will be targeted by the ECRX instruction
- From Slave Element (Src): specifies the slave address of the data to be read
- Number of Bytes: specifies the number of bytes to read from the slave ECOM(100) PLC
- To Master Element (Dest): specifies the location where the slave data will be placed in the master ECOM100 PLC
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Slave ID	K	K0-90
From Slave Element (Src)	X,Y,C,S,T,CT,GX,GY,V,P	See DL06 V-memory map
Number of Bytes	K	K1-128
To Master Element (Dest)	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

ECRX Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



ECOM100 Config
ECOM100 IB-710
ECOM100 # K0
Slot K1
Status V400
Workspace V401
Msg Buffer (65 WORDs) V402 - V502

(example continued on next page)

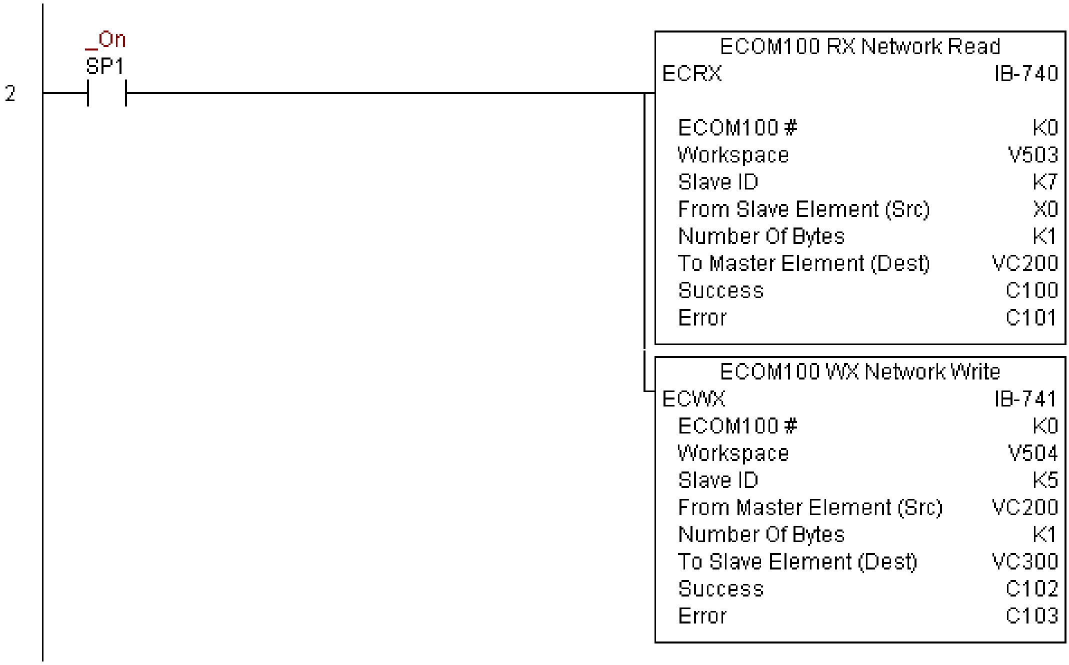
ECRX Example (cont'd)

Rung 2: Using ECOM100# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the ECRX and ECWX work with the ECOM100 Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP "busy bits" or "error bits", or what slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the ECRX and ECWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending ECOM100 IBoxes below the ECWX, then the very next scan the ECRX would start its request again.

Using the ECRX and ECWX for all of your ECOM100 network reads and writes is the fastest the PLC can do networking. For local Serial Ports, DCM modules, or the original ECOM modules, use the NETCFG and NETRX/NETWX IBoxes.



ECOM100 RX Network Read
ECRX IB-740

ECOM100 #	K0
Workspace	V503
Slave ID	K7
From Slave Element (Src)	X0
Number Of Bytes	K1
To Master Element (Dest)	VC200
Success	C100
Error	C101

ECOM100 WX Network Write
ECWX IB-741

ECOM100 #	K0
Workspace	V504
Slave ID	K5
From Master Element (Src)	VC200
Number Of Bytes	K1
To Slave Element (Dest)	VC300
Success	C102
Error	C103

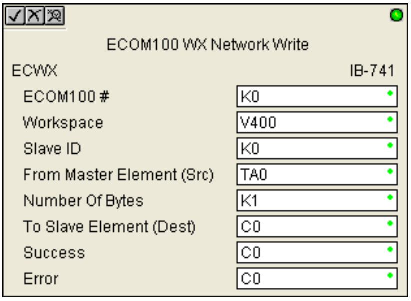
ECOM100 WX Network Write(ECWX) (IB-741)

DS	Used
HPP	N/A

ECOM100 WX Network Write performs the WX instruction with built-in interlocking with all other ECOM100 RX (ECRX) and ECOM100 WX (ECWX) IBoxes in your program to simplify communications networking. It will perform the WX on the specified ECOM100#'s network, which corresponds to a specific unique ECOM100 Configuration (ECOM100) IBox at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Whenever this IBox has power, it will write data from the master's V-Memory buffer to the specified slave starting with the given slave element, giving other ECOM100 RX and ECOM100 WX IBoxes on that ECOM100# network a chance to execute.



ECOM100 WX Network Write

ECWX IB-741

ECOM100 #	K0
Workspace	V400
Slave ID	K0
From Master Element (Src)	TA0
Number Of Bytes	K1
To Slave Element (Dest)	C0
Success	C0
Error	C0

For example, if you wish to read and write data continuously from 5 different slaves, you can have all of these ECRX and ECWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically.

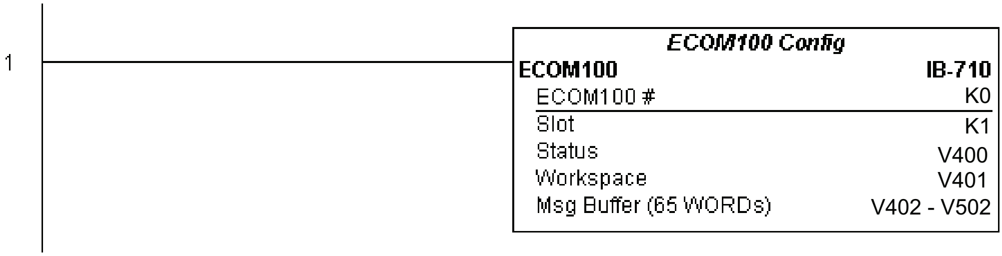
ECWX Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Slave ID: specifies the slave ECOM(100) PLC that will be targeted by the ECWX instruction
- From Master Element (Src): specifies the location in the master ECOM100 PLC where the data will be sourced from
- Number of Bytes: specifies the number of bytes to write to the slave ECOM(100) PLC
- To Slave Element (Dest): specifies the slave address the data will be written to
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Slave ID	K	K0-90
From Master Element (Src)	V	See DL06 V-memory map - Data Words
Number of Bytes	K	K1-128
To Slave Element (Dest)	X,Y,C,S,T,CT,GX,GY,V,P	See DL06 V-memory map
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

ECWX Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



(example continued on next page)

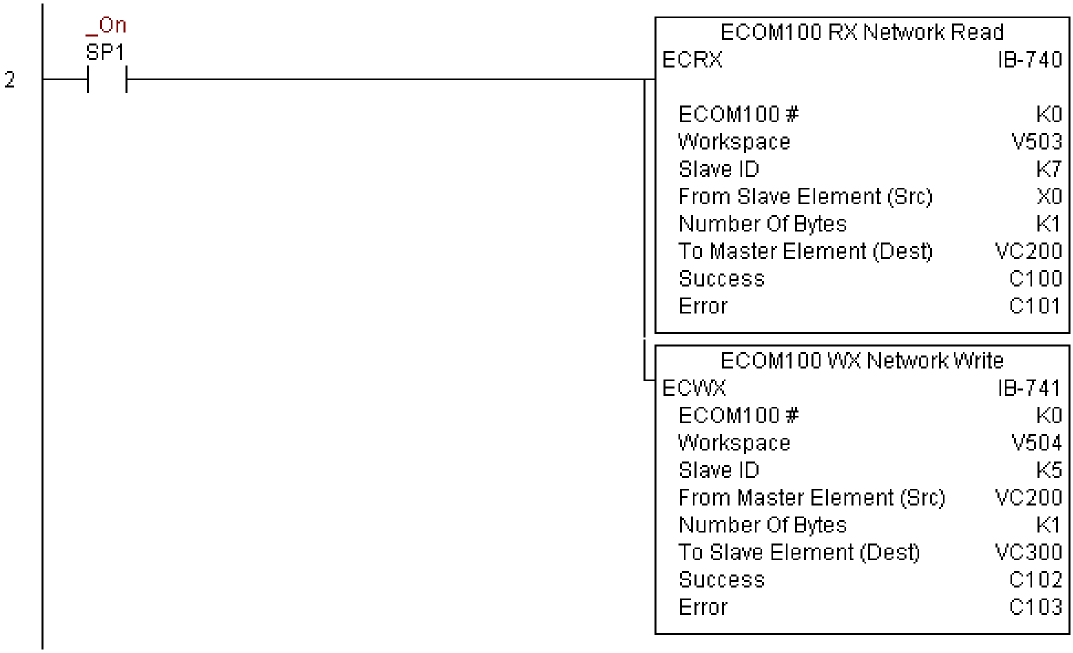
ECWX Example (cont’d)

Rung 2: Using ECOM100# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the ECRX and ECWX work with the ECOM100 Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP “busy bits” or “error bits”, or what slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the ECRX and ECWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending ECOM100 IBoxes below the ECWX, then the very next scan the ECRX would start its request again.

Using the ECRX and ECWX for all of your ECOM100 network reads and writes is the fastest the PLC can do networking. For local Serial Ports, DCM modules, or the original ECOM modules, use the NETCFG and NETRX/NETWX IBoxes.



ECOM100 RX Network Read
ECRX IB-740

ECOM100 #	K0
Workspace	V503
Slave ID	K7
From Slave Element (Src)	X0
Number Of Bytes	K1
To Master Element (Dest)	VC200
Success	C100
Error	C101

ECOM100 WX Network Write
ECWX IB-741

ECOM100 #	K0
Workspace	V504
Slave ID	K5
From Master Element (Src)	VC200
Number Of Bytes	K1
To Slave Element (Dest)	VC300
Success	C102
Error	C103

NETCFG Network Configuration (NETCFG) (IB-700)

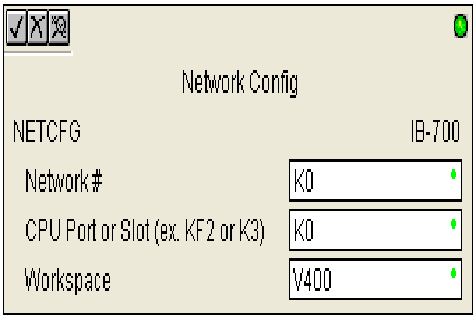
DS	Used
HPP	N/A

Network Config defines all the common information necessary for performing RX/WX Networking using the NETRX and NETWX IBox instructions via a local CPU serial port, DCM or ECOM module.

You must have the Network Config instruction at the top of your ladder/stage program with any other configuration IBoxes.

If you use more than one local serial port, DCM or ECOM in your PLC for RX/WX Networking, you must have a different Network Config instruction for EACH RX/WX network in your system that utilizes any NETRX/NETWX IBox instructions.

The Workspace parameter is an internal, private register used by the Network Config IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.



Network Config

NETCFG IB-700

Network# K0

CPU Port or Slot (ex. KF2 or K3) K0

Workspace V400

The 2nd parameter “CPU Port or Slot” is the same value as in the high byte of the first LD instruction if you were coding the RX or WX rung yourself. This value is CPU and port specific (check your PLC manual). Use KF2 for the DL06 CPU serial port 2. If using a DCM or ECOM module, use Kx, where x equals the slot where the module is installed.

Since this logic only executes on the first scan, this IBox cannot have any input logic.

NETCFG Parameters

- Network#: specifies a unique # for each ECOM(100) or DCM network to use
- CPU Port or Slot: specifies the CPU port number or slot number of DCM/ECOM(100) used
- Workspace: specifies a V-memory location that will be used by the instruction

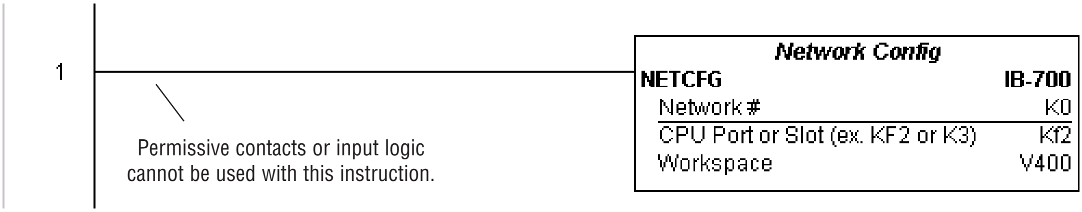
Parameter		DL06 Range
Network#	K	K0-255
CPU Port or Slot	K	K0-FF
Workspace	V	See DL06 V-memory map - Data Words

NETCFG Example

The Network Configuration IBox coordinates all of the interaction with other Network IBoxes (NETRX/NETWX). You must have a Network Configuration IBox for each serial port network, DCM module network, or original ECOM module network in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines Network# K0 to be for the local CPU serial port #2 (KF2). For local CPU serial ports or DCM/ECOM modules, use the same value you would use in the most significant byte of the first LD instruction in a normal RX/WX rung to reference the port or module. Any NETRX or NETWX IBoxes that need to reference this specific network would enter K0 for their Network# parameter.

The Workspace register is used to maintain state information about the port or module, along with proper sharing and interlocking with the other NETRX and NETWX IBoxes in the program. This V-memory register must not be used anywhere else in the entire program.



Permissive contacts or input logic cannot be used with this instruction.

Network Config

NETCFG IB-700

Network # K0

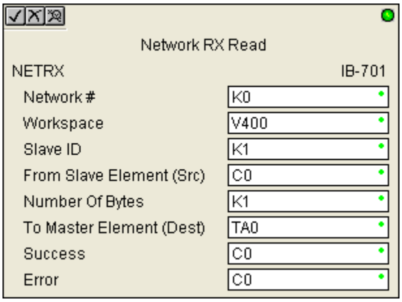
CPU Port or Slot (ex. KF2 or K3) Kf2

Workspace V400

Network RX Read (NETRX) (IB-701)

DS	Used
HPP	N/A

Network RX Read performs the RX instruction with built-in interlocking with all other Network RX (NETRX) and Network WX (NETWX) IBoxes in your program to simplify communications networking. It will perform the RX on the specified Network #, which corresponds to a specific unique Network Configuration (NETCFG) at the top of your program.



Network RX Read

NETRX IB-701

Network # K0

Workspace V400

Slave ID K1

From Slave Element (Src) C0

Number Of Bytes K1

To Master Element (Dest) TA0

Success C0

Error C0

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Whenever this IBox has power, it will read element data from the specified slave into the given destination V-Memory buffer, giving other Network RX and Network WX IBoxes on that Network # a chance to execute.

For example, if you wish to read and write data continuously from 5 different slaves, you can have all of these NETRX and NETWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically.

NETRX Parameters

- Network#: specifies the (CPU port's, DCM's, ECOM's) Network # defined by the NETCFG instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Slave ID: specifies the slave PLC that will be targeted by the NETRX instruction
- From Slave Element (Src): specifies the slave address of the data to be read
- Number of Bytes: specifies the number of bytes to read from the slave device
- To Master Element (Dest): specifies the location where the slave data will be placed in the master PLC
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed

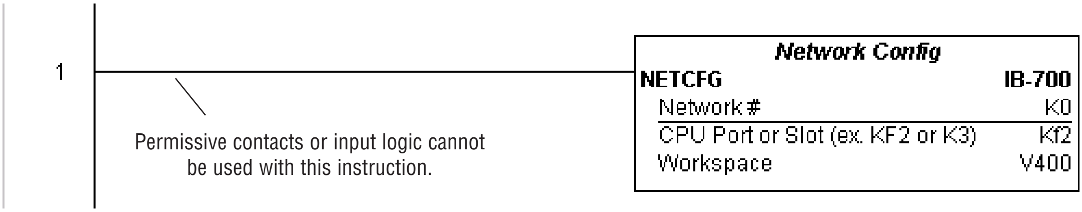
Parameter		DL06 Range
Network#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Slave ID	K, V	K0-90: See DL06 V-memory map
From Slave Element (Src)	X,Y,C,S,T,CT,GX,GY,V,P	See DL06 V-memory map
Number of Bytes	K	K1-128
To Master Element (Dest)	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

NETRX Example

Rung 1: The Network Configuration IBox coordinates all of the interaction with other Network IBoxes (NETRX/NETWX). You must have a Network Configuration IBox for each serial port network, DCM module network, or original ECOM module network in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines Network# K0 to be for the local CPU serial port #2 (KF2). For local CPU serial ports or DCM/ECOM modules, use the same value you would use in the most significant byte of the first LD instruction in a normal RX/WX rung to reference the port or module. Any NETRX or NETWX IBoxes that need to reference this specific network would enter K0 for their Network# parameter.

The Workspace register is used to maintain state information about the port or module, along with proper sharing and interlocking with the other NETRX and NETWX IBoxes in the program. This V-Memory register must not be used anywhere else in the entire program.



Network Config

NETCFG IB-700

Network # K0

CPU Port or Slot (ex. KF2 or K3) Kf2

Workspace V400

Permissive contacts or input logic cannot be used with this instruction.

(example continued on next page)

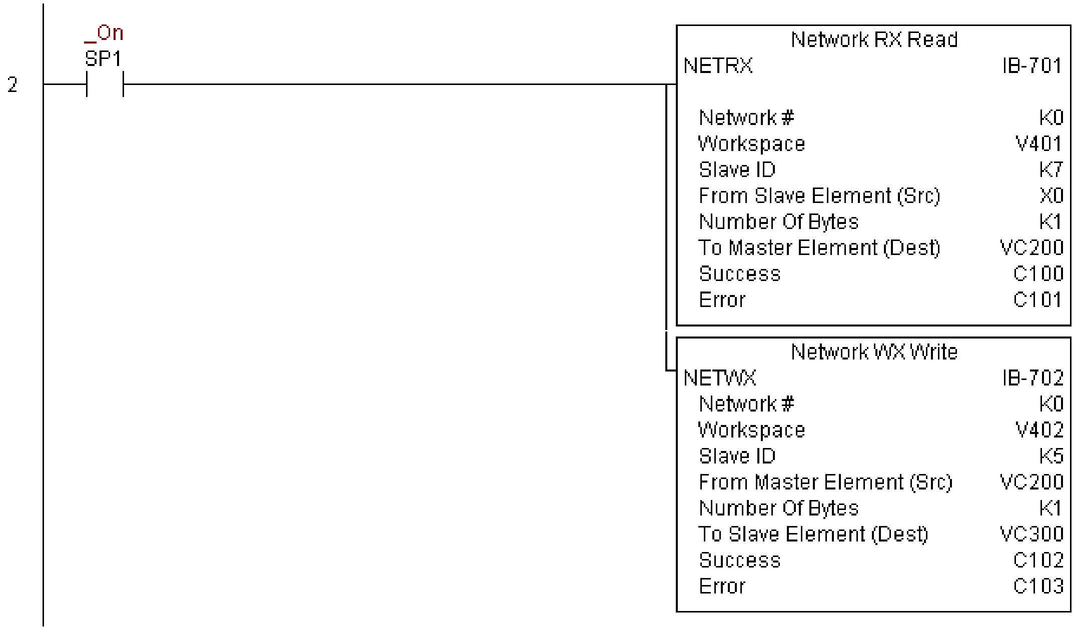
NETRX Example (cont'd)

Rung 2: Using Network# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the NETRX and NETWX work with the Network Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP "busy bits" or "error bits", or what port number or slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the NETRX and NETWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending NETRX or NETWX IBoxes below this IBox, then the very next scan the NETRX would start its request again.

Using the NETRX and NETWX for all of your serial port, DCM, or original ECOM network reads and writes is the fastest the PLC can do networking. For ECOM100 modules, use the ECOM100 and ECRX/ECWX IBoxes.



Network RX Read
NETRX IB-701

Network #	K0
Workspace	V401
Slave ID	K7
From Slave Element (Src)	X0
Number Of Bytes	K1
To Master Element (Dest)	VC200
Success	C100
Error	C101

Network WX Write
NETWX IB-702

Network #	K0
Workspace	V402
Slave ID	K5
From Master Element (Src)	VC200
Number Of Bytes	K1
To Slave Element (Dest)	VC300
Success	C102
Error	C103

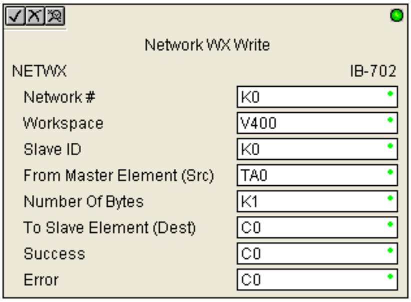
Network WX Write (NETWX) (IB-702)

DS	Used
HPP	N/A

Network WX Write performs the WX instruction with built-in interlocking with all other Network RX (NETRX) and Network WX (NETWX) IBoxes in your program to simplify communications networking. It will perform the WX on the specified Network #, which corresponds to a specific unique Network Configuration (NETCFG) at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Whenever this IBox has power, it will write data from the master's V-Memory buffer to the specified slave starting with the given slave element, giving other Network RX and Network WX IBoxes on that Network # a chance to execute.



Network WX Write

NETWX IB-702

Network #	K0
Workspace	V400
Slave ID	K0
From Master Element (Src)	TA0
Number Of Bytes	K1
To Slave Element (Dest)	C0
Success	C0
Error	C0

For example, if you wish to read and write data continuously from 5 different slaves, you can have all of these NETRX and NETWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically.

NETWX Parameters

- Network#: specifies the (CPU port's, DCM's, ECOM's) Network # defined by the NETCFG instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Slave ID: specifies the slave PLC that will be targeted by the NETWX instruction
- From Master Element (Src): specifies the location in the master PLC where the data will be sourced from
- Number of Bytes: specifies the number of bytes to write to the slave PLC
- To Slave Element (Dest): specifies the slave address the data will be written to
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed

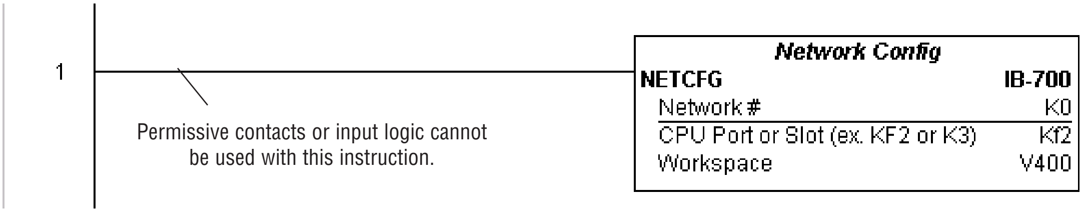
Parameter		DL06 Range
Network#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Slave ID	K,V	K0-90: See DL06 V-memory map
From Master Element (Src)	V	See DL06 V-memory map - Data Words
Number of Bytes	K	K1-128
To Slave Element (Dest)	X,Y,C,S,T,CT,GX,GY,V,P	See DL06 V-memory map
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

NETWX Example

Rung 1: The Network Configuration IBox coordinates all of the interaction with other Network IBoxes (NETRX/NETWX). You must have a Network Configuration IBox for each serial port network, DCM module network, or original ECOM module network in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines Network# K0 to be for the local CPU serial port #2 (KF2). For local CPU serial ports or DCM/ECOM modules, use the same value you would use in the most significant byte of the first LD instruction in a normal RX/WX rung to reference the port or module. Any NETRX or NETWX IBoxes that need to reference this specific network would enter K0 for their Network# parameter.

The Workspace register is used to maintain state information about the port or module, along with proper sharing and interlocking with the other NETRX and NETWX IBoxes in the program. This V-Memory register must not be used anywhere else in the entire program.



Network # K0
CPU Port or Slot (ex. KF2 or K3) Kf2
Workspace V400

Permissive contacts or input logic cannot be used with this instruction.

(example continued on next page)

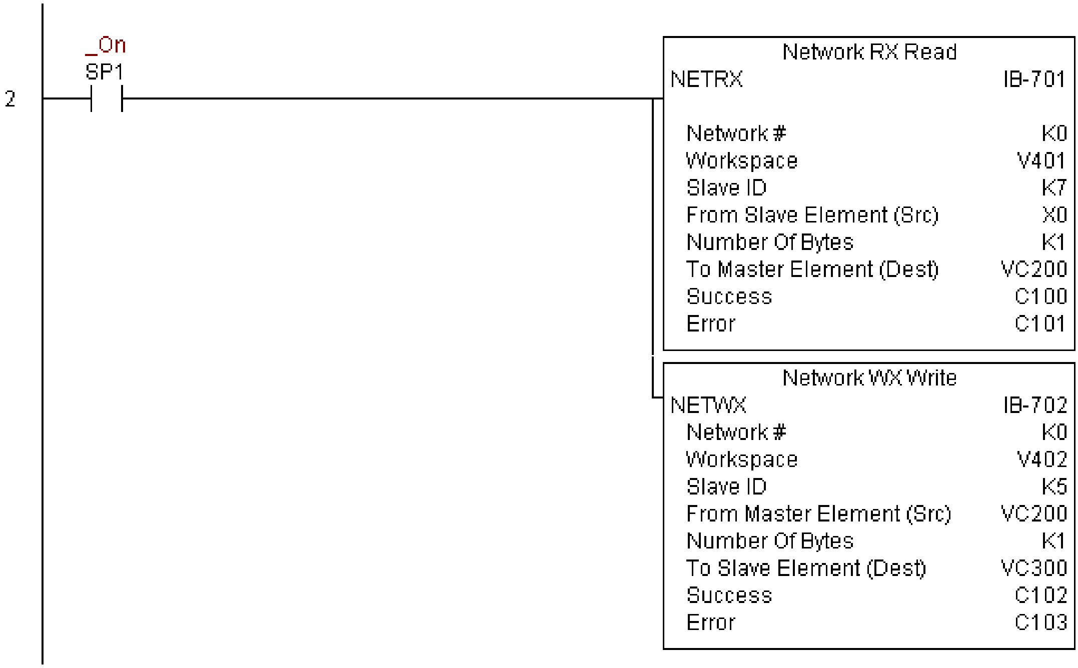
NETWX Example (cont'd)

Rung 2: Using Network# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the NETRX and NETWX work with the Network Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP "busy bits" or "error bits", or what port number or slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the NETRX and NETWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending NETRX or NETWX IBoxes below this IBox, then the very next scan the NETRX would start its request again.

Using the NETRX and NETWX for all of your serial port, DCM, or original ECOM network reads and writes is the fastest the PLC can do networking. For ECOM100 modules, use the ECOM100 and ECRX/ECWX IBoxes.



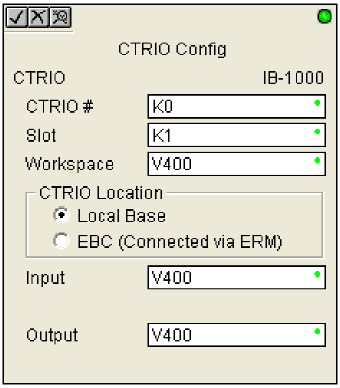
Network RX Read
NETRX IB-701
Network # K0
Workspace V401
Slave ID K7
From Slave Element (Src) X0
Number Of Bytes K1
To Master Element (Dest) VC200
Success C100
Error C101

Network WX Write
NETWX IB-702
Network # K0
Workspace V402
Slave ID K5
From Master Element (Src) VC200
Number Of Bytes K1
To Slave Element (Dest) VC300
Success C102
Error C103

CTRIO Configuration (CTRIO) (IB-1000)

DS	Used
HPP	N/A

CTRIO Config defines all the common information for one specific CTRIO module which is used by the other CTRIO IBox instructions (for example, CTRLDPR - CTRIO Load Profile, CTREDRL - CTRIO Edit and Reload Preset Table, CTRRTLM - CTRIO Run to Limit Mode, ...).


CTRIO Config

CTRIO IB-1000

CTRIO # K0

Slot K1

Workspace V400

CTRIO Location

● Local Base

○ EBC (Connected via ERM)

Input V400

Output V400

The Input/Output parameters for this instruction can be copied directly from the CTRIO Workbench configuration for this CTRIO module. Since the behavior is slightly different when the CTRIO module is in an EBC Base via an ERM, you must specify whether the CTRIO module is in a local base or in an EBC base. The DL06 PLC only supports local base operation at this time.

You must have the CTRIO Config IBox at the top of your ladder/stage program along with any other configuration IBoxes.

If you have more than one CTRIO in your PLC, you must have a different CTRIO Config IBox for EACH CTRIO module in your system that utilizes any CTRIO IBox instructions. Each CTRIO Config IBox must have a UNIQUE CTRIO# value. This is how the CTRIO IBoxes differentiate between the different CTRIO modules in your system.

The Workspace parameter is an internal, private register used by the CTRIO Config IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Since this logic only executes on the first scan, this IBox cannot have any input logic.

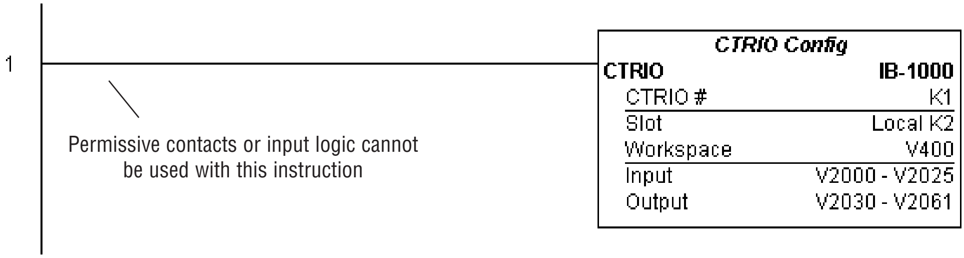
CTRIO Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number
- Slot: specifies which PLC option slot the CTRIO module occupies
- Workspace: specifies a V-memory location that will be used by the instruction
- CTRIO Location: specifies where the module is located (local base only for DL06)
- Input: This needs to be set to the same V-memory register as is specified in CTRIO Workbench as 'Starting V address for inputs' for this unique CTRIO.
- Output: This needs to be set to the same V-memory register as is specified in CTRIO Workbench as 'Starting V address for outputs' for this unique CTRIO.

Parameter		DL06 Range
CTRIO#	K	K0-255
Slot	K	K1-4
Workspace	V	See DL06 V-memory map - Data Words
Input	V	See DL06 V-memory map - Data Words
Output	V	See DL06 V-memory map - Data Words

CTRIO Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



CTRIO Add Entry to End of Preset Table (CTRADPT) (IB-1005)

DS	Used
HPP	N/A

CTRIO Add Entry to End of Preset Table, on a leading edge transition to this IBox, will append an entry to the end of a memory based Preset Table on a specific CTRIO Output resource. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

Entry Type:

K0: Set

K1: Reset

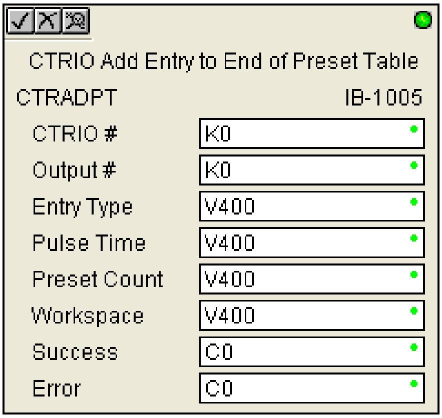
K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.



CTRIO Add Entry to End of Preset Table

CTRADPT IB-1005

CTRIO #	K0
Output #	K0
Entry Type	V400
Pulse Time	V400
Preset Count	V400
Workspace	V400
Success	C0
Error	C0

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

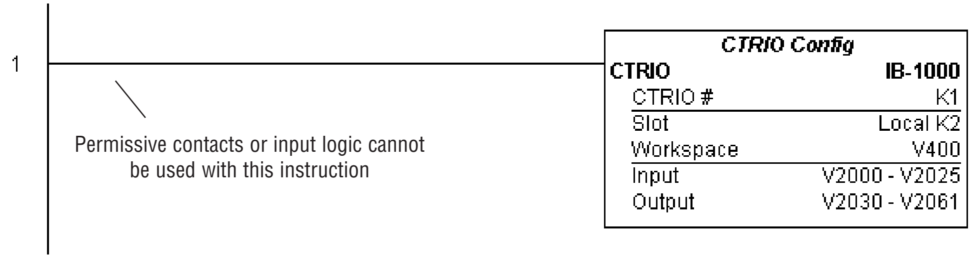
CTRAPT Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config)
- Output#: specifies a CTRIO output to be used by the instruction
- Entry Type: specifies the Entry Type to be added to the end of a Preset Table
- Pulse Time: specifies a pulse time for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Entry Type	V,K	K0-5; See DL06 V-memory map - Data Words
Pulse Time	V,K	K0-65535; See DL06 V-memory map - Data Words
Preset Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRADPT Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.

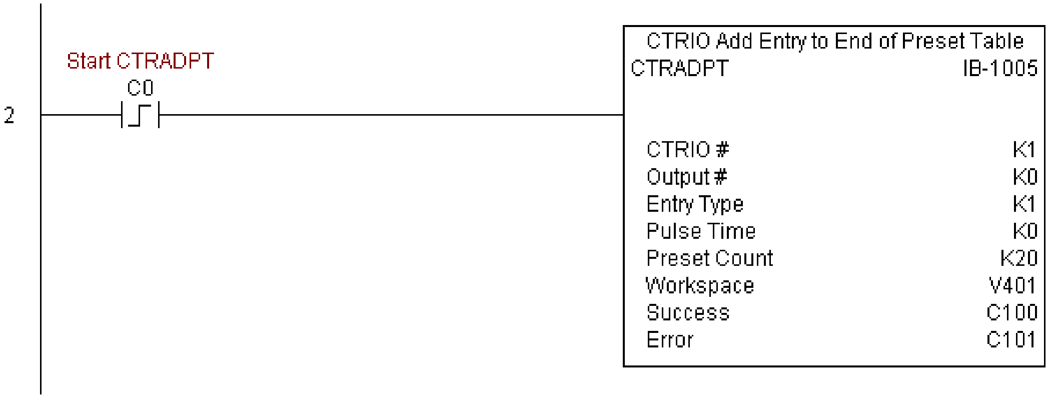

1
Permissive contacts or input logic cannot
be used with this instruction

CTRIO Config
CTRIO IB-1000
CTRIO # K1
Slot Local K2
Workspace V400
Input V2000 - V2025
Output V2030 - V2061

Rung 2: This rung is a sample method for enabling the CTRADPT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRADPT instruction to add a new preset to the preset table for output #0 on the CTRIO in slot 2. The new preset will be a command to RESET (entry type K1=reset), pulse time is left at zero as the reset type does not use this, and the count at which it will reset will be 20.

Operating procedure for this example code is to load the CTRADPT_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on and stay on for all counts past 10. Now reset the counter with C1, enable C0 to execute CTRADPT command to add a reset for output #0 at a count of 20, turn on C2 to enable output #0, then turn encoder to value of 10+ (output #0 should turn on) and then continue on to count of 20+ (output #0 should turn off).

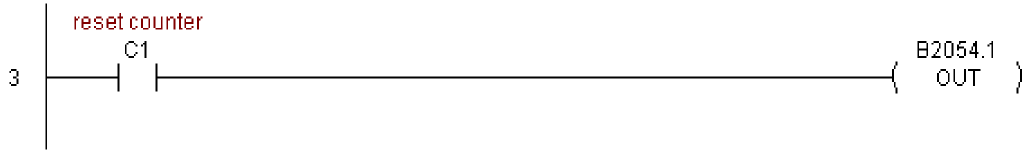


CTRIO Add Entry to End of Preset Table
CTRADPT IB-1005
CTRIO # K1
Output # K0
Entry Type K1
Pulse Time K0
Preset Count K20
Workspace V401
Success C100
Error C101

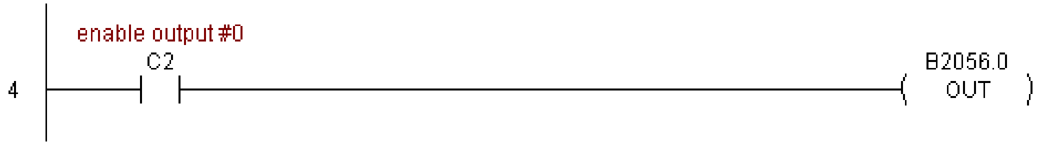
(example continued on next page)

CTRADPT Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



Rung 4: This rung allows the operator to enable output #0 from the ladder code.

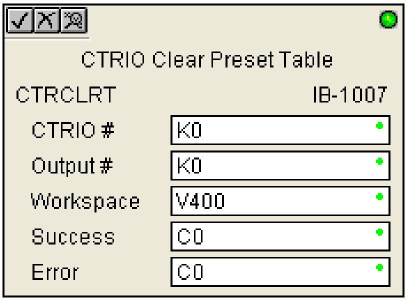


CTRIO Clear Preset Table (CTRCLRT) (IB-1007)

DS	Used
HPP	N/A

CTRIO Clear Preset Table will clear the RAM based Preset Table on a leading edge transition to this IBox. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.


CTRIO Clear Preset Table

CTRCLRT IB-1007

CTRIO # K0

Output # K0

Workspace V400

Success C0

Error C0

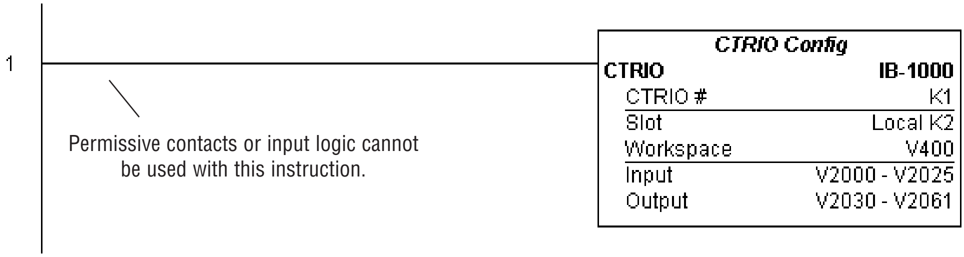
CTRCLRT Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config)
- Output#: specifies a CTRIO output to be used by the instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRCLRT Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



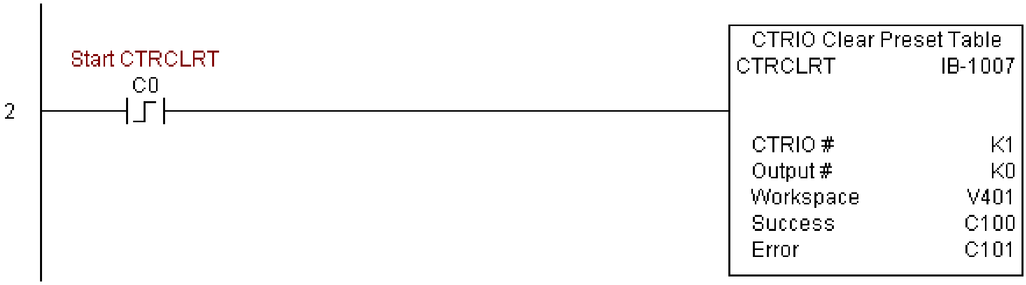
Permissive contacts or input logic cannot be used with this instruction.

CTRIO Config
CTRIO IB-1000
CTRIO # K1
Slot Local K2
Workspace V400
Input V2000 - V2025
Output V2030 - V2061

Rung 2: This rung is a sample method for enabling the CTRCLRT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRCLRT instruction to clear the preset table for output #0 on the CTRIO in slot 2.

Operating procedure for this example code is to load the CTRCLRT_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on and stay on until a count of 20 is reached, where it will turn off. Now reset the counter with C1, enable C0 to execute CTRCLRT command to clear the preset table, turn on C2 to enable output #0, then turn encoder to value of 10+ (output #0 should NOT turn on).

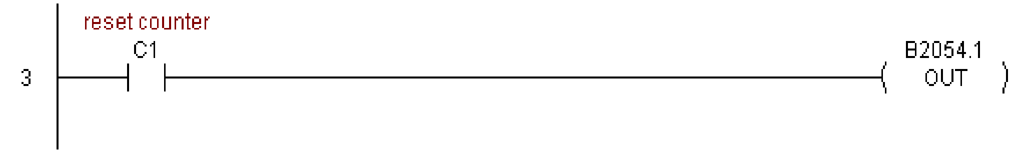


CTRIO Clear Preset Table
CTRCLRT IB-1007
CTRIO # K1
Output # K0
Workspace V401
Success C100
Error C101

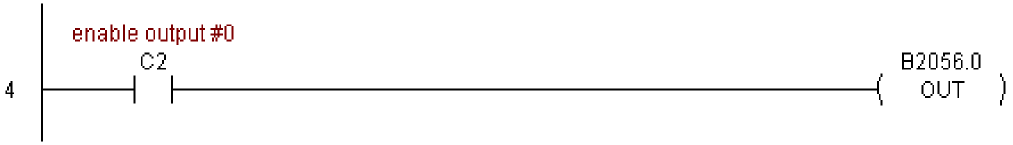
(example continued on next page)

CTRCLRT Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



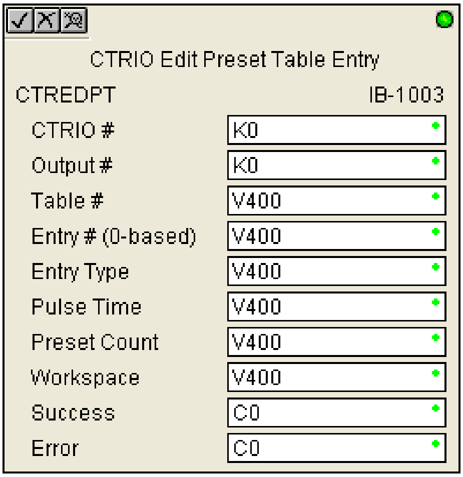
Rung 4: This rung allows the operator to enable output #0 from the ladder code.



CTRIO Edit Preset Table Entry (CTREDPT) (IB-1003)

DS	Used
HPP	N/A

CTRIO Edit Preset Table Entry, on a leading edge transition to this IBox, will edit a single entry in a Preset Table on a specific CTRIO Output resource. This IBox is good if you are editing more than one entry in a file at a time. If you wish to do just one edit and then reload the table immediately, see the CTRIO Edit and Reload Preset Table Entry (CTREDRL) IBox. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.



CTRIO Edit Preset Table Entry

CTREDPT IB-1003

CTRIO #	K0
Output #	K0
Table #	V400
Entry # (0-based)	V400
Entry Type	V400
Pulse Time	V400
Preset Count	V400
Workspace	V400
Success	C0
Error	C0

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

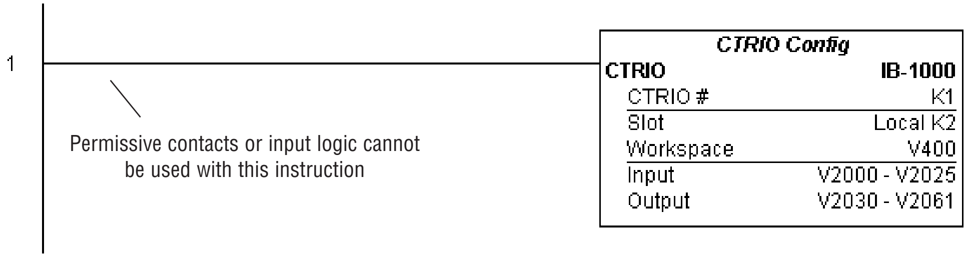
CTREDPT Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Table#: specifies the Table number of which an Entry is to be edited
- Entry#: specifies the Entry location in the Preset Table to be edited
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Table#	V,K	K0-255; See DL06 V-memory map - Data Words
Entry#	V,K	K0-255; See DL06 V-memory map - Data Words
Entry Type	V,K	K0-5; See DL06 V-memory map - Data Words
Pulse Time	V,K	K0-65535; See DL06 V-memory map - Data Words
Preset Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTREDPT Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



Permissive contacts or input logic cannot be used with this instruction

CTRIO Config

CTRIO **IB-1000**

CTRIO #	K1
Slot	Local K2
Workspace	V400
Input	V2000 - V2025
Output	V2030 - V2061

(example continued on next page)

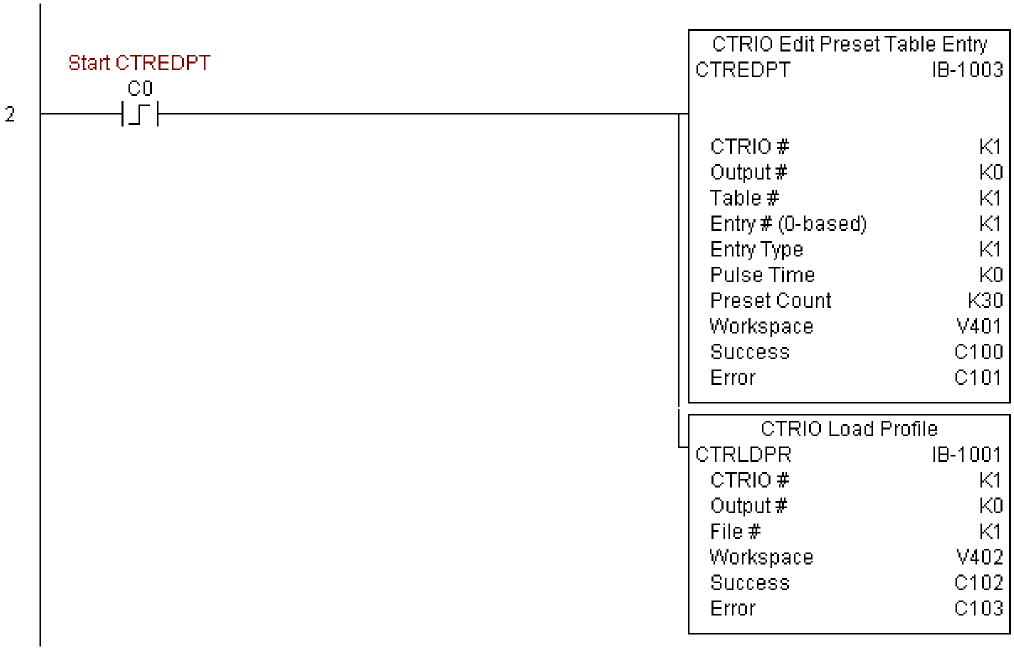
CTREDPT Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTREDPT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTREDPT instruction to change the second preset from a reset at a count of 20 to a reset at a count of 30 for output #0 on the CTRIO in slot 2.

Operating procedure for this example code is to load the CTREDPT_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on and stay on until a count of 20 is reached, where it will turn off. Now reset the counter with C1, enable C0 to execute CTREDPT command to change the second preset, turn on C2 to enable output #0, then turn encoder to value of 10+ (output #0 should turn on) and then continue past a count of 30 (output #0 should turn off).

Note that we must also reload the profile after changing the preset(s), this is why the CTRLDPR command follows the CTREDPT command in this example.

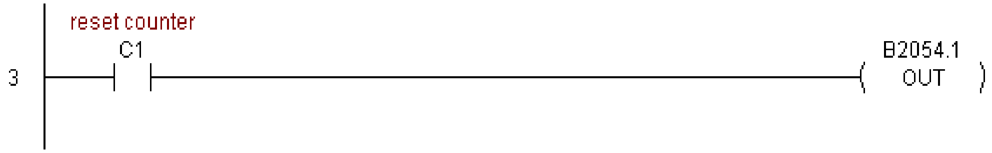


```
CTRIO Edit Preset Table Entry
CTREDPT                 IB-1003

  CTRIO #                    K1
  Output #                   K0
  Table #                    K1
  Entry # (0-based)          K1
  Entry Type                 K1
  Pulse Time                 K0
  Preset Count              K30
  Workspace                V401
  Success                  C100
  Error                    C101

CTRIO Load Profile
CTRLDPR                 IB-1001
  CTRIO #                    K1
  Output #                   K0
  File #                     K1
  Workspace                V402
  Success                  C102
  Error                    C103
```

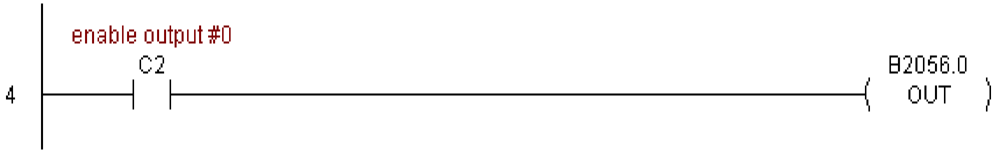
(example continued on next page)

CTREDPT Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



Rung 4: This rung allows the operator to enable output #0 from the ladder code.



CTRIO Edit Preset Table Entry and Reload (CTREDRL) (IB-1002)

DS	Used
HPP	N/A

CTRIO Edit Preset Table Entry and Reload, on a leading edge transition to this IBox, will perform this dual operation to a CTRIO Output resource in one CTRIO command. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

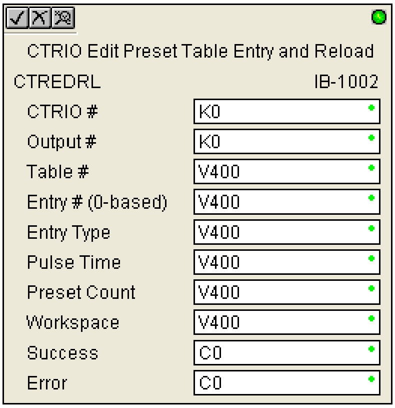
K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



CTRIO Edit Preset Table Entry and Reload

CTREDRL IB-1002

CTRIO #	K0
Output #	K0
Table #	V400
Entry # (0-based)	V400
Entry Type	V400
Pulse Time	V400
Preset Count	V400
Workspace	V400
Success	C0
Error	C0

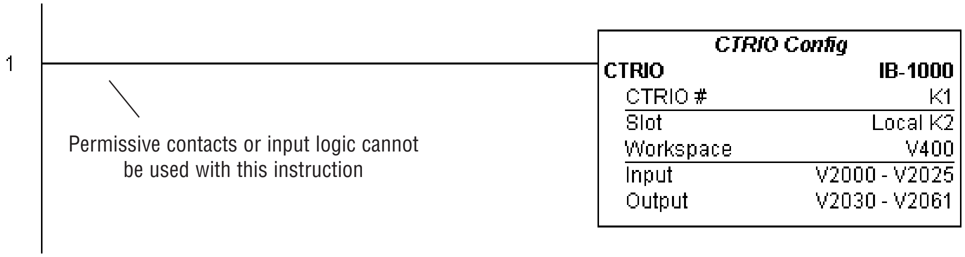
CTREDRL Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Table#: specifies the Table number of which an Entry is to be edited
- Entry#: specifies the Entry location in the Preset Table to be edited
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Table#	V,K	K0-255; See DL06 V-memory map - Data Words
Entry#	V,K	K0-255; See DL06 V-memory map - Data Words
Entry Type	V,K	K0-5; See DL06 V-memory map - Data Words
Pulse Time	V,K	K0-65535; See DL06 V-memory map - Data Words
Preset Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTREDRL Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.


1
Permissive contacts or input logic cannot be used with this instruction

CTRIO Config	
CTRIO	IB-1000
CTRIO #	K1
Slot	Local K2
Workspace	V400
Input	V2000 - V2025
Output	V2030 - V2061

(example continued on next page)

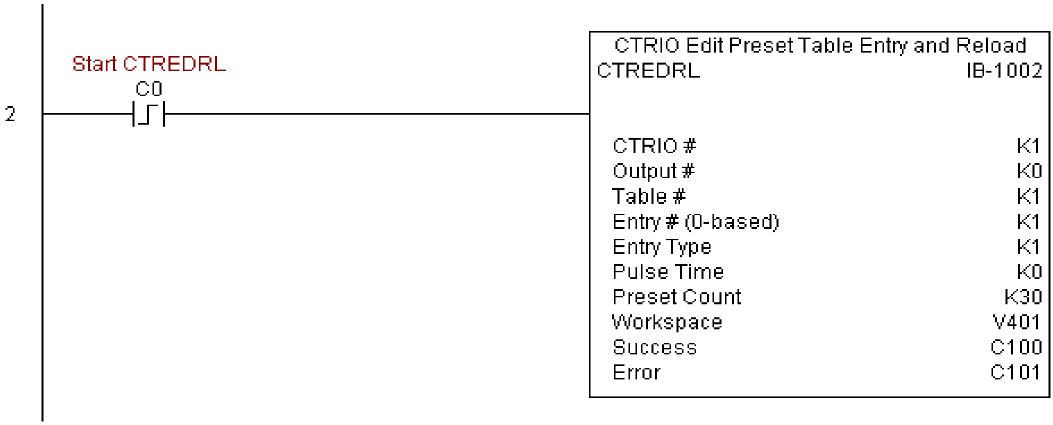
CTREDRL Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTREDRL command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTREDRL instruction to change the second preset in file 1 from a reset at a value of 20 to a reset at a value of 30.

Operating procedure for this example code is to load the CTREDRL_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on, continue to a count above 20 and the output #0 light will turn off. Now reset the counter with C1, enable C0 to execute CTREDRL command to change the second preset count value to 30, then turn encoder to value of 10+ (output #0 should turn on) and continue on to a value of 30+ and the output #0 light will turn off.

Note that it is not necessary to reload this file separately, however, the command can only change one value at a time.

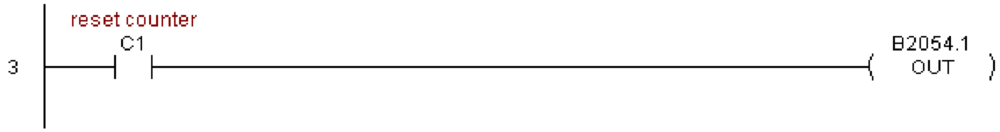


```
CTRIO Edit Preset Table Entry and Reload
CTREDRL                          IB-1002

CTRIO #                               K1
Output #                              K0
Table #                               K1
Entry # (0-based)                     K1
Entry Type                            K1
Pulse Time                            K0
Preset Count                         K30
Workspace                           V401
Success                             C100
Error                               C101
```

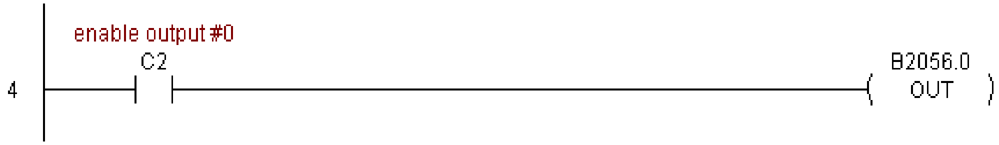
(example continued on next page)

CTREDRL Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



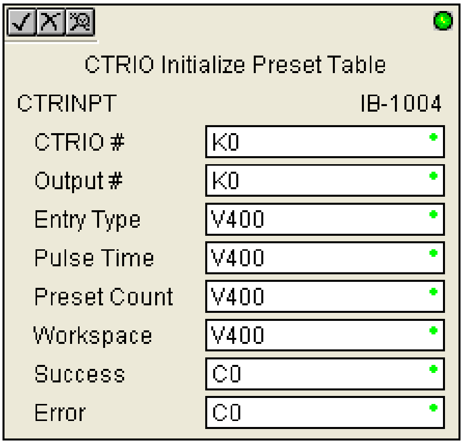
Rung 4: This rung allows the operator to enable output #0 from the ladder code.



CTRIO Initialize Preset Table (CTRINPT) (IB-1004)

DS	Used
HPP	N/A

CTRIO Initialize Preset Table, on a leading edge transition to this IBox, will create a single entry Preset Table in memory but not as a file, on a specific CTRIO Output resource. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.


CTRIO Initialize Preset Table

CTRINPT IB-1004

CTRIO #	K0
Output #	K0
Entry Type	V400
Pulse Time	V400
Preset Count	V400
Workspace	V400
Success	C0
Error	C0

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

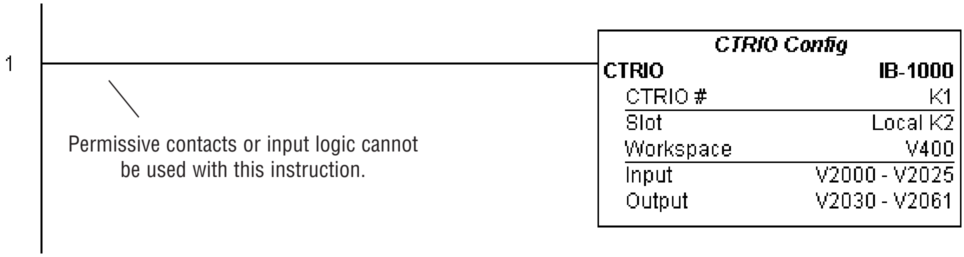
CTRINPT Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Entry Type	V,K	K0-5; See DL06 V-memory map - Data Words
Pulse Time	V,K	K0-65535; See DL06 V-memory map - Data Words
Preset Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRINPT Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



Permissive contacts or input logic cannot be used with this instruction.

CTRIO Config

CTRIO	IB-1000
CTRIO #	K1
Slot	Local K2
Workspace	V400
Input	V2000 - V2025
Output	V2030 - V2061

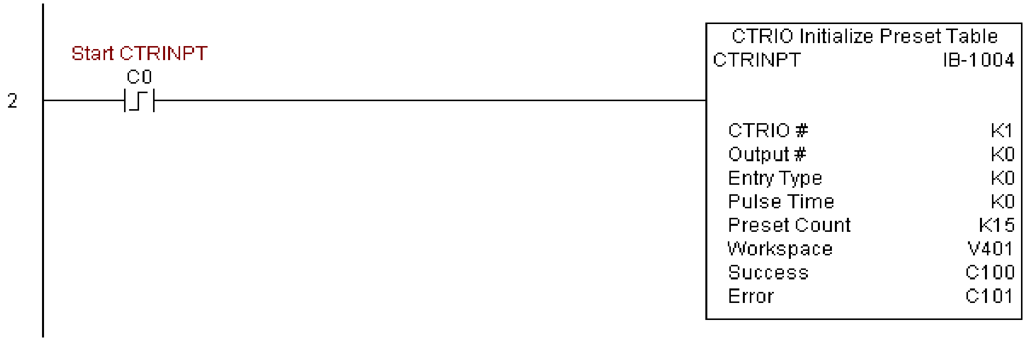
(example continued on next page)

CTRINPT Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTRINPT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRINPT instruction to create a single entry preset table, but not as a file, and use it for the output #0. In this case the single preset will be a set at a count of 15 for output #0.

Operating procedure for this example code is to load the CTRINPT_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 15 and output #0 light will not come on. Now reset the counter with C1, enable C0 to execute CTRINPT command to create a single preset table with a preset to set output #0 at a count of 15, then turn encoder to value of 15+ (output #0 should turn on).



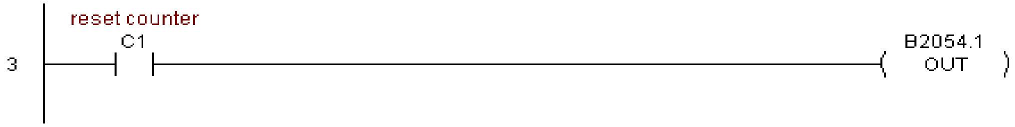
CTRIO Initialize Preset Table
CTRINPT IB-1004

CTRIO #	K1
Output #	K0
Entry Type	K0
Pulse Time	K0
Preset Count	K15
Workspace	V401
Success	C100
Error	C101

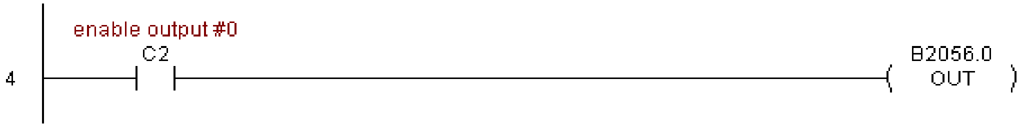
(example continued on next page)

CTRINPT Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



Rung 4: This rung allows the operator to enable output #0 from the ladder code.



CTRIO Initialize Preset Table (CTRINTR) (IB-1010)

DS	Used
HPP	N/A

CTRIO Initialize Preset Table, on a leading edge transition to this IBox, will create a single entry Preset Table in memory but not as a file, on a specific CTRIO Output resource.This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

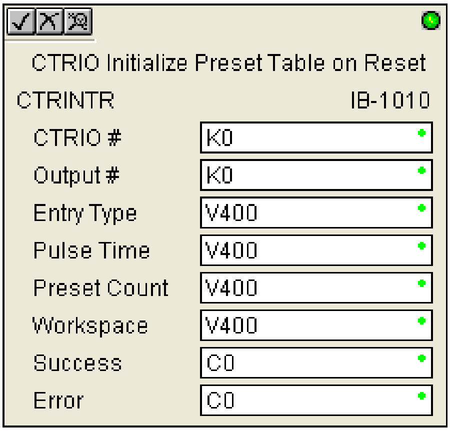
K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



CTRIO Initialize Preset Table on Reset

CTRINTR IB-1010

CTRIO #	K0
Output #	K0
Entry Type	V400
Pulse Time	V400
Preset Count	V400
Workspace	V400
Success	C0
Error	C0

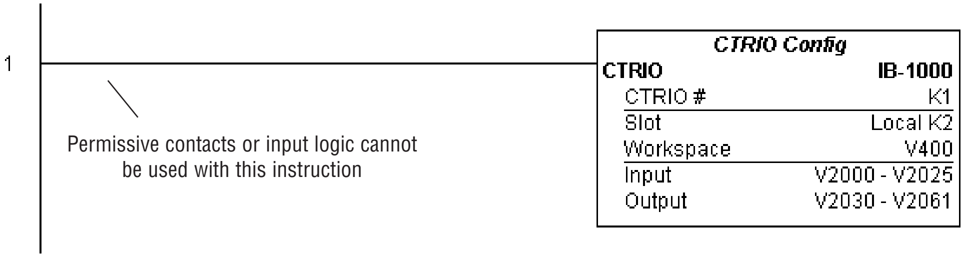
CTRINTR Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Entry Type	V,K	K0-5; See DL06 V-memory map - Data Words
Pulse Time	V,K	K0-65535; See DL06 V-memory map - Data Words
Preset Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRINTR Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



Permissive contacts or input logic cannot be used with this instruction

CTRIO Config

CTRIO	**IB-1000**
CTRIO #	K1
Slot	Local K2
Workspace	V400
Input	V2000 - V2025
Output	V2030 - V2061

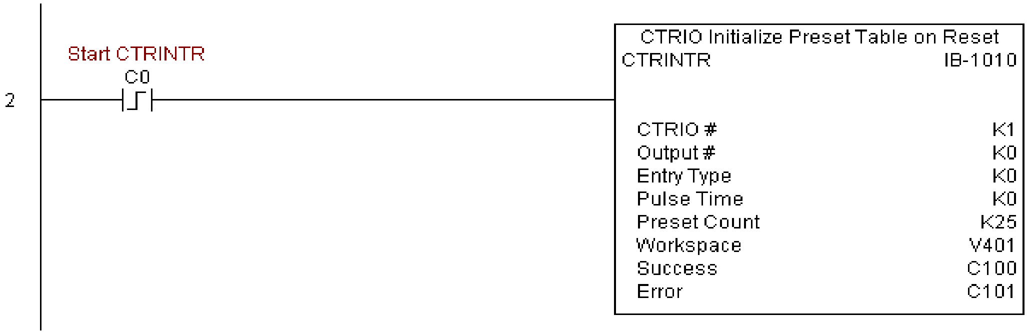
(example continued on next page)

CTRINTR Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTRINTR command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRINTR instruction to create a single entry preset table, but not as a file, and use it for output #0, the new preset will be loaded when the current count is reset. In this case the single preset will be a set at a count of 25 for output #0.

Operating procedure for this example code is to load the CTRINTR_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on. Now turn on C0 to execute the CTRINTR command, reset the counter with C1, then turn encoder to value of 25+ (output #0 should turn on).



CTRIO Initialize Preset Table on Reset

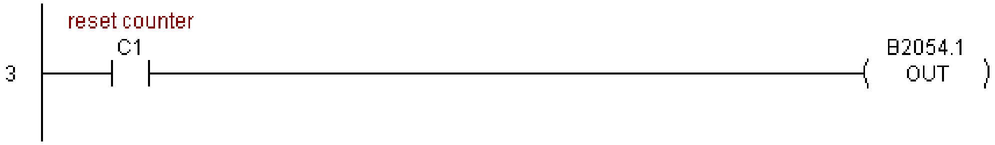
CTRINTR IB-1010

CTRIO #	K1
Output #	K0
Entry Type	K0
Pulse Time	K0
Preset Count	K25
Workspace	V401
Success	C100
Error	C101

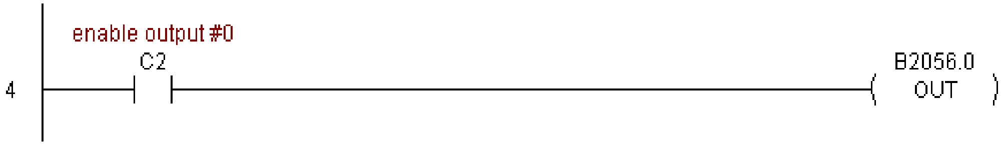
(example continued on next page)

CTRINTR Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



Rung 4: This rung allows the operator to enable output #0 from the ladder code.

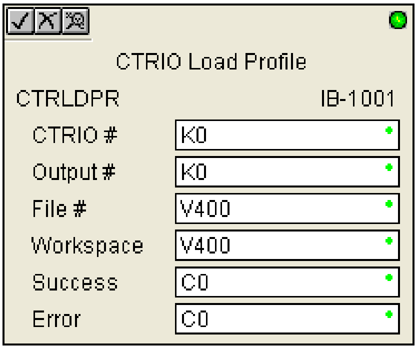


CTRIO Load Profile (CTRLDPR) (IB-1001)

DS	Used
HPP	N/A

CTRIO Load Profile loads a CTRIO Profile File to a CTRIO Output resource on a leading edge transition to this IBox. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



CTRIO Load Profile

CTRLDPR IB-1001

CTRIO # K0

Output # K0

File # V400

Workspace V400

Success C0

Error C0

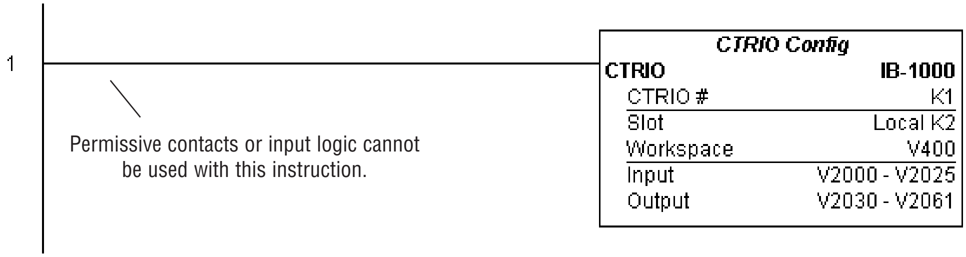
CTRLDPR Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config)
- Output#: specifies a CTRIO output to be used by the instruction
- File#: specifies a CTRIO profile File number to be loaded
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
File#	V,K	K0-255; See DL06 V-memory map - Data Words
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

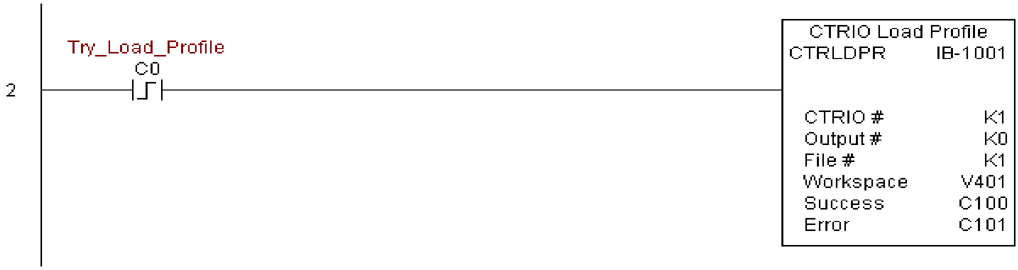
CTRLDPR Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.


1
Permissive contacts or input logic cannot be used with this instruction.

CTRIO Config
CTRIO IB-1000
CTRIO # K1
Slot Local K2
Workspace V400
Input V2000 - V2025
Output V2030 - V2061

Rung 2: This CTRIO Load Profile IBox will load File #1 into the working memory of Output 0 in CTRIO #1. This example program requires that you load CTRLDPR_IBox.cwb into your Hx-CTRIO(2) module.

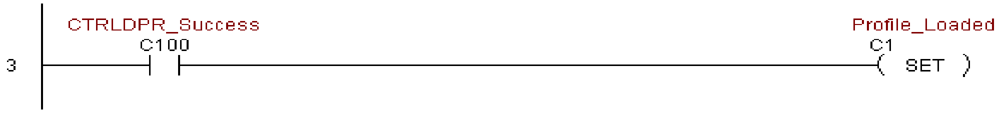


CTRIO Load Profile
CTRLDPR IB-1001
CTRIO # K1
Output # K0
File # K1
Workspace V401
Success C100
Error C101

(example continued on next page)

CTRLDPR Example (cont’d)

Rung 3: If the file is successfully loaded, set Profile_Loaded.



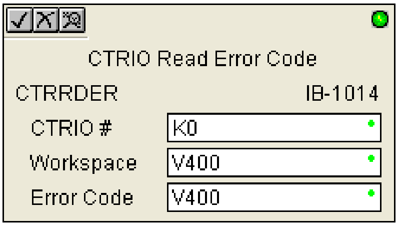
CTRIO Read Error (CTRRDER) (IB-1014)

DS	Used
HPP	N/A

CTRIO Read Error Code will get the decimal error code value from the CTRIO module (listed below) and place it into the given Error Code register, on a leading edge transition to the IBox

Since the Error Code in the CTRIO is only maintained until another CTRIO command is given, you must use this instruction immediately after the CTRIO IBox that reports an error via its Error bit parameter.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



CTRIO Read Error Code

CTRRDER IB-1014

CTRIO # K0

Workspace V400

Error Code V400

Error Codes:

0: No Error

100: Specified command code is unknown or unsupported

101: File number not found in the file system

102: File type is incorrect for specified output function

103: Profile type is unknown

104: Specified input is not configured as a limit on this output

105: Specified limit input edge is out of range

106: Specified input function is unconfigured or invalid

107: Specified input function number is out of range

108: Specified preset function is invalid

109: Preset table is full

110: Specified Table entry is out of range

111: Specified register number is out of range

112: Specified register is an unconfigured input or output

2001: Error reading Error Code - cannot access CTRIO via ERM

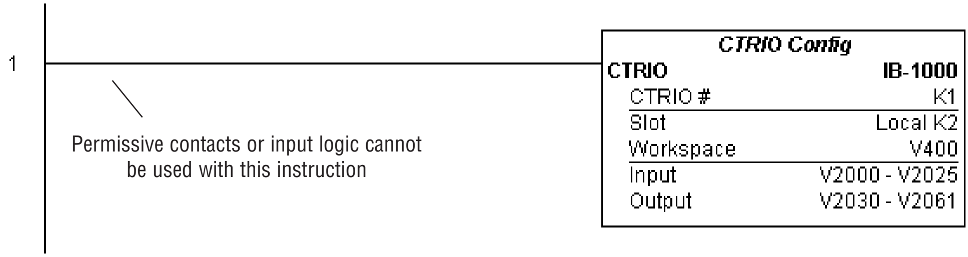
CTRRDER Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config)
- Workspace: specifies a V-memory location that will be used by the instruction
- Error Code: specifies the location where the Error Code will be written

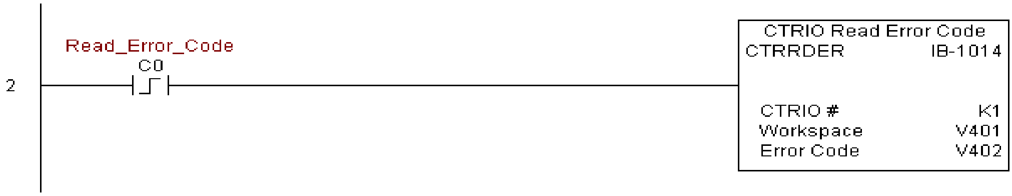
Parameter		DL06 Range
CTRIO#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Error Code	V	See DL06 V-memory map - Data Words

CTRRDER Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



Rung 2: This CTRIO Read Error Code IBox will read the Extended Error information from CTRIO #1. This example program requires that you load CTRRDER_IBox.cwb into your Hx-CTRIO(2) module.



CTRIO Run to Limit Mode (CTRRTLM) (IB-1011)

DS	Used
HPP	N/A

CTRIO Run To Limit Mode, on a leading edge transition to this IBox, loads the Run to Limit command and given parameters on a specific Output resource. The CTRIO's Input(s) must be configured as Limit(s) for this function to work.

Valid Hexadecimal Limit Values:

K00 - Rising Edge of Ch1/C

K10 - Falling Edge of Ch1/C

K20 - Both Edges of Ch1/C

K01 - Rising Edge of Ch1/D

K11 - Falling Edge of Ch1/D

K21 - Both Edges of Ch1/D

K02 - Rising Edge of Ch2/C

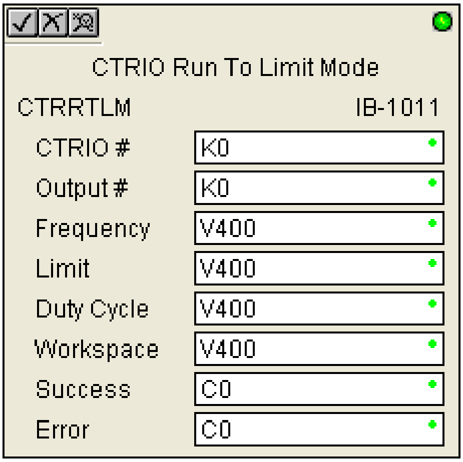
K12 - Falling Edge of Ch2/C

K22 - Both Edges of Ch2/C

K03 - Rising Edge of Ch2/D

K13 - Falling Edge of Ch2/D

K23 - Both Edges of Ch2/D



CTRIO Run To Limit Mode

CTRRTLM IB-1011

CTRIO #	K0
Output #	K0
Frequency	V400
Limit	V400
Duty Cycle	V400
Workspace	V400
Success	C0
Error	C0

This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

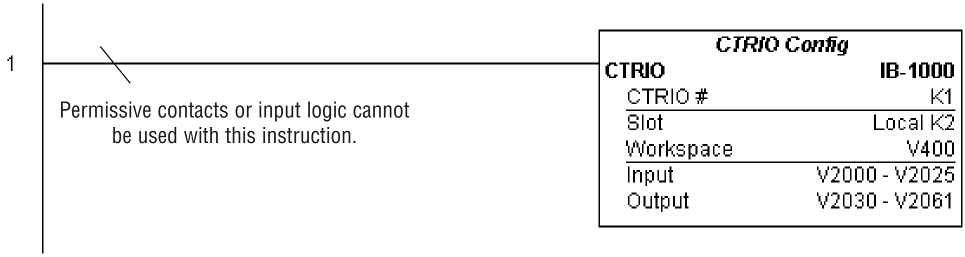
CTRRTLM Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Frequency: specifies the output pulse rate (H0-CTRIO: 20Hz - 25KHz / H0-CTRIO2: 20Hz - 250 KHz)
- Limit: the CTRIO's Input(s) must be configured as Limit(s) for this function to operate
- Duty Cycle: specifies the % of on time versus off time. This is a hex number. Default of 0 is 50%, also entering 50 will yield 50%. 50% duty cycle is defined as on half the time and off half the time
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

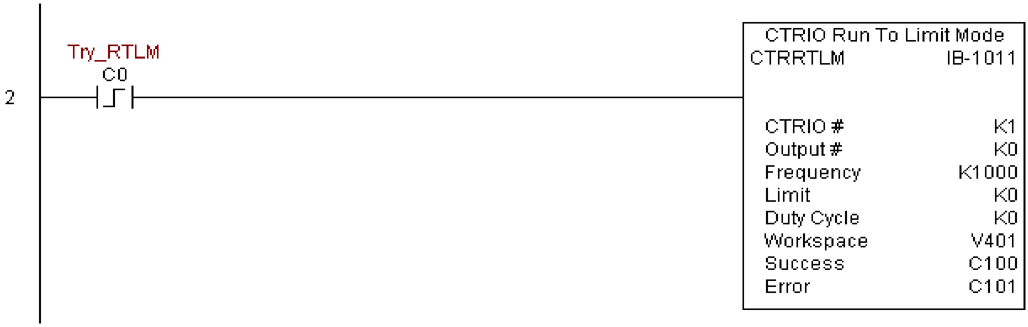
Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Frequency	V,K	K20-20000; See DL06 V-memory map - Data Words
Limit	V,K	K0-FF; See DL06 V-memory map - Data Words
Duty Cycle	V,K	K0-99; See DL06 V-memory map - Data Words
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRRTLM Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



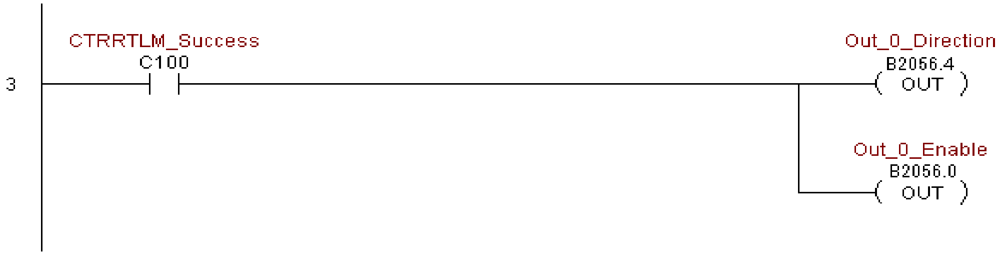
Rung 2: This CTRIO Run To Limit Mode IBox sets up Output #0 in CTRIO #1 to output pulses at a Frequency of 1000 Hz until Llimit #0 comes on. This example program requires that you load CTRRTLM_IBox.cwb into your Hx-CTRIO(2) module.



(example continued on next page)

CTRRTLM Example (cont'd)

Rung 3: If the Run To Limit Mode parameters are OK, set the Direction Bit and Enable the output.

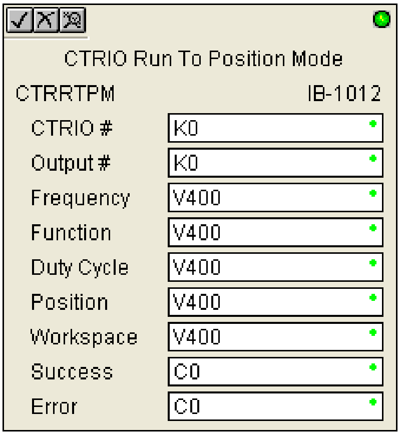


CTRIO Run to Position Mode (CTRRTPM) (IB-1012)

DS	Used
HPP	N/A

CTRIO Run To Position Mode, on a leading edge transition to this IBox, loads the Run to Position command and given parameters on a specific Output resource.

Valid Function Values are:
00: Less Than Ch1/Fn1
10: Greater Than Ch1/Fn1
01: Less Than Ch1/Fn2
11: Greater Than Ch1/Fn2
02: Less Than Ch2/Fn1
12: Greater Than Ch2/Fn1
03: Less Than Ch2/Fn2
13: Greater Than Ch2/Fn2


CTRIO Run To Position Mode

CTRRTPM IB-1012

CTRIO #	K0
Output #	K0
Frequency	V400
Function	V400
Duty Cycle	V400
Position	V400
Workspace	V400
Success	C0
Error	C0

This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

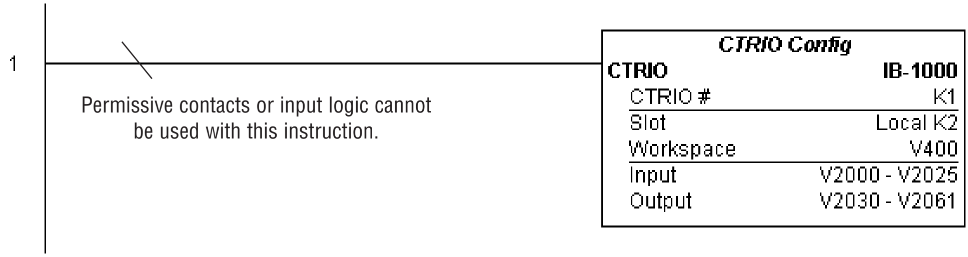
CTRRTPM Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Frequency: specifies the output pulse rate (H0-CTRIO: 20Hz - 25KHz / H0-CTRIO2: 20Hz - 250 KHz)
- Duty Cycle: specifies the % of on time versus off time. This is a hex number. Default of 0 is 50%, also entering 50 will yield 50%. 50% duty cycle is defined as on half the time and off half the time
- Position: specifies the count value, as measured on the encoder input, at which the output pulse train will be turned off
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Frequency	V,K	K20-20000; See DL06 V-memory map - Data Words
Duty Cycle	V,K	K0-99; See DL06 V-memory map
Position	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRRTPM Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



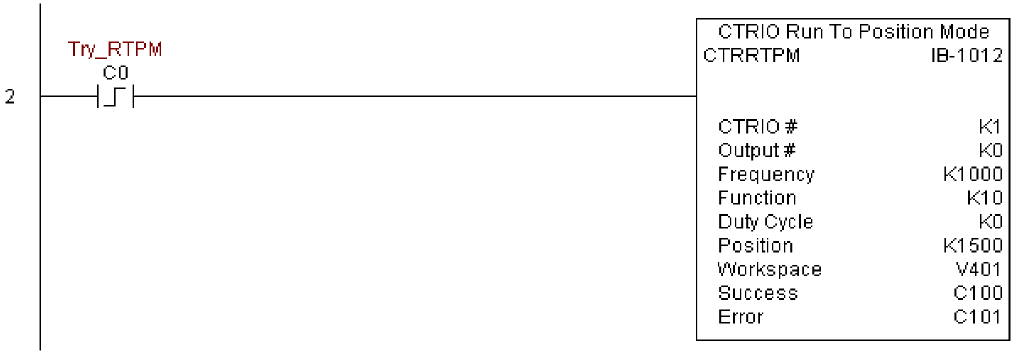
Permissive contacts or input logic cannot be used with this instruction.

CTRIO Config
CTRIO **IB-1000**
CTRIO # K1
Slot Local K2
Workspace V400
Input V2000 - V2025
Output V2030 - V2061

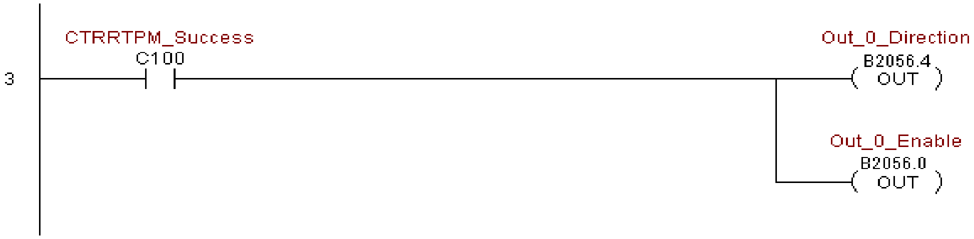
(example continued on next page)

CTRRTPM Example (cont'd)

Rung 2: This CTRIO Run To Position Mode IBox sets up Output #0 in CTRIO #1 to output pulses at a Frequency of 1000 Hz, use the 'Greater than Ch1/Fn1' comparison operator, until the input position of 1500 is reached. This example program requires that you load CTRRTPM_IBox.cwb into your Hx-CTRIO(2) module.



Rung 3: If the Run To Position Mode parameters are OK, set the Direction Bit and Enable the output.



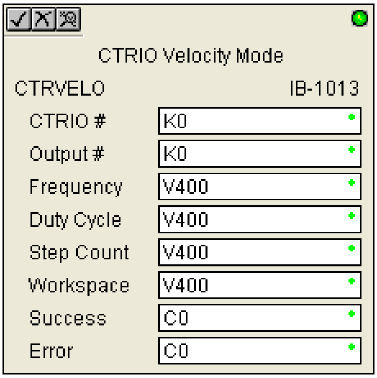
CTRIO Velocity Mode (CTRVELO) (IB-1013)

DS	Used
HPP	N/A

CTRIO Velocity Mode loads the Velocity command and given parameters on a specific Output resource on a leading edge transition to this IBox.

This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



CTRIO Velocity Mode

CTRVELO IB-1013

CTRIO # K0

Output # K0

Frequency V400

Duty Cycle V400

Step Count V400

Workspace V400

Success C0

Error C0

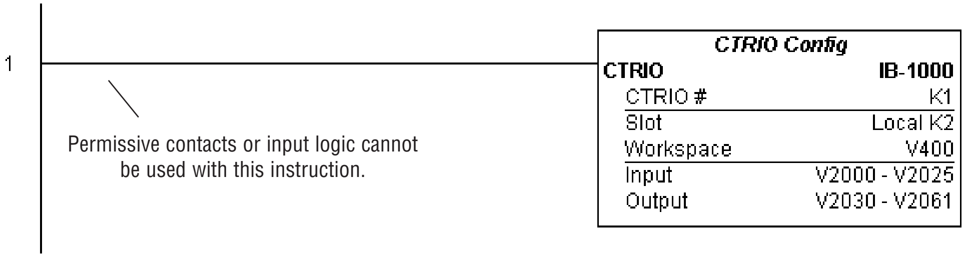
CTRVELO Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Frequency: specifies the output pulse rate (H0-CTRIO: 20Hz - 25KHz / H0-CTRIO2: 20Hz - 250 KHz)
- Duty Cycle: specifies the % of on time versus off time. This is a hex number. Default of 0 is 50%, also entering 50 will yield 50%. 50% duty cycle is defined as on half the time and off half the time
- Step Count: This DWORD value specifies the number of pulses to output. A Step Count value of -1 (or 0XFFFFFFFF) causes the CTRIO to output pulses continuously. Negative Step Count values must be V-Memory references.
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Frequency	V,K	K20-20000; See DL06 V-memory map - Data Words
Duty Cycle	V,K	K0-99; See DL06 V-memory map
Step Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRVELO Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.

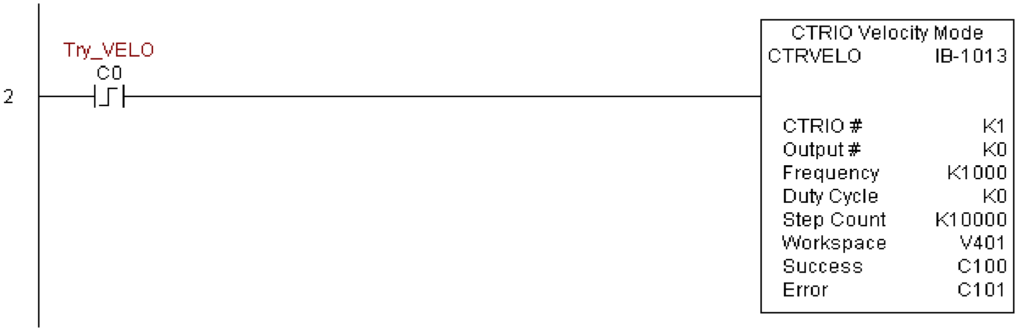

1
Permissive contacts or input logic cannot be used with this instruction.

CTRIO Config

CTRIO **IB-1000**

CTRIO #	K1
Slot	Local K2
Workspace	V400
Input	V2000 - V2025
Output	V2030 - V2061

Rung 2: This CTRIO Velocity Mode IBox sets up Output #0 in CTRIO #1 to output 10,000 pulses at a Frequency of 1000 Hz. This example program requires that you load CTRVELO_IBox.cwb into your Hx-CTRIO(2) module.



CTRIO Velocity Mode

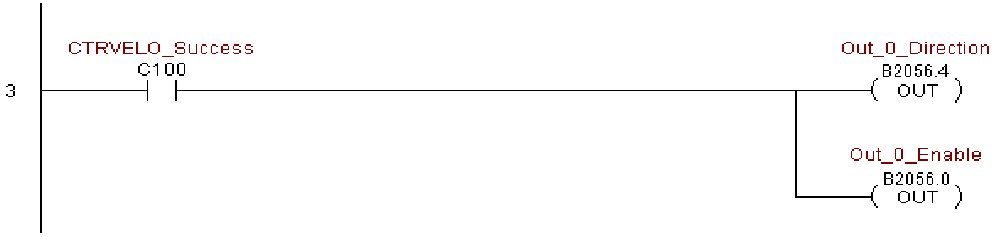
CTRVELO IB-1013

CTRIO #	K1
Output #	K0
Frequency	K1000
Duty Cycle	K0
Step Count	K10000
Workspace	V401
Success	C100
Error	C101

(example continued on next page)

CTRVELO Example (cont'd)

Rung 3: If the Velocity Mode parameters are OK, set the Direction Bit and Enable the output.

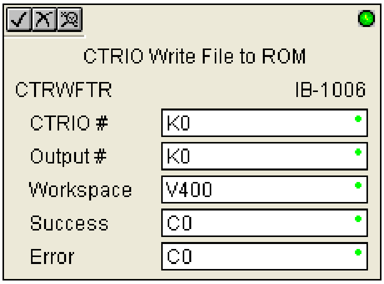


CTRIO Write File to ROM (CTRWFTR) (IB-1006)

DS	Used
HPP	N/A

CTRIO Write File to ROM writes the runtime changes made to a loaded CTRIO Preset Table back to Flash ROM on a leading edge transition to this IBox. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



CTRIO Write File to ROM

CTRWFTR IB-1006

CTRIO # K0

Output # K0

Workspace V400

Success C0

Error C0

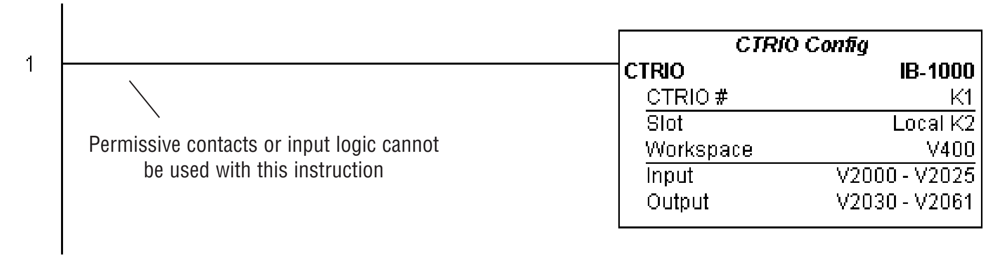
CTRWFTR Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

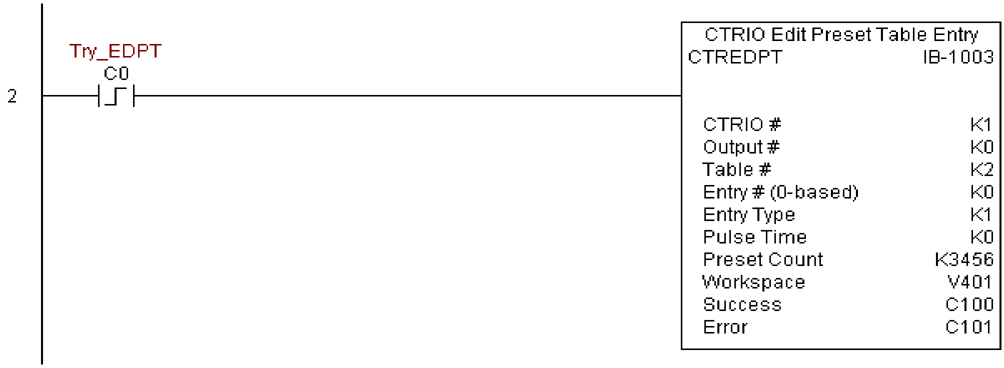
CTRWFTR Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.


1
Permissive contacts or input logic cannot be used with this instruction

CTRIO Config	
CTRIO	IB-1000
CTRIO #	K1
Slot	Local K2
Workspace	V400
Input	V2000 - V2025
Output	V2030 - V2061

Rung 2: This CTRIO Edit Preset Table Entry IBox will change Entry 0 in Table #2 to be a RESET at Count 3456. This example program requires that you load CTRWFTR_IBox.cwb into your Hx-CTRIO(2) module.

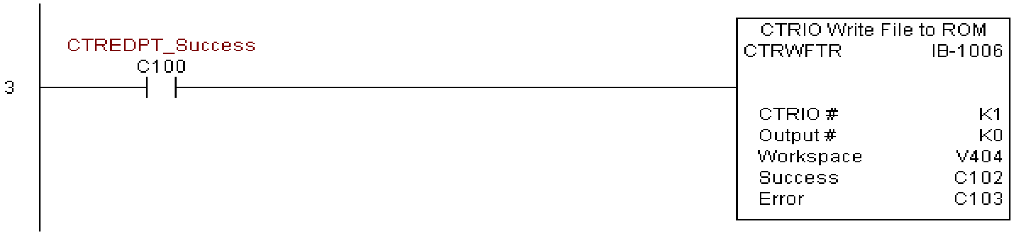


CTRIO Edit Preset Table Entry	
CTREDPT	IB-1003
CTRIO #	K1
Output #	K0
Table #	K2
Entry # (0-based)	K0
Entry Type	K1
Pulse Time	K0
Preset Count	K3456
Workspace	V401
Success	C100
Error	C101

(example continued on next page)

CTRWFTR Example (cont'd)

Rung 3: If the file is successfully edited, use a Write File To ROM IBox to save the edited table back to the CTRIO's ROM, thereby making the changes retentive.


3
CTREDPT_Success
C100
CTRIO Write File to ROM
CTRWFTR
IB-1006
CTRIO #
K1
Output #
K0
Workspace
V404
Success
C102
Error
C103