D3–350 CPU User Manual

Automationdirect.com

WARNING

Thank you for purchasing automation equipment from ***Automationdirect.com***™. We want your new ***Direct***LOGIC™ 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 are 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.com***™ *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 Desk Reference. 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.com***™ 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.com***™ disclaims any proprietary interest in the marks and names of others.

AVERTISSEMENT

Nous vous remercions d'avoir acheté l'équipement d'automatisation de **Automationdirect.com**™. Nous tenons à ce que votre nouvel équipement d'automatisation ***Direct***LOGIC™ 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.com**™ 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.com**™, 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.com**™ nie tout intérêt dans les autres marques et désignations.

Manual Revisions

If you contact us in reference to this manual, remember to include the revision number.

Title: D3–350 CPU User Manual
Manual Number: D3–350–M

Issue	Date	Description of Changes
Original	8/97	Original Issue
Rev A	6/98	Minor corrections
Rev B	5/99	Automationdirect.com
Rev C	8/02	Replaced F3–16TA–1 with F3–16TA–2

Table of Contents

Chapter 1: Getting Started

Chapter 2: Installation, Wiring, and Specifications

Chapter 4: System Design and Configuration

Chapter 5: Standard RLL Instructions

Chapter 6: Drum Instruction Programming

Chapter 7: RLL*PLUS* Stage Programming

Chapter 8: PID Loop Operation

Appendix F: I/O Addressing Conventional Method

Index

1

Getting Started

In This Chapter. . . .

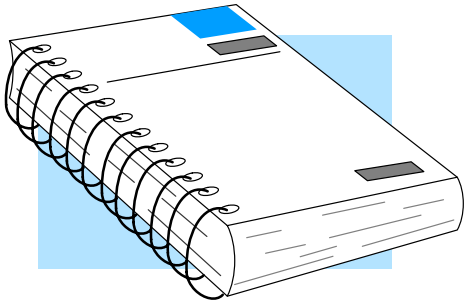
— Introduction
— DL305 System Components
— Programming Methods
— ***Direct***LOGIC™ Part Numbering System
— Quick Start for PLC Validation and Programming
— Steps to Designing a Successful System

Introduction

The Purpose of this Manual

Thank you for purchasing our DL305 family of products. This manual shows you how to install, program, and maintain the equipment. 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 DL305 PLCs and components, and for the PLC programmer. If you understand PLC systems, our manuals will provide all the information you need to start and keep your system up and running.



Where to Begin

If you already understand PLCs please read Chapter 2, “Installation, Wiring, and Specifications”, and proceed on to other chapters as needed. Keep this manual handy for reference when you have questions. If you are a new DL305 customer, we suggest you read this manual completely to understand the wide variety of features in the DL305 family of products. We believe you will be pleasantly surprised with how much you can accomplish with PLC***Direct***™ products.

Supplemental Manuals

If you have purchased operator interfaces or ***Direct***SOFT™, you will need to supplement this manual with the manuals that are written for these products.

Technical Support

We realize that even though we strive to be the best, the information may be arranged in such a way you cannot find what you are looking for. First, check these resources for help in locating the information:

- **Table of Contents** – chapter and section listing of contents, in the front of this manual
- **Quick Guide to Contents** – chapter summary listing on the next page
- **Appendices** – reference material for key topics, near the end of this manual
- **Index** – alphabetical listing of key words, at the end of this manual

You can also check our online resources for the latest product support information:

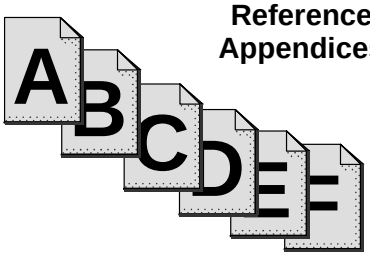
- **Internet** – the address is http://automationdirect.plcdirect.com

If you still need assistance, please call us at 770–844–4200. Our technical support group is glad to work with you in answering your questions. They are available Monday through Friday from 9:00 A.M. to 6:00 P.M. Eastern Standard Time. If you have a comment or question about any of our products, services, or manuals, please fill out and return the ‘Suggestions’ card that was shipped with this manual.

Chapters The main contents of this manual are organized into the following nine chapters:

1	**Getting Started**	introduces the various components of a DL305 PLC system. Also includes tips on getting started and how to design a successful system.
2	**Installation, Wiring, and Specifications**	explains how to prepare for system installation, and gives safety guidelines to help protect your personnel and machinery. It includes system and I/O wiring diagrams, and I/O module specifications.
3	**CPU Specifications and Operation**	lists the specifications for the operating system and communications ports for the DL350 CPU. PLC scan activities, task execution times, data types, and CPU memory maps are given for each CPU type.
4	**System Design and Configuration**	explains various ways to expand the I/O and communications networks for your DL305 PLC system. Topics include base power budgeting for I/O modules, local expansion, slice I/O, remote I/O, and MODBUS® networks.
5	**Standard RLL Instructions**	describes how each of the 150+ standard instructions operate. Example programs for both ***Direct***SOFT and the Handheld Programmer are included. Note that drum and stage instructions are in subsequent chapters.
6	**Drum Instruction Programming**	describes the four different timer/event drum sequencers available in the DL350 CPU. It begins with a detailed theory of operation, covers control techniques, and reveals how drums solve sequential process problems.
7	**RLL*Plus* Stage Programming**	ideal for the beginning programmer – it shows how to develop simple state transition diagrams for your process, and then convert them to stage programs. This programming method is a real time-saver for programming and debug.
8	**PID Loop Operation**	explains how the DL350 PID Loop capability can control up to 4–loops. Discusses several topics, including PID variable data, loop tuning, alarms, ramp/soak profiles, and more.
9	**Maintenance and Troubleshooting**	is a guide designed to aid you in diagnosing, repairing and avoiding system problems. It includes tips for finding I/O circuit problems, ladder program errors, etc.

Appendices Additional reference information on the DL350 is in the following five appendices:



Reference Appendices

- **A** – Auxiliary Functions
- **B** – DL350 Error Codes
- **C** – Instruction Execution Times
- **D** – Special Relays
- **E** – Product Weights
- **F** – Base and I/O Confguration (Conventional Method)

DL305 System Components

The DL305 family is a versatile product line that provides a wide variety of features in an extremely compact package. The CPUs are small, but offer many instructions normally only found in larger, more expensive systems. The modular design also offers more flexibility in the fast moving industry of control systems. The following is a summary of the major DL305 system components.

CPUs

There are three feature enhanced CPUs in this product line, the DL330, DL340, and the DL350. This manual covers the DL350 CPU ***only***. The DL330 and DL340 CPUs are covered in detail in the DL305C User Manual. The DL350 CPU includes built-in communication ports, a large amount of program memory, a substantial instruction set and advanced diagnostics. It also features drum timers, floating–point math, and built in PID loops with automatic tuning.

Bases

Three base sizes are available: 5 slot, 8 slot, and 10 slot. One slot is for the CPU, the remaining slots are for I/O modules. All bases include a built-in power supply. Currently there are two versions of the bases. The xxxxx–1 bases were designed to compliment the DL350 CPU. Any of the three CPUs will work in either type of base and the bases can be mixed in a system. When the DL350 CPU is used in an old base, or if it is in a system of mixed bases, it will act similar to the DL340 CPU in addressing and I/O configuration (See Appendix F).

I/O Configuration

The DL350 CPU can support up to 368 I/O points with the bases currently available. These points can be assigned as input or output points. The DL305 system can also be expanded by adding remote I/O. The DL350 also provides a built–in master for remote I/O networks. The I/O configuration is explained in Chapter 4, System Design and Configuration.

I/O Modules

The DL305 has some of the most powerful modules in the industry. A complete range of discrete modules which support 24 VDC, 110/220 VAC and up to 10A relay outputs are offered. The analog modules provide 12 bit resolution and several selections of input and output signal ranges (including bipolar).

Programming Methods

There are two programming methods available to the DL350 CPU, RLL (Relay Ladder Logic) and RLL*PLUS* (Stage Programming). Both the ***Direct***SOFT™ programming package and the handheld programmer support RLL and Stage.

DirectSOFT **Programming for Windows**™

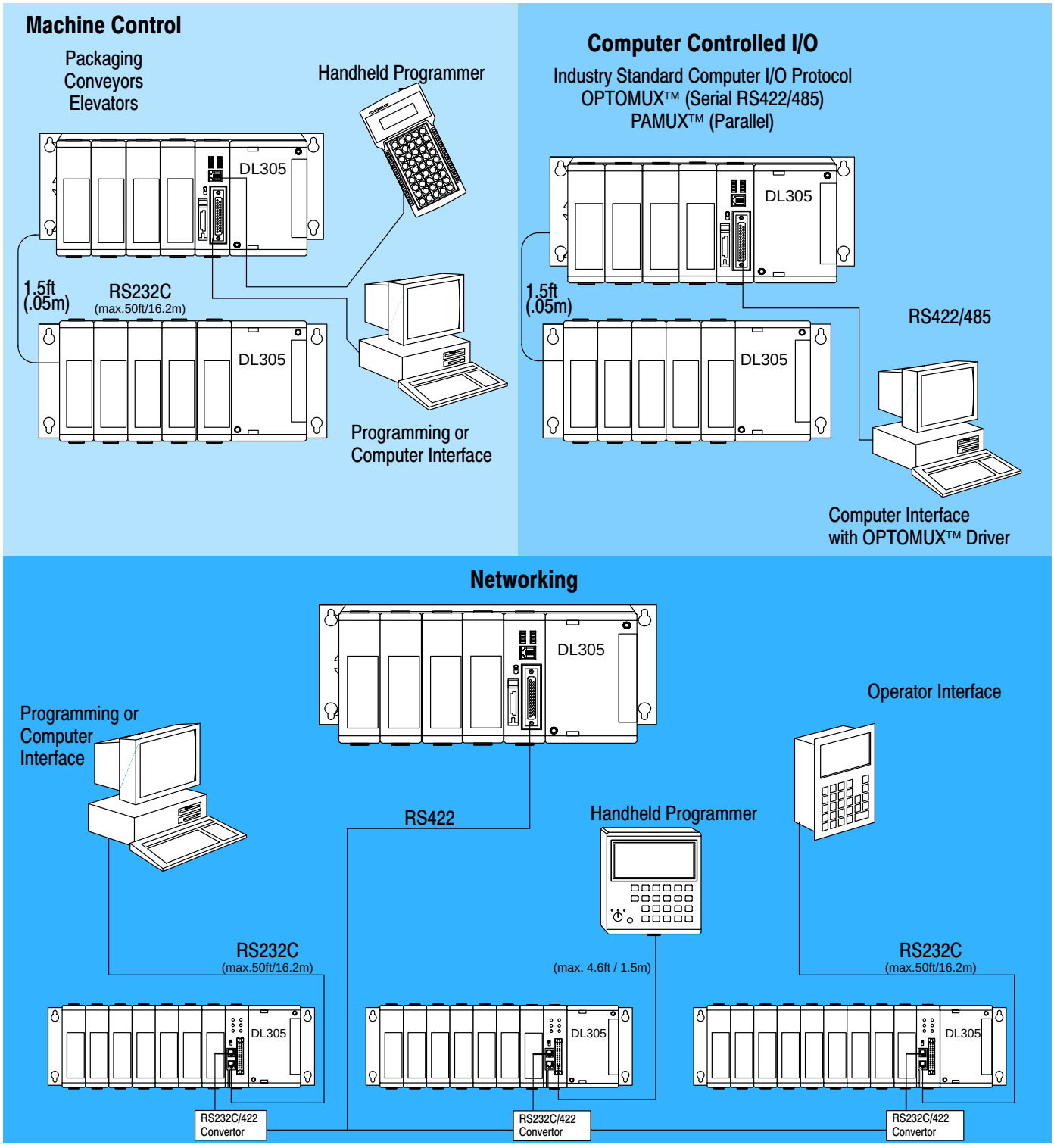
The DL305 can be programmed with one of the most advanced programming packages in the industry —***Direct***SOFT. ***Direct***SOFT is a Windows-based software package that supports many Windows-features you are already know, such as cut and paste between applications, point and click editing, viewing and editing multiple application programs at the same time, etc. ***Direct***SOFT universally supports the ***Direct***LOGIC™ CPU families. This means you can use the *same* ***Direct***SOFT package to program DL105, DL205, DL305, DL405 or any new CPUs we may add to our product line. There is a separate manual that discusses the ***Direct***SOFT programming software.

Handheld Programmer

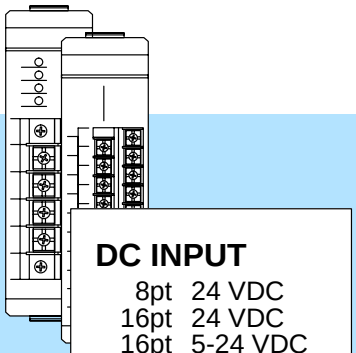
The DL350 CPU has a built-in programming port for use with the DL205 handheld programmer (D2–HPP). The handheld programmer can be used to create, modify and debug your application program. A separate manual that discusses the Handheld Programmer is available.

DL305 System Diagrams

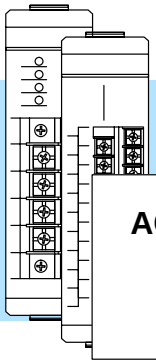
The diagram below shows the major components and configurations of the DL305 system. The next two pages show specific components for building your system.



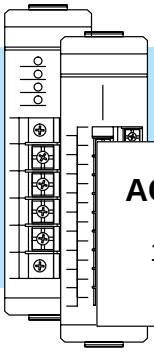
*Direct*LOGIC™



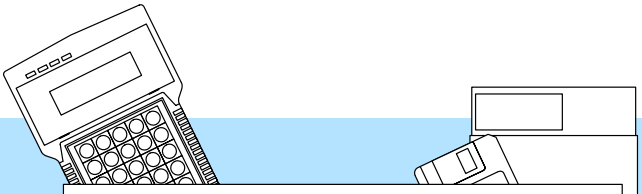
DC INPUT
8pt 24 VDC
16pt 24 VDC
16pt 5-24 VDC
16pt 12-24 VDC



AC INPUT
8pt 110 VAC
16pt 110 VAC
8pt 220 VAC

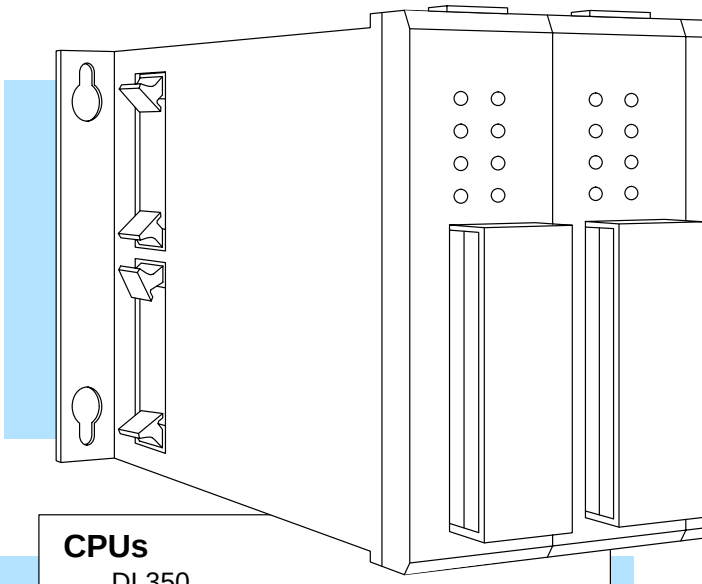


AC/DC INPUT
8pt 24 VAC/DC
16pt 24 VAC/DC

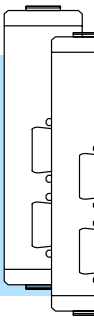


PROGRAMMING
Handheld Programmer for RLL and RLL*PLUS*

DirectSOFT Programming for Windows™



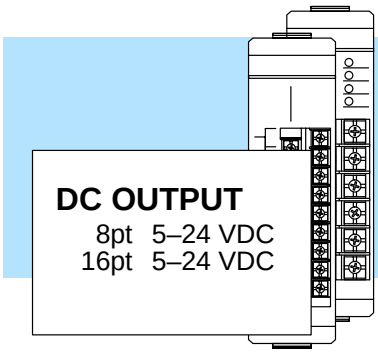
CPUs
DL350
7.6K Built in Flash memory
and 2 Built-in Ports



ASCII BASIC Modules
RS232C / RS422 / RS485
Built-in Radio Modem
Built-in Telephone Modem
Program Memory 64K/128K

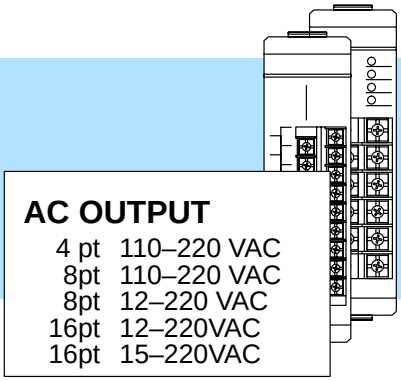
BASES
5 Slot Base w/Expansion Capability, 110/220 VAC P/S
5 Slot Base w/Expansion Capability, 24 VDC Supply
8 Slot Base w/Expansion Capability, 110/220 VAC P/S
10 Slot Base w/Expansion Capability, 110/220 VAC P/S

DL305 Family



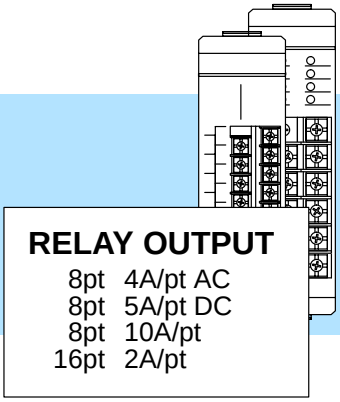
DC OUTPUT

8pt	5–24 VDC
16pt	5–24 VDC



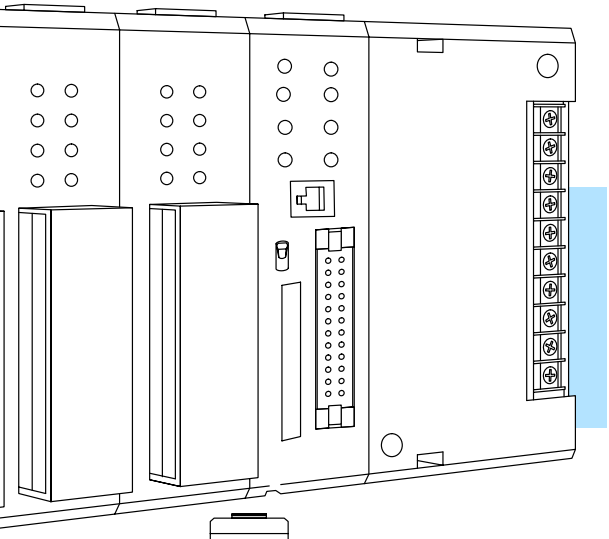
AC OUTPUT

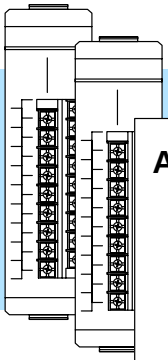
4 pt	110–220 VAC
8pt	110–220 VAC
8pt	12–220 VAC
16pt	12–220VAC
16pt	15–220VAC



RELAY OUTPUT

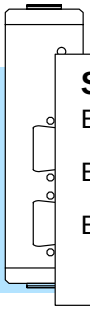
8pt	4A/pt AC
8pt	5A/pt DC
8pt	10A/pt
16pt	2A/pt





ANALOG

4ch	INPUT
8ch	INPUT
16ch	INPUT
2ch	OUTPUT
4ch	OUTPUT
8ch	TEMPERATURE TRANSDUCER INPUT
8ch	THERMOCOUPLE INPUT



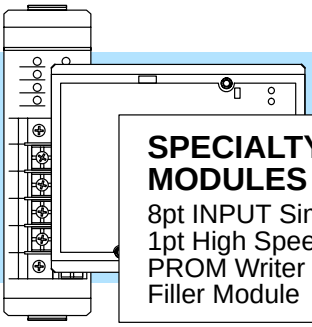
SPECIALTY CPUs

Bridge CPU to connect to host w/OPTOMUX™ Driver
Bridge CPU w/FACTS Extended Basic Programming
Bridge CPU to connect to High-speed PAMUX™ compatible host

NETWORKING

RS232C Data Communication Unit
RS422 Data Communication Unit
MODBUS® Slave Module
MODBUS® Slave Module w/Radio Modem

Universal connector:
RS232C / RS422/485 Convertor



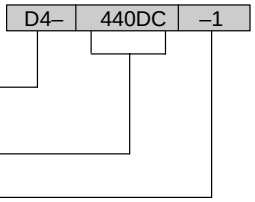
SPECIALTY MODULES / UNITS

8pt INPUT Simulator
1pt High Speed Counter
PROM Writer Unit
Filler Module

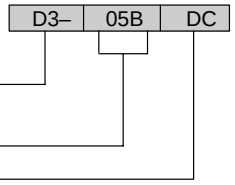
*Direct*LOGIC™ Part Numbering System

As you examine this manual, you will notice there are many different products available. Sometimes it is difficult to remember the specifications for any given product. However, if you take a few minutes to understand the numbering system, it may save you some time and confusion. The charts below show how the part numbering systems work for each product category. Part numbers for accessory items such as cables, batteries, memory cartridges, etc. are typically an abbreviation of the description for the item.

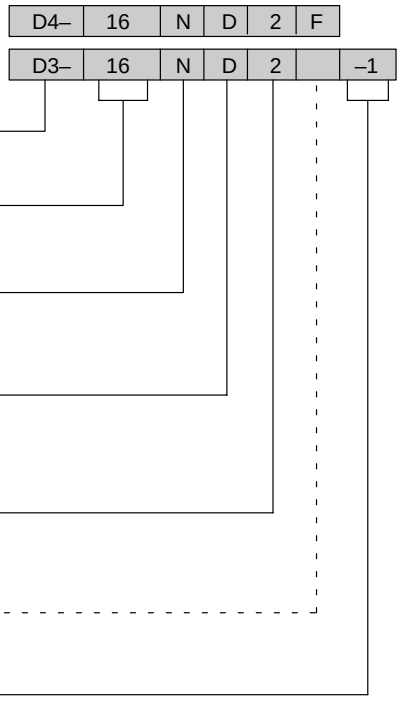
CPUs **Specialty CPUs**	
Product family	D1/F1 D2/F2 D3/F3 D4/F4
Class of CPU / Abbreviation	230...,330...,430...
Denotes a differentiation between Similar modules	–1, –2, –3, –4



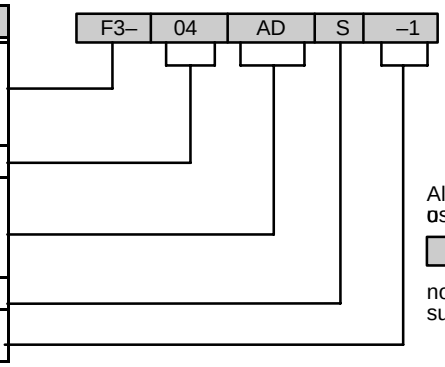
Bases	
Product family	D2/F2 D3/F3 D4/F4
Number of slots	##B
Type of Base	DC or empty



Discrete I/O	
DL205 Product family DL305 Product family DL405 Product family	D2/F2 D3/F3 D4/F4
Number of points	04/08/12/16/32
Input Output Combination	N T C
AC DC Either Relay	A D E R
Current Sinking Current Sourcing Current Sinking/Sourcing	1 2 3
High Current Isolation Fast I/O	H S F
Denotes a differentiation between Similar modules	–1, –2, –3, –4



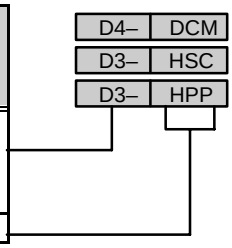
Analog I/O	
DL205 Product family	D2/F2
DL305 Product family	D3/F3
DL405 Product family	D4/F4
Number of channels	02/04/08/16
Input (Analog to Digital)	AD
Output (Digital to Analog)	DA
Combination	AND
Isolated	S
Denotes a differentiation between Similar modules	–1, –2, –3, –4



Alternate example of Analog I/O using abbreviations

note: –n indicates thermocouple type such as: J, K, T, R, S or E

Communication and Networking Special I/O and Devices Programming	
DL205 Product family	D2/F2
DL305 Product family	D3/F3
DL405 Product family	D4/F4
Name Abbreviation	see example

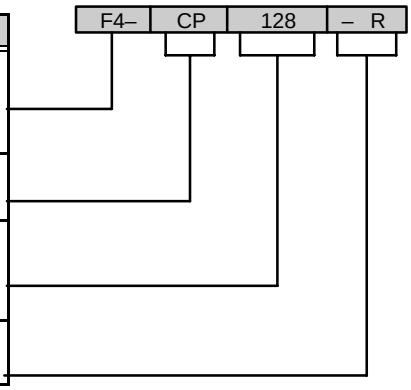


DCM (Data Communication Module)

HSC (High Speed Counter)

HPP (RLL PLUS Handheld Programmer)

CoProcessors and ASCII BASIC Modules	
DL205 Product family	D2/F2
DL305 Product family	D3/F3
DL405 Product family	D4/F4
CoProcessor	CP
ASCII BASIC	AB
64K memory	64
128K memory	128
512K memory	512
Radio modem	R
Telephone modem	T



Quick Start for PLC Validation and Programming

If you have experience with PLCs, or want to setup a quick example, this section is what you want to use. This example is not intended to explain everything needed to start-up your system. It is only intended to provide a general picture of what is needed to get your system powered-up.

Step 1: Unpack the DL305 Equipment

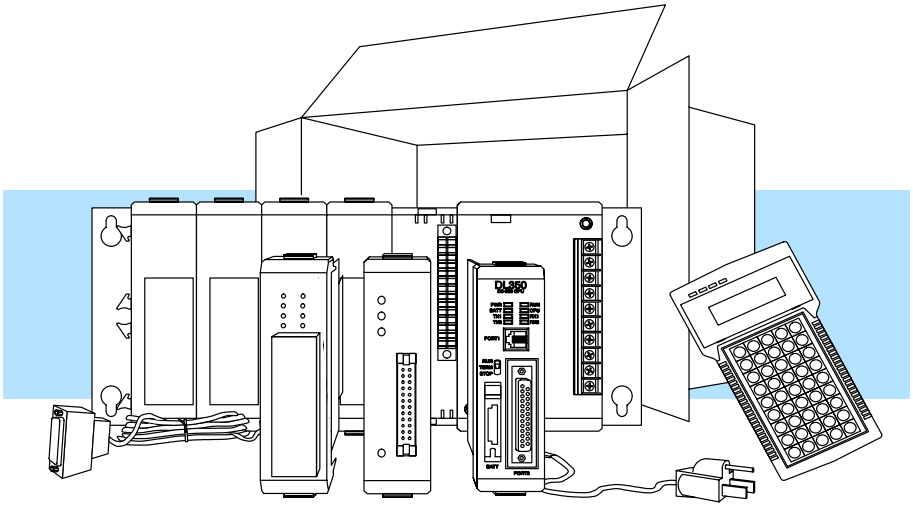
Unpack the DL305 equipment and verify you have the parts necessary to build this demonstration system. The minimum parts needed are as follows:

- Base
- CPU
- D3–08ND2 DC input module or a D3–08SIM input simulator module
- D3–08TD2 DC output module
- *Power cord
- *Hook up wire
- *A 24 VDC toggle switch (if not using the input simulator module)
- *A screwdriver, regular or Phillips type

* These items are not supplied with your PLC.

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

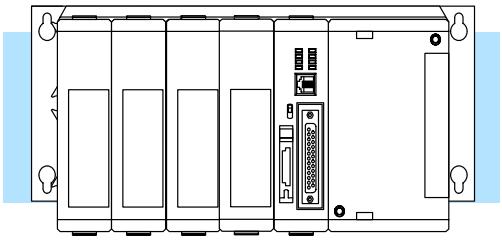
- ***Direct***SOFT Programming Software, ***Direct***SOFT Manual, and a programming cable (connects the CPU to a personal computer), or
- D2–HPP Handheld Programmer and the Handheld Programmer Manual



Step 2: Install the CPU and I/O Modules

Insert the CPU and I/O into the base. The CPU must go into the first slot of the base (adjacent to the power supply).

- Each unit has a plastic retaining clip at the top and bottom.
- With the unit square to the base, slide it in using the upper and lower guides.
- Gently push the unit back until it is firmly seated in the backplane and the plastic clips lock in place.

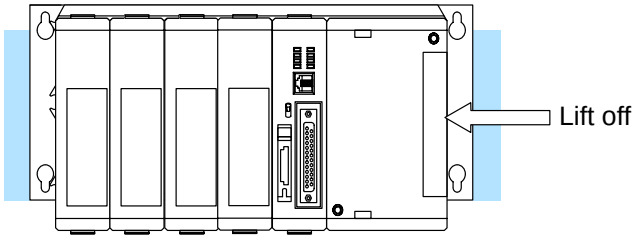


CPU must reside in first slot!

Placement of discrete, analog and relay modules are not critical and may go in any slot in any base however for this example install the output module in the slot next to the CPU and the input module in the next. Limiting factors for other types of modules are discussed in Chapter 4, System Design and Configuration. You must also make sure you do not exceed the power budget for each base in your system configuration. Power budgeting is also discussed in Chapter 4.

Step 3: Remove Terminal Strip Access Cover

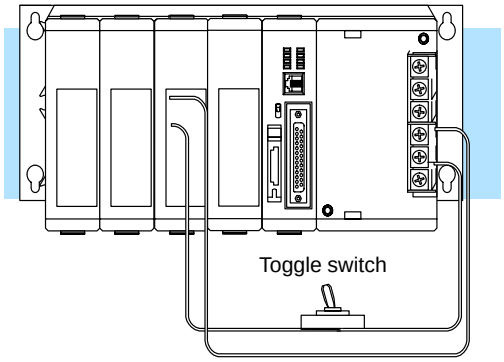
Remove the terminal strip cover. It is a small strip of clear plastic that is located on the base power supply.



Step 4: Add I/O Simulation

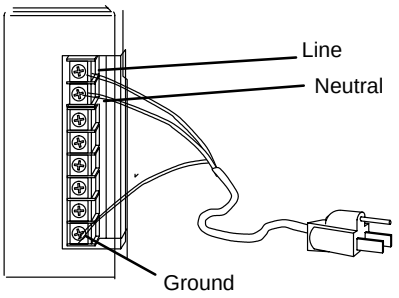
To finish this quick start exercise or study other examples in this manual, you will need to install an input simulator module (or wire an input switch as shown below), and add an output module. Using an input simulator is the quickest way to get physical inputs for checking out the system or a new program. To monitor output status, any discrete output module will work.

Wire the switches or other field devices prior to applying power to the system to ensure a point is not accidentally turned on during the wiring operation. Wire the input module (X0) to the toggle switch and 24VDC auxiliary power supply on the CPU terminal strip as shown. Chapter 2, Installation, Wiring, and Specifications provides a list of I/O wiring guidelines.



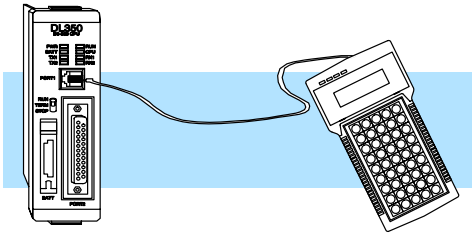
Step 5: Connect the Power Wiring

Connect the wires as shown. Observe all precautions stated earlier in this manual. For details on wiring see Chapter 2, Installation, Wiring, and Specifications. When the wiring is complete, replace the CPU and module covers. Do not apply power at this time.



Step 6: Connect the Handheld Programmer

Connect the D2–HPP to the top port (RJ style phone jack) of the CPU using the appropriate cable.



Step 7: Switch On the System Power

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

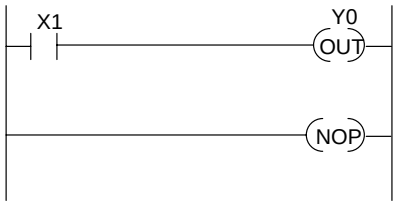
Step 8: Enter the Program

Slide the Mode Switch on the CPU to the STOP position and then back to the TERM position. This puts the CPU in the program mode and allows access to the CPU program. The PGM indicator should be illuminated on the HPP. Enter the following keystrokes on the HPP:

NOTE: It is not necessary for you to configure the I/O for this system since the DL350 CPU automatically examines any installed modules and establishes the correct configuration.

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
GX OUT	→	C 2	ENT



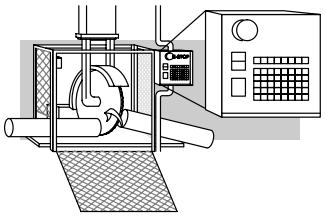
After entering the simple example program slide the switch from the TERM position to the RUN position and back to TERM. The RUN indicator on the CPU will come on indicating the CPU has entered the run mode. If not repeat Step 8 insuring the program is entered properly or refer to the troubleshooting guide in chapter 9.

During Run mode operation, the output status indicator on the output module should reflect the switch status. When the switch is on the output should be on.

Steps to Designing a Successful System

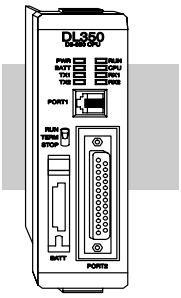
Step 1: Review the Installation Guidelines

Always make safety your first priority in any system application. Chapter 2 provides several guidelines that will help provide a safer, more reliable system. This chapter also includes wiring guidelines for the various system components.



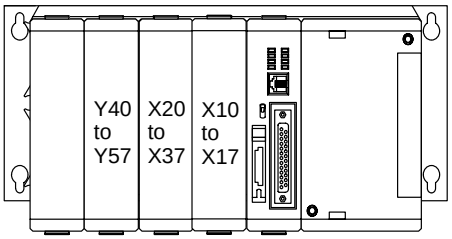
Step 2: Understand the CPU Setup Procedures

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



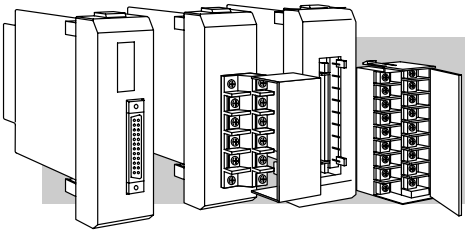
Step 3: Understand the I/O System Configurations

It is important to understand how your local I/O system can be configured. It is also important to understand how the system Power Budget is calculated. This can affect your I/O placement and/or configuration options.



Step 4: Determine the I/O Module Specifications and Wiring Characteristics

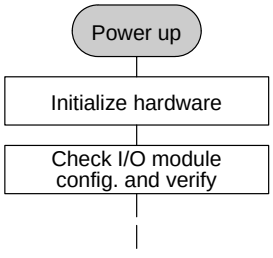
There are many different I/O modules available with the DL305 system. Chapter 2 provides the specifications and wiring diagrams for the discrete I/O modules.



NOTE: Specialty modules have their own manuals and are not included in this manual.

Step 5: Understand the System Operation

Before you begin to enter a program, it is very helpful to understand how the DL305 system processes information. This involves not only program execution steps, but also involves the various modes of operation and memory layout characteristics. See Chapter 3 for more information.

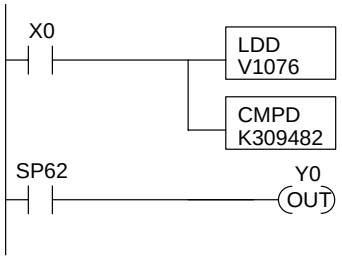


Step 6: Review the Programming Concepts

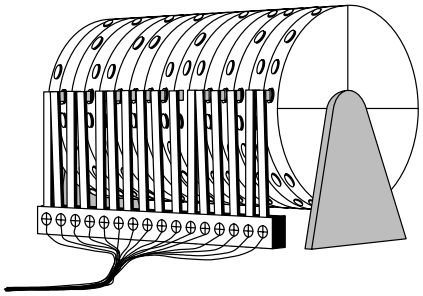
The DL305 provides four main approaches to solving the application program, including the PID loop task depicted in the next figure.

- 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 augment drums, stages, and loops.
- The DL350 has four timer/event drum types, each with up to 16 steps. They offer both time and/or event-based step transitions. Drums are 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 of your process.
- The DL350 PID Loop Operation uses setup tables to configure 4 loops. Features include; auto tuning, alarms, SP ramp/soak generation, and more.

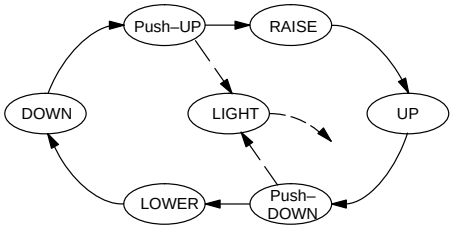
Standard RLL Programming
(see Chapter 5)



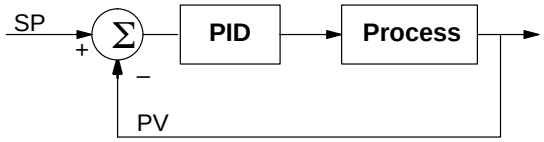
Timer/Event Drum Sequencer
(see Chapter 6)



Stage Programming
(see Chapter 7)

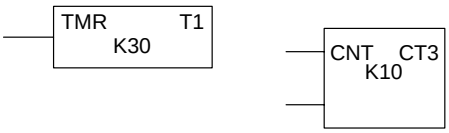


PID Loop Operation
(see Chapter 8)



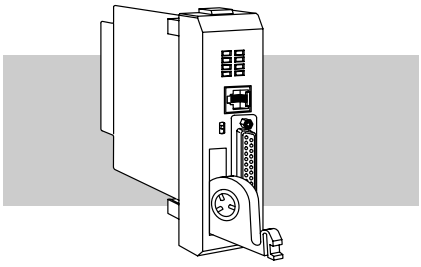
Step 7: Choose the Instructions

Once you have installed the system and understand the theory of operation, you can choose from one of the most powerful instruction sets available.



Step 8: Understand the Maintenance and Troubleshooting Procedures

Equipment failures can occur at any time. Switches fail, batteries need to be replaced, etc. In most cases, the majority of the troubleshooting and maintenance time is spent trying to locate the problem. The DL305 system has many built-in features that help you quickly identify problems. Refer to Chapter 9 for diagnostics and troubleshooting tips.



2

Installation, Wiring, and Specifications

In This Chapter. . . .

Safety Guidelines

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 or damage to equipment. Do not rely on the automation system alone to provide a safe operating environment. You should use external electromechanical devices, such as relays or limit switches, that are independent of the PLC application to provide protection for any part of the system that may cause personal injury or damage.

Every automation application is different, so there may be special requirements for your particular application. Make sure you follow all national, state, and local government requirements 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.

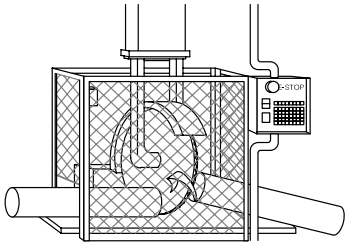
Safety Techniques

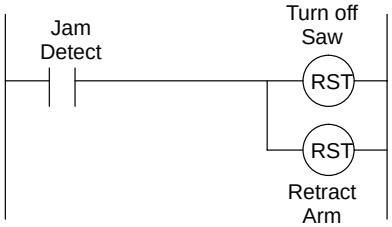
The publications mentioned provide many ideas and requirements for system safety. At a minimum, you should follow these regulations. Using the techniques listed below will further help reduce the risk of safety problems.

- Orderly system shutdown sequence in the PLC control program.
- Emergency stop switch for disconnecting system power.

Orderly System Shutdown

The first level of protection can be provided with the PLC control program by identifying machine problems. Analyze your application and identify any shutdown sequences that must be performed. Typical problems such as jammed or missing parts, empty bins, etc., create a risk of personal injury or equipment damage.





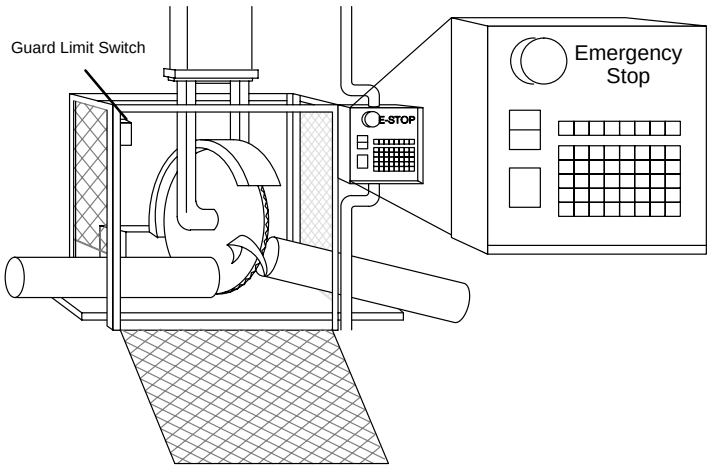
WARNING: The control program *must not* be the only form of protection for any problems that may result in a risk of personal injury or equipment damage.

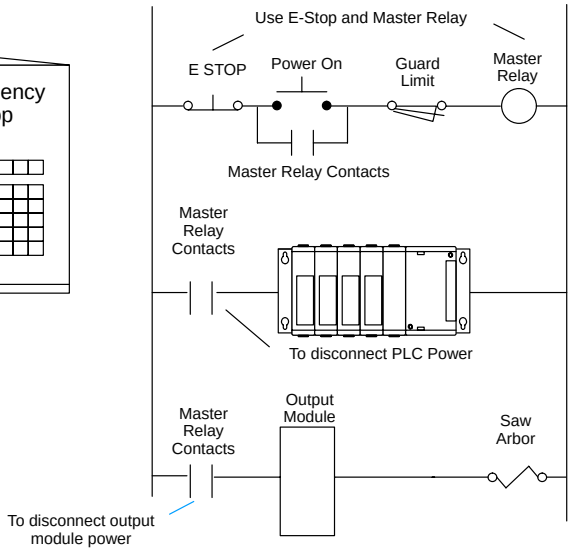
System Power Disconnect

By using electromechanical devices, such as master control relays and/or limit switches, you can prevent accidental equipment startup. When installed properly, these devices will prevent *any* machine operations from occurring.

For example, if the machine has a jammed part, the PLC control program can turn off the saw blade and retract the arbor. However, since the operator must open the guard to remove the part, you must include a bypass switch to disconnect *all* system power any time the guard is opened.

The operator must also have a quick method of manually disconnecting *all* system power. This is accomplished with a mechanical device clearly labeled as an **Emergency Stop** switch.





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 ensure a known starting point.

Mounting Guidelines

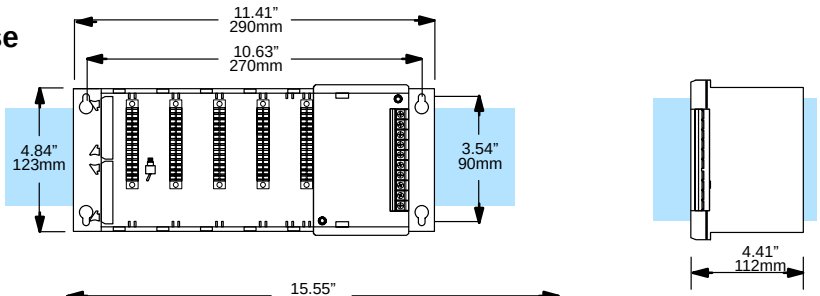
Before installing the PLC system you will need to know the dimensions for the components. The diagrams on the following pages provide the component dimensions to use in defining your enclosure specifications. Remember to leave room for potential expansion.

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

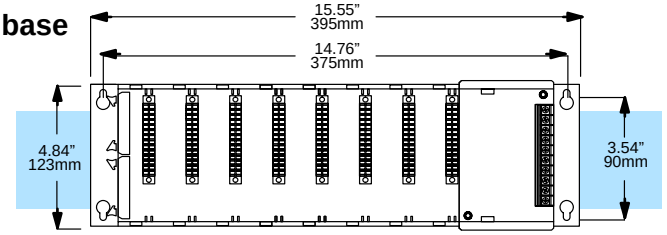
Base Dimensions

The following information shows the proper mounting dimensions. The height dimension is the same for all bases. The depth varies depending on your choice of I/O module. The length varies as the number of slots increase. Make sure you have followed the installation guidelines for proper spacing.

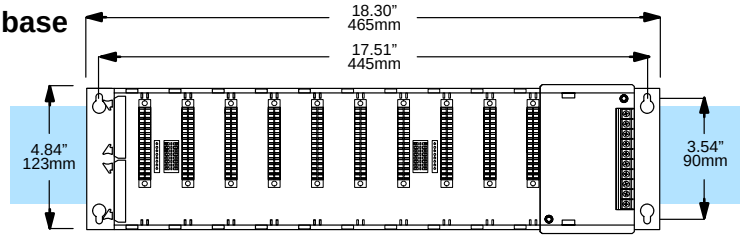
5 slot base



8 slot base



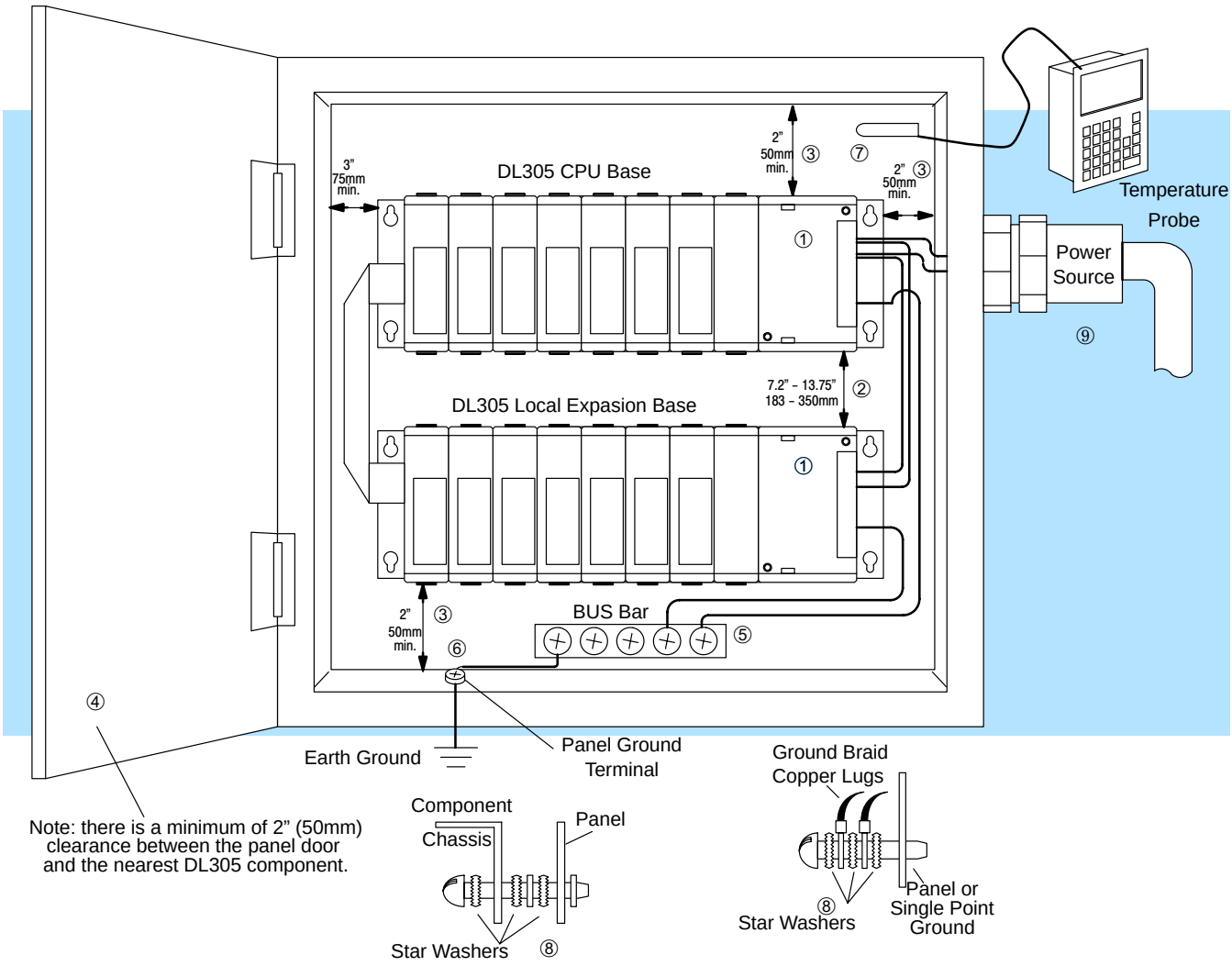
10 slot base



Panel Mounting and Layout

It is important to design your panel properly to help ensure the DL305 products operate within their environmental and electrical limits. The system installation should comply with all appropriate electrical codes and standards. It is important the system also conforms to the operating standards for the application to insure proper performance.

1. Mount the bases horizontally to provide proper ventilation.
2. If you place more than one base in a cabinet, there should be a minimum of 7.2” (183mm) between bases.
3. Provide a minimum clearance of 2” (50mm) between the base and all sides of the cabinet. There should also be at least 1.2” (30mm) of clearance between the base and any wiring ducts.
4. There must be a minimum of 2” (50mm) clearance between the panel door and the nearest DL305 component.



5. The ground terminal on the DL305 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. Remove anodized finishes and use copper lugs and star washers at termination points. A general rule is to achieve a 0.1 ohm of DC resistance between the DL305 base and the single point ground.
6. 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. For this connection you should use #12 AWG stranded copper wire as a minimum. Minimum wire sizes, color coding, and general safety practices should comply with appropriate electrical codes and standards for your region.

   A good common ground reference (Earth ground) is essential for proper operation of the DL305. 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. Properly evaluate any installations where the ambient temperature may approach the lower or upper limits of the specifications. Place a temperature probe in the panel, close the door and operate the system until the ambient temperature has stabilized. If the ambient temperature is not within the operating specification for the DL305 system, measures such as installing a cooling/heating source must be taken to get the ambient temperature within the DL305 operating specifications.
8. Device mounting bolts and ground braid termination bolts should be #10 copper bolts or equivalent. Tapped holes instead of nut–bolt arrangements should be used whenever possible. To assure good contact on termination areas impediments such as paint, coating or corrosion should be removed in the area of contact.
9. The DL305 system is designed to be powered by 110/220 VAC, or 24 VDC normally available throughout an industrial environment. Isolation transformers and noise suppression devices are not normally necessary, but may be helpful in eliminating/reducing suspect power problems.

Enclosures

Your selection of a proper enclosure is important to ensure safe and proper operation of your DL305 system. Applications of DL305 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

Environmental Specifications

The following table lists the environmental specifications that generally apply to the DL305 system (CPU, Bases, I/O Modules). The ranges that vary for the Handheld Programmer are noted at the bottom of this chart. I/O module operation may fluctuate depending on the ambient temperature and your application. Please refer to the appropriate I/O module specifications for the temperature derating curves applying to specific modules.

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

* 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° to70° C).
**Equipment will operate below 30% humidity. However, static electricity problems occur much more frequently at lower humidity levels. 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.

Power

The power source must be capable of supplying voltage and current complying with the base power supply specifications.

Specifications	D3–05B–1	D3–05BDC–1	D3–08B–1	D3–10B–1
Input Voltage Range	85–264 VAC 47–63Hz	20.5–30 VDC <10% ripple	85–264 VAC 47–63Hz	85–264 VAC 47–63Hz
Base Power Consumption	85 VA max	48 Watts	85 VA max	85 VA max
Inrush Current max.	30A	30A	30A	30A
Dielectric Strength	1500VAC for 1 minute between terminals of AC P/S, Run output, Common, 24VDC	1500VAC for 1 minute between 24VDC input terminals and Run output	1500VAC for 1 minute between terminals of AC P/S, Run output, Common, 24VDC	2000VAC for 1 minute between terminals of AC P/S, Run output, Common, 24VDC
Insulation Resistance	>10MΩ at 500VDC	>10MΩ at 500VDC	>10MΩ at 500VDC	>10MΩ at 500VDC
Power Supply Output (Voltage Ranges and Ripple)	(5VDC) 4.75–5.25V less than 0.25V p–p (9VDC) 8.0–10.0V less than 0.45 V p–p (24VDC) 20–28V less than 1.2V p–p	(5VDC) 4.75–5.25V less than 0.25V p–p (9VDC) 8.5–13.5V less than 0.45 V p–p (24VDC) 20–28V less than 1.2V p–p	(5VDC) 4.75–5.25V less than 0.25V p–p (9VDC) 8.0–10.0V less than 0.45 V p–p (24VDC) 20–28V less than 1.2V p–p	(5VDC) 4.75–5.25V less than 0.25V p–p (9VDC) 8.0–10.0V less than 0.45 V p–p (24VDC) 20–28V less than 1.2V p–p

Agency Approvals

Some applications require agency approvals. Typical agency approvals which your application may require are:

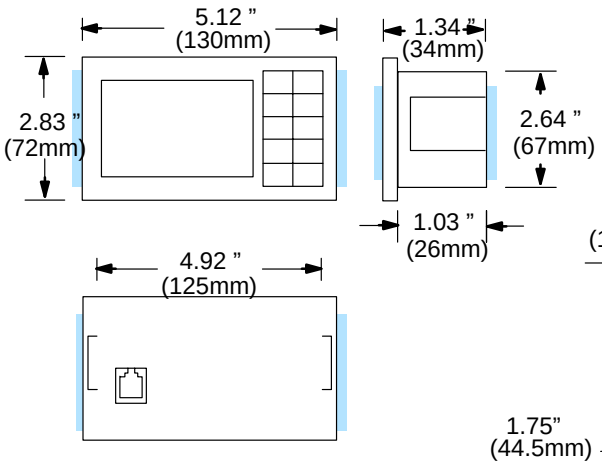
- UL (Underwriters' Laboratories, Inc.)
- CSA (Canadian Standards Association)
- FM (Factory Mutual Research Corporation)
- CUL (Canadian Underwriters' Laboratories, Inc.)

Component Dimensions

Before installing your PLC system you will need to know the dimensions for the components in your system. The diagrams on the following pages provide the component dimensions and should be used to define your enclosure specifications. Remember to leave room for potential expansion. Appendix E provides the weights for each component.

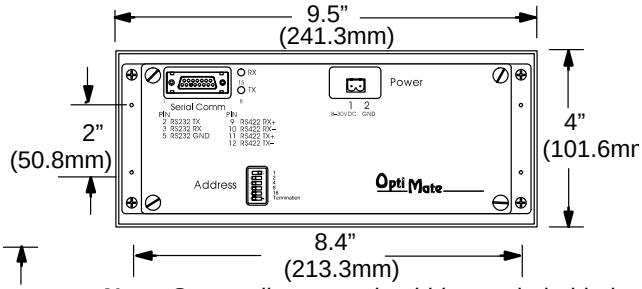
NOTE: If you are using other components in your system, make sure you refer to the appropriate manual to determine how those units can affect mounting dimensions.

*Direct*VIEW 1000

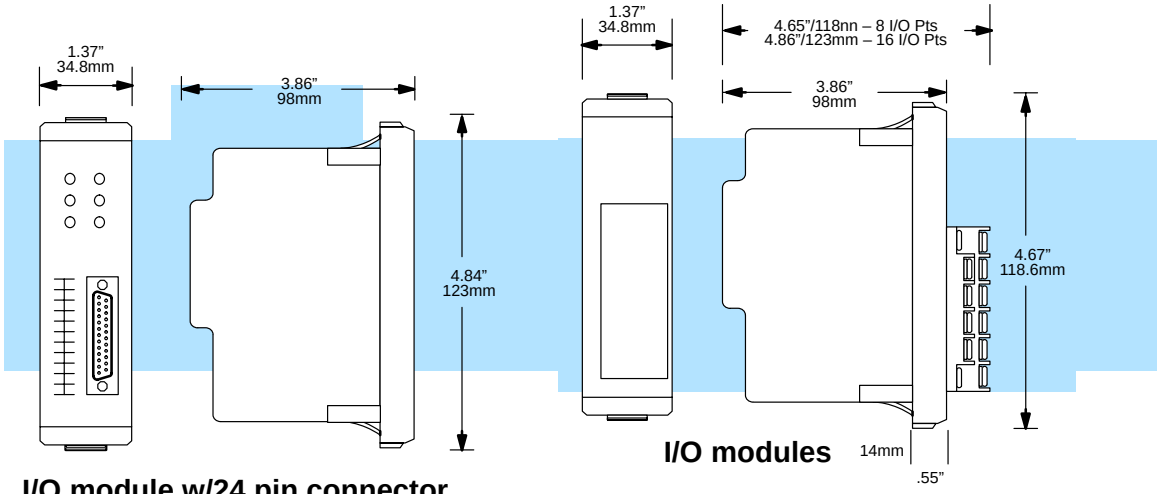


Optimation Units

(Large panel rear view shown)

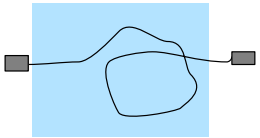


Note: Space allowance should be made behind the panel for the serial cable, and power connector. If you will be adding or removing panels for a multi-drop, then you may want to allow for hand room to reach the address switch on the back. We recommend 4 inches.

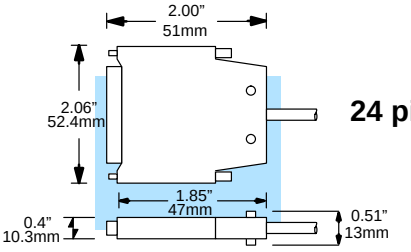


I/O module w/24 pin connector

I/O modules



Handheld programmer cable



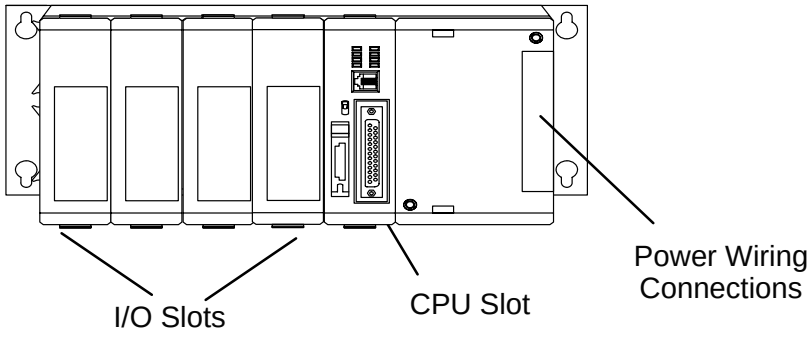
24 pin connector

Installing DL305 Bases

Choosing the Base Type

The DL305 system offers three different sizes of bases and two different power supply options.

The following diagram shows an example of a 5-slot base.

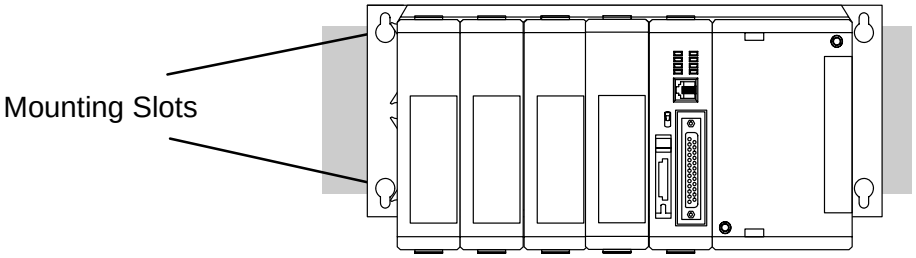


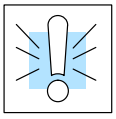
Your choice of base depends on three things.

- Number of I/O modules required
- Input power requirement (AC or DC power)
- Available power budget

Mounting the Base

All I/O configurations of the DL305 may use any of the base configurations. The bases are secured to the equipment panel or mounting location using four M4 screws in the corner mounting cut–outs of the base. The full mounting dimensions are given in the previous section on Mounting Guidelines.

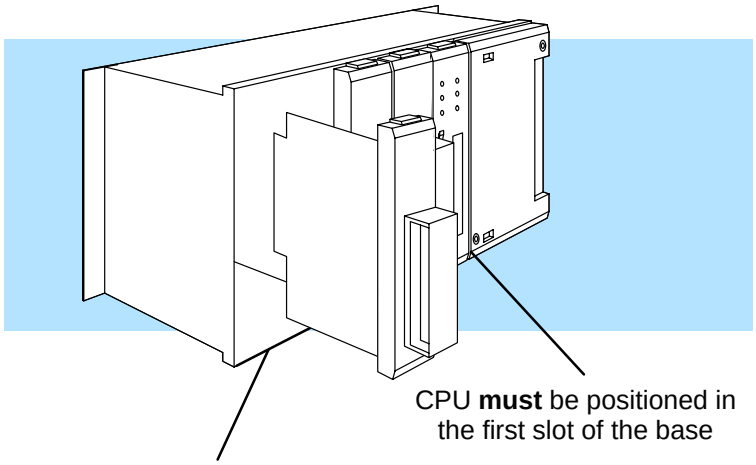




WARNING: To minimize the risk of electrical shock, personal injury, or equipment damage, always disconnect the system power before installing or removing any system component.

Installing Components in the Base

When inserting components into the base, align the PC board(s) of the module with the grooves on the top and bottom of the base. Push the module straight into the base until it is firmly seated in the backplane connector. Once the module is inserted into the base, push in the retaining clips (located at the top and bottom of the module) to firmly secure the module to the base.



WARNING: Minimize the risk of electrical shock, personal injury, or equipment damage, always disconnect the system power before installing or removing any system component.

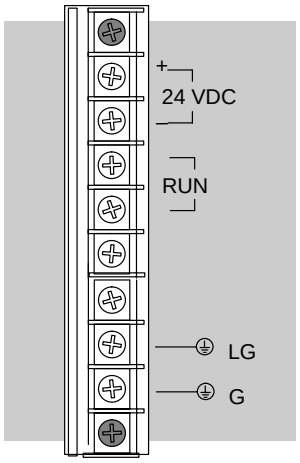
Base Wiring Guidelines

Base Wiring

The diagram shows the terminal connections located on the power supply of the DL305 xxxxx–1 bases. The base terminals can accept up to 16 AWG.

NOTE: You can connect either a 115 VAC or 220 VAC supply to the AC terminals. Special wiring or jumpers are not required as with some of the other ***Direct***LOGIC™ products.

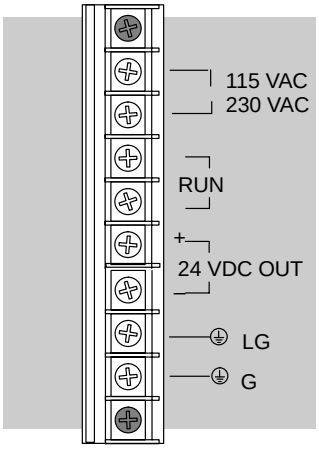
24 VDC Base Terminal Strip



Logic Ground

Frame Ground

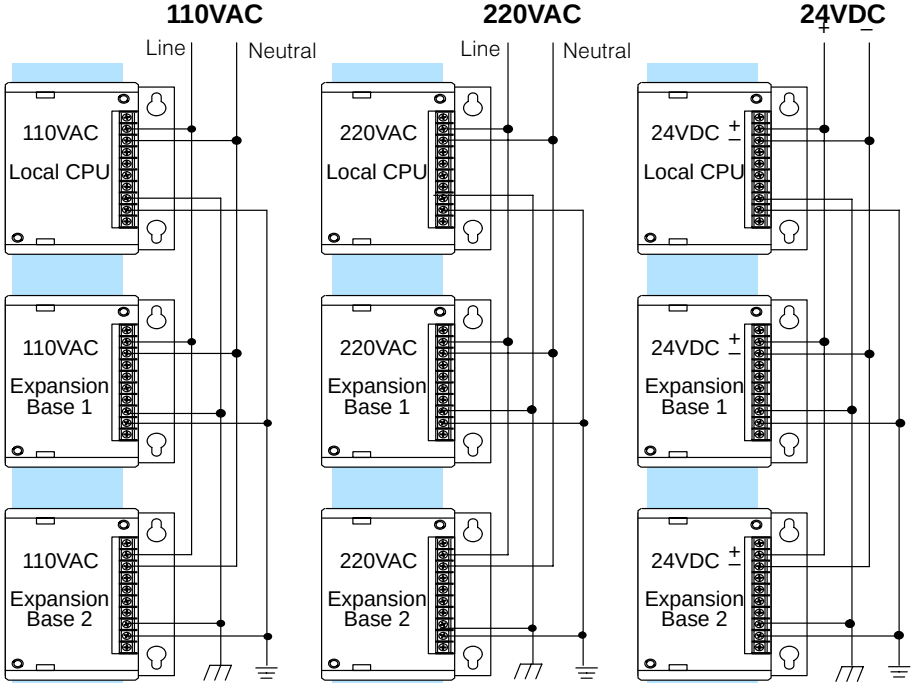
110/220 VAC Base Terminal Strip



WARNING: Once the power wiring is connected, install the plastic protective cover. When the cover is removed there is a risk of electrical shock if you accidentally touch the wiring or wiring terminals.

Expansion Base Wiring

The following example illustrates connections when using Expansion bases.

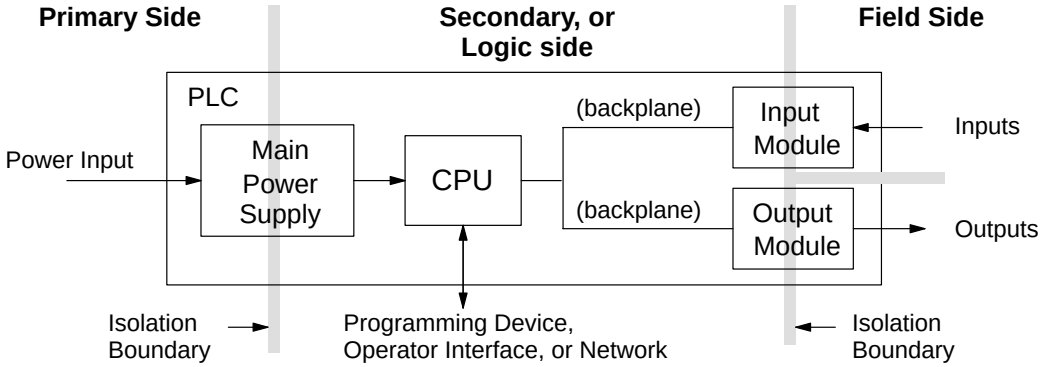


I/O Wiring Strategies

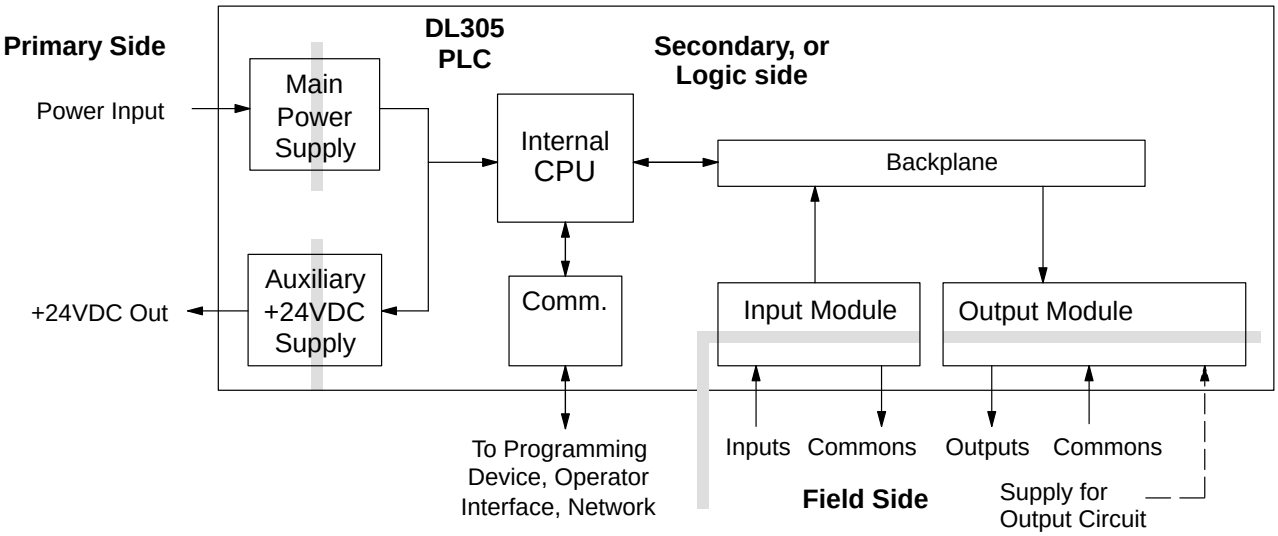
The DL305 PLC system is very flexible and will work in many different wiring configurations. By studying this section before actual installation, you can probably find the best wiring strategy for your application . This will help to lower system cost, wiring errors, and avoid safety problems.

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 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 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.*



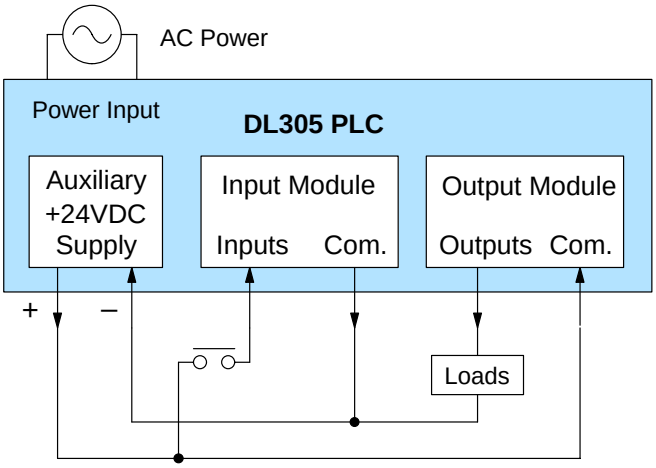
The next figure shows the physical layout of a DL305 PLC system, as viewed from the front. In addition to the basic circuits covered above, AC-powered bases include an auxiliary +24VDC power supply with its own isolation boundary. Since the supply output is isolated from the other three circuits, it can power input and/or output circuits!



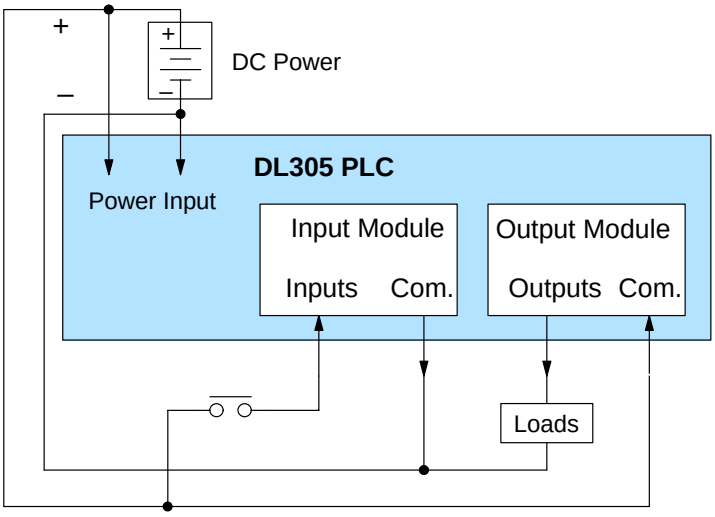
In some cases, using the built-in auxiliary +24VDC supply can result in a cost savings for your control system. It can power combined loads up to 100 mA. Be careful not to exceed the current rating of the supply. If you are the system designer for your application, you may be able to select and design in field devices which can use the +24VDC auxiliary supply.

Powering I/O Circuits with the Auxiliary Supply

All AC powered DL305 bases feature the internal auxiliary supply. If input devices AND output loads need +24VDC power, the auxiliary supply may be able to power both circuits as shown in the following diagram.



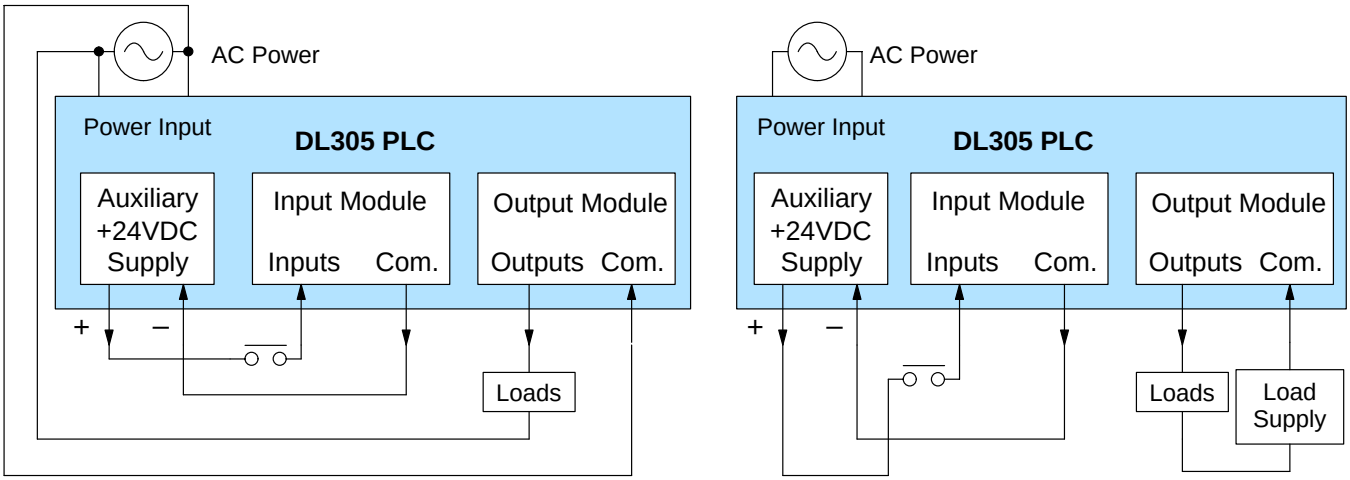
DC-powered DL305 bases are designed for application environments in which low-voltage DC power is more readily available than AC. These include a wide range of battery–powered applications, such as remotely-located control, in vehicles, portable machines, etc. For this application type, all input devices and output loads typically use the same DC power source. Typical wiring for DC-powered applications is shown in the following diagram.



Powering I/O Circuits Using Separate Supplies

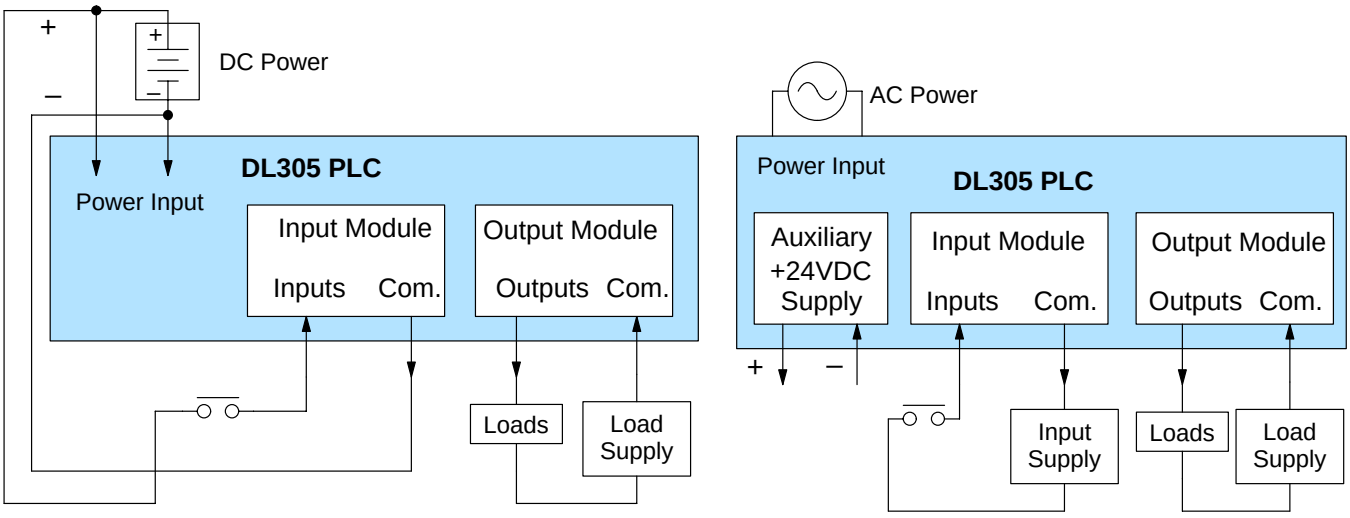
In most applications it will be necessary to power the input devices from one power source, and to power output loads from another source. Loads often require high-energy AC power, while input sensors use low-energy DC. If a machine operator is likely to come in close contact with input wiring, then safety reasons also require isolation from high-energy output circuits. It is most convenient if the loads can use the same power source as the PLC, and the input sensors can use the auxiliary supply, as shown to the left in the figure below.

If the loads cannot be powered from the PLC supply, then a separate supply must be used as shown to the right in the figure below.



Some applications will use the PLC external power source to also power the input circuit. This typically occurs on DC-powered PLCs, as shown in the drawing below to the left. The inputs share the PLC power source supply, while the outputs have their own separate supply.

A worst-case scenario, from a cost and complexity view-point, is an application which requires separate power sources for the PLC, input devices, and output loads. The example wiring diagram below on the right shows how this can work, but also the auxiliary supply output is an unused resource. You will want to avoid this situation if possible.



Sinking / Sourcing Concepts

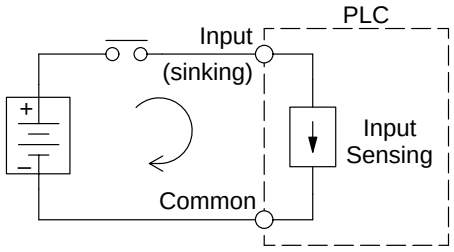
Before going further in the study of wiring strategies, you must have a solid understanding of "*sinking*" and "*sourcing*" concepts. Use of these terms occurs frequently in input or output circuit discussions. It is the goal of this section to make these concepts easy to understand, further ensuring your success in installation. First the following short definitions are provided, followed by practical applications.

Sinking = provides a path to supply **ground (–)**

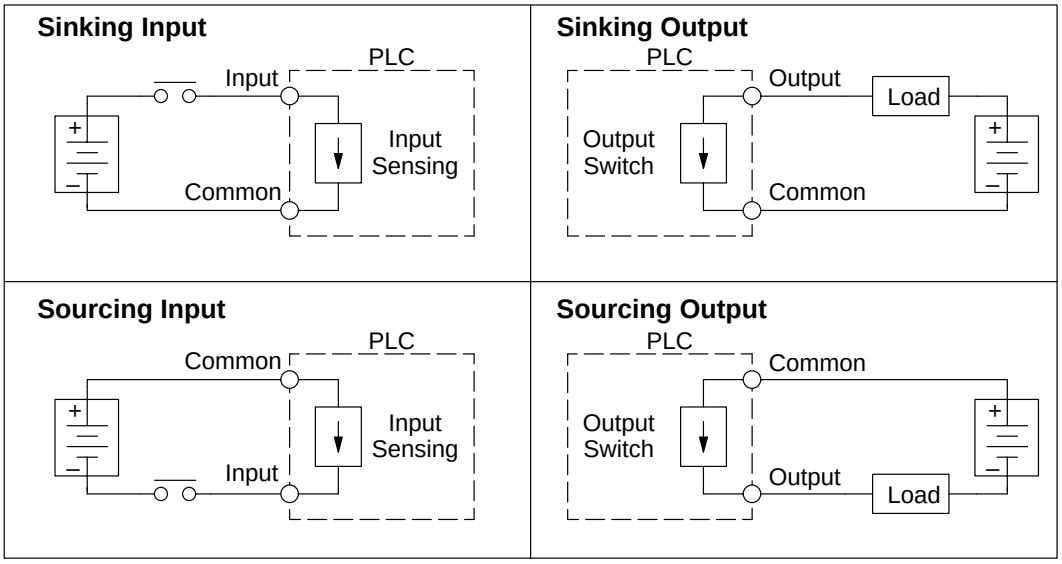
Sourcing = provides a path to supply **source (+)**

First you will notice these are only associated with DC circuits and not AC, because of the reference to (+) and (–) polarities. Therefore, *sinking and sourcing terminology only applies to DC input and output circuits.* Input and output points that are sinking or sourcing *only* 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, you can successfully connect the supply and field device every time by understanding "sourcing" and "sinking".

For example, the figure to the right depicts a "sinking" input. To properly connect the external supply, you will have to connect it so the input *provides a path to ground (–)*. 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, the circuit has been completed . Current flows in the direction of the arrow when the switch is closed.

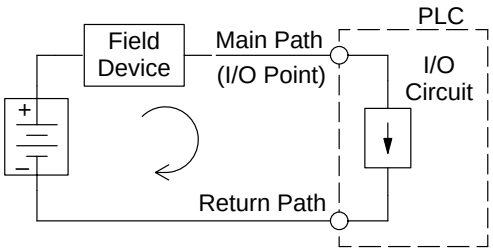


By applying the circuit principle above to the four possible combinations of input/output sinking/sourcing types as shown below. The I/O module specifications at the end of this chapter list the input or output type.

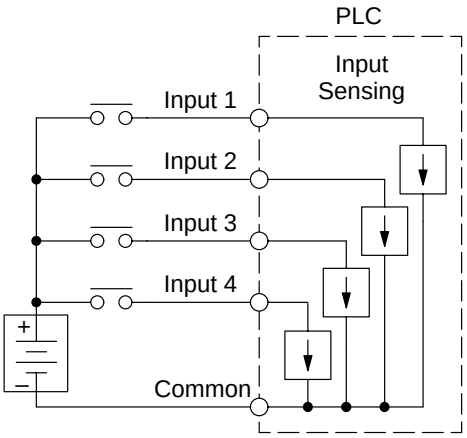


I/O "Common" Terminal Concepts

In order for a PLC I/O circuit to operate, current must enter at one terminal and exit at another. Therefore, 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.



If there was unlimited space and budget for I/O terminals, every I/O point could have two dedicated terminals as the figure above shows. However, providing this level of flexibility is not practical or even necessary for most applications. Therefore, most Input or Output points on PLCs are in groups which share the return path (called *commons*). 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.



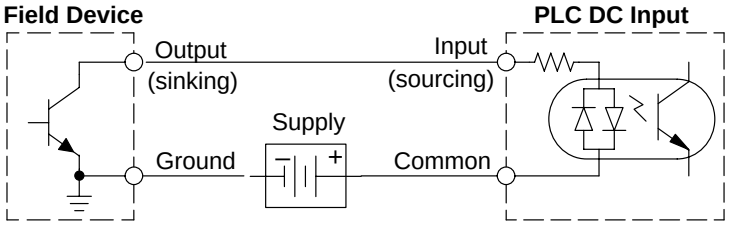
NOTE: In the circuit above, the current in the common path is 4 times any channel's input current when all inputs are energized. This is especially important in output circuits, where heavier gauge wire is sometimes necessary on commons.

Connecting DC I/O to "Solid State" Field Devices

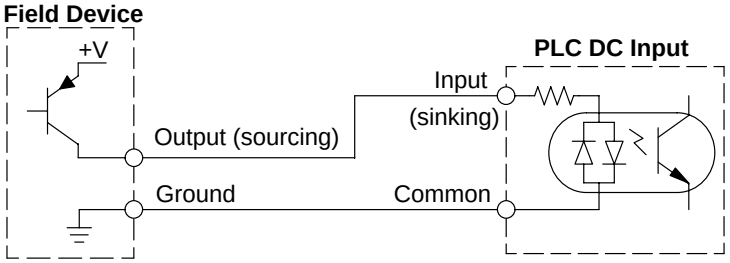
In the previous section on Sourcing and Sinking concepts, the DC I/O circuits were explained to sometimes will only allow current to flow one way. This is also true for many of the field devices which have solid-state (transistor) interfaces. In other words, field devices can also be sourcing or sinking. *When connecting two devices in a series DC circuit, one must be wired as sourcing and the other as sinking*.

Solid State Input Sensors

Several DL305 DC input modules are flexible because 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 +24 auxiliary supply or another supply (+12 VDC or +24VDC), as long as the input specifications are met.



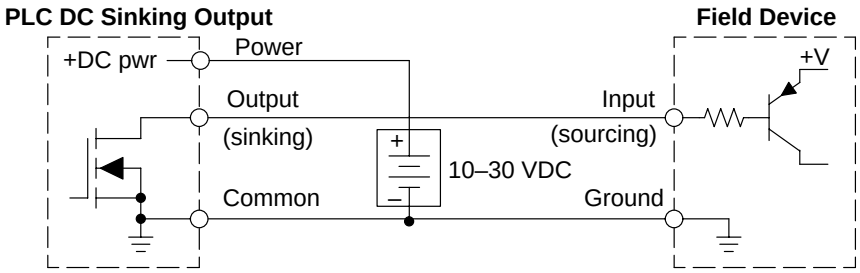
In the next circuit, a field device has an open-emitter PNP transistor output. It sources current to the PLC input point, which sinks the current back to ground. Since the field device is sourcing current, no additional power supply is required.



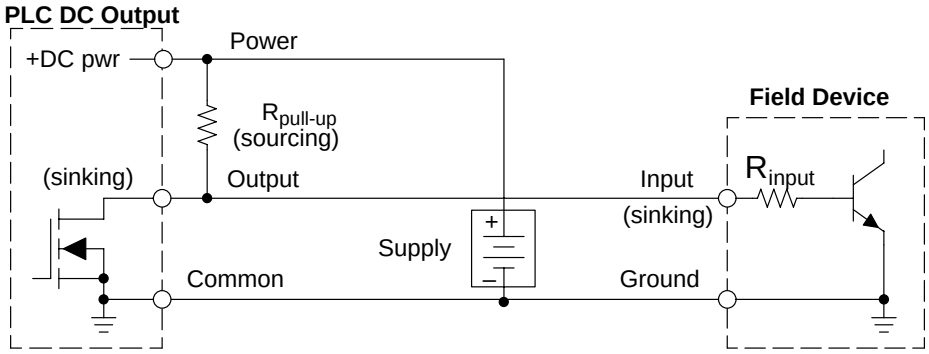
Solid State Output Loads

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 control signal, not to send DC power to an actuator.

Several of the DL305 DC output modules are the sinking type. This means that each DC output provides a path to ground when it is energized. 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 a PLC sinking DC output point is connected to the sinking input of a field device. This is tricky, because both the PLC output and field device input are sinking type. Since the circuit must have one sourcing and one sinking device, a sourcing capability needs to be added to the PLC output by using a pull-up resistor. In the circuit below, a $R_{pull\text{-}up}$ is connected from the output to the DC output circuit power input.



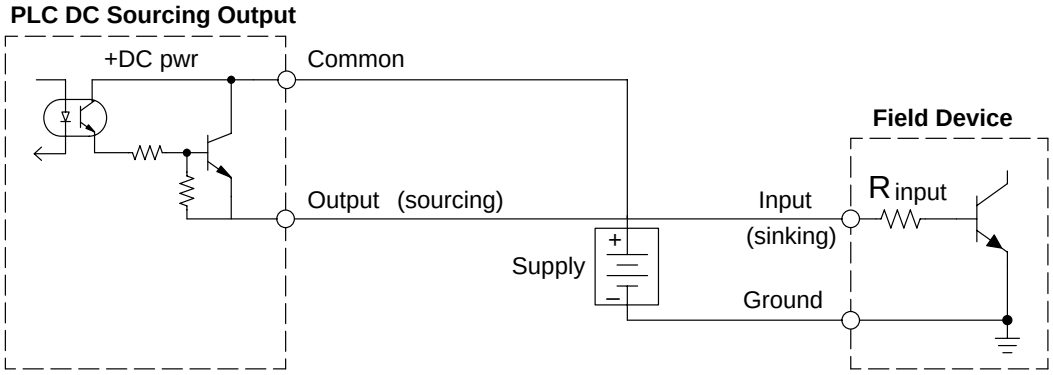
NOTE 1: DO NOT attempt to drive a heavy load (>25 mA) with this pull-up method
NOTE 2: Using the pull-up resistor to implement a sourcing output has the effect of inverting the output point logic. In other words, the field device input is energized when the PLC output is OFF, from a ladder logic point-of-view. 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 $R_{pull\text{-}up}$. In order to do so, you 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 $R_{pull\text{-}up}$. Then calculate the power $P_{pull\text{-}up}$ (in watts), in order to size $R_{pull\text{-}up}$ properly.

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

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

Of course, the easiest way to drive a sinking input field device as shown below is to use a DC sourcing output module. The Darlington NPN stage will have about 1.5 V ON-state saturation, but this is not a problem with low-current solid-state loads.



Relay Output Guidelines

Four output modules in the DL305 I/O family feature relay outputs: D3–08TR, F3–08TRS–1, F3–08TRS–2, D3–16TR. Relays are best for the following applications:

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

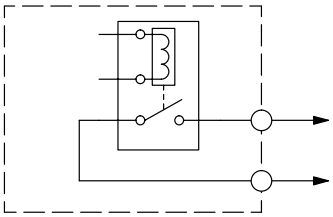
Some applications in which NOT to use relays:

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

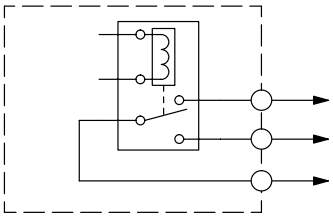
Relay outputs in the DL305 output modules are available in two contact arrangements, shown to the right. The Form A type, or SPST (single pole, single throw) type is normally open and is the simplest to use. The Form C type, or SPDT (single pole, double throw) type has a center contact which moves and a stationary contact on either side. This provides a normally closed contact and a normally open contact.

Some relay output module's relays share common terminals, which connect to the wiper contact in each relay of the bank. Other relay modules have relays which are completely isolated from each other. In all cases, the module drives the relay coil when the corresponding output point is on.

Relay with Form A contacts



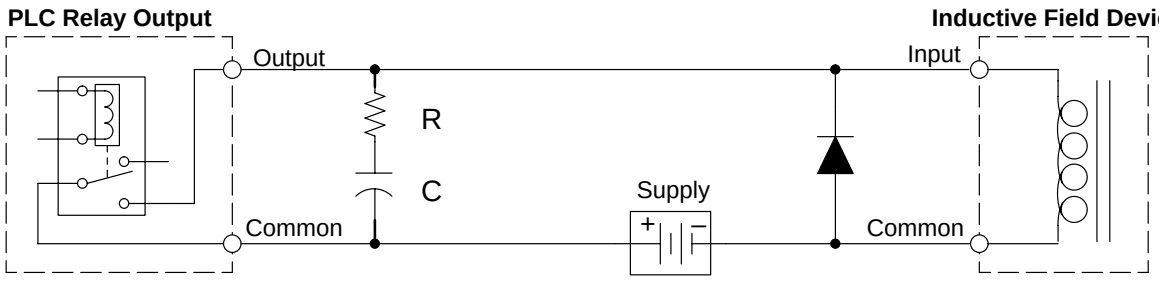
Relay with Form C contacts



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. However, there are some steps you can take to help prolong the life of relay contacts:

- Switch the relay on or off only when the application requires it.
- If you have the option, switch the load on or off at a time when it will draw the least current.
- Take measures to suppress inductive voltage spikes from inductive DC loads such as contactors and solenoids (circuit given below).



Adding external contact protection may extend relay life beyond the number of contact cycles listed in the specification tables for relay modules. High current inductive loads such as clutches, brakes, motors, direct-acting solenoid valves, and motor starters will benefit the most from external contact protection.

The RC network must be located close to the relay module output connector. To find the values for the RC snubber network, first determine the voltage across the contacts when open, and the current through them when closed. If the load supply is AC, then convert the current and voltage values to peak values:

Now you are ready to calculate values for R and C, according to the formulas:

$$C\ (\mu F) = \frac{I^2}{10} \qquad R\ (\Omega) = \frac{V}{10 \times I^x}\ \text{, where } x = 1 + \frac{50}{V}$$

C minimum = 0.001 μF, the voltage rating of C must be ≥ V, non-polarized

R minimum = 0.5 Ω, 1/2 W, tolerance is ± 5%

For example, suppose a relay contact drives a load at 120VAC, 1/2 A. Since this example has an AC power source, first calculate the peak values:

$$I_{peak} = I_{rms} \times 1.414, = 0.5 \times 1.414 = 0.707 \text{ Amperes}$$

$$V_{peak} = V_{rms} \times 1.414 = 120 \times 1.414 = 169.7 \text{ Volts}$$

Now, finding the values of R and C,:

$$C\ (\mu F) = \frac{I^2}{10} = \frac{0.707^2}{10} = 0.05\ \mu F, \text{ voltage rating} \geq 170 \text{ Volts}$$

$$R\ (\Omega) = \frac{V}{10 \times I^x}\ \text{, where } x = 1 + \frac{50}{V}$$

$$x = 1 + \frac{50}{169.7} = 1.29 \qquad R\ (\Omega) = \frac{169.7}{10 \times 0.707^{1.29}} = 26\ \Omega, 1/2\ W, \pm 5\%$$

If the contact is switching a DC inductive load, add a diode across the load as near to load coil as possible. 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 as the contacts are opening.

For best results, follow these guidelines in using a noise suppression diode:

- DO NOT use this circuit with an AC power supply.
- 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.

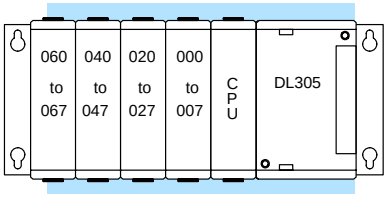
I/O Modules Position, Wiring, and Specification

Slot Numbering

The DL305 bases each provide different numbers of slots for use with the I/O modules. You may notice the bases refer to 5-slot, 8-slot, etc. One of the slots is dedicated to the CPU, so you always have one less I/O slot. For example, you have four I/O slots with a 5-slot base. The I/O slots are numbered 0 – 3. The CPU slot always contains a CPU and is not numbered.

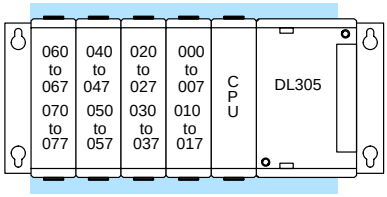
The examples below show the I/O numbering for a 5 slot local CPU base with 8 point I/O and a 5 slot local CPU base with 16 point I/O using the xxxxx–1 bases.

5 Slot Base Using 8 Point I/O Modules



Slot Number: 3—2—1—0

5 Slot Base Using 16 Point I/O Modules



Slot Number: 3—2—1—0

I/O Module Placement Rules

There are some limitations that determine where you can place certain types of modules. Some modules require certain locations and may limit the number or placement of other modules. If you have difficulty with some of the explanations, please look ahead to the illustrations in this chapter. They should clear up any gray areas in the explanation and you will probably find the configuration you intend to use in your installation.

In all of the configurations mentioned the number of slots from the CPU that are to be used can roll over into an expansion base if necessary. For example if a rule states a module must reside in one of the six slots adjacent to the CPU, and the system configuration is comprised of two 5 slot bases, slots 1 and 2 of the expansion base are valid locations.

The following table provides the general placement rules for the DL305 components.

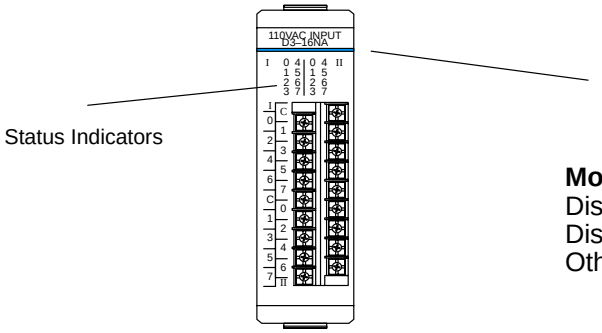
Module	Restriction
CPU	The CPU must reside in the first slot of the local CPU base. The first slot is the closest slot to the power supply.
16 Point I/O Modules	Any slot.
Analog Modules	Analog modules must reside in any valid 16 point I/O slot.
ASCII Basic Modules	ASCII Basic modules must reside in any valid 16 point I/O slot.
High Speed Counter	The D3–350 CPU does not support a high speed counter module.

Discrete Module Status Indicators

The discrete modules provide LED status indicators to show status of input points.

Color Coding of I/O Modules

The DL305 family of I/O modules have a color coding scheme to help you quickly identify if a module is either an input module, output module, or a specialty module. This is done through a color bar indicator located on the front of each module. The color scheme is listed below:



Module Type	Color Code
Discrete/Analog Output	Red
Discrete/Analog Input	Blue
Other	White

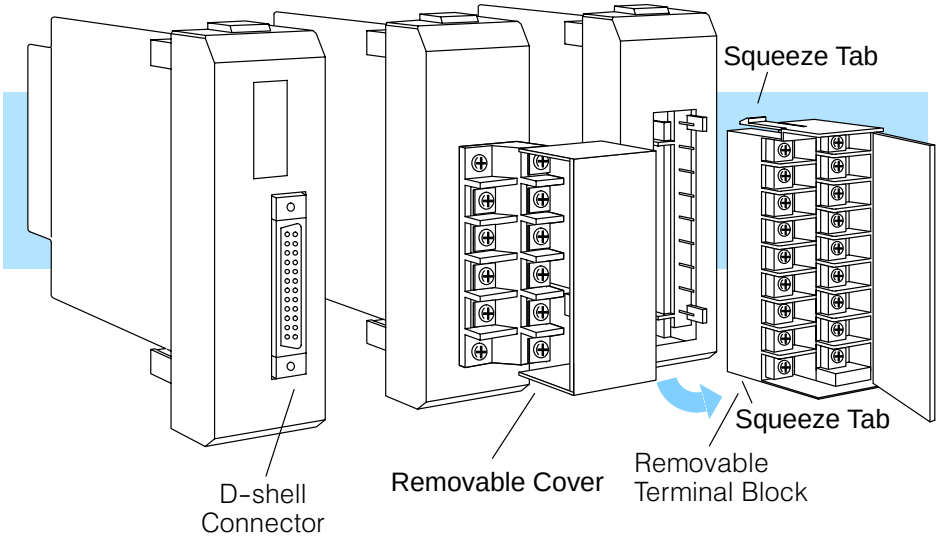
Wiring the Different Module Connectors

There are three types of module connectors for the DL305 I/O. Some modules have normal screw terminal connectors. Other modules have connectors with recessed screws. The recessed screws help minimize the risk of someone accidentally touching active wiring. The third type has a D–shell connector for special cable connections.

Both types of screw connectors can be easily removed. If you examine the connectors closely, you will notice there are squeeze tabs on the top and bottom. To remove the terminal block, press the squeeze tabs and pull the terminal block away from the module.

We also have DIN rail mounted terminal blocks, DINnectors (refer to our catalog for a complete listing of all available products). The DINnectors come with special pre–assembled cables with the I/O connectors installed and wired.

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



I/O Wiring Checklist

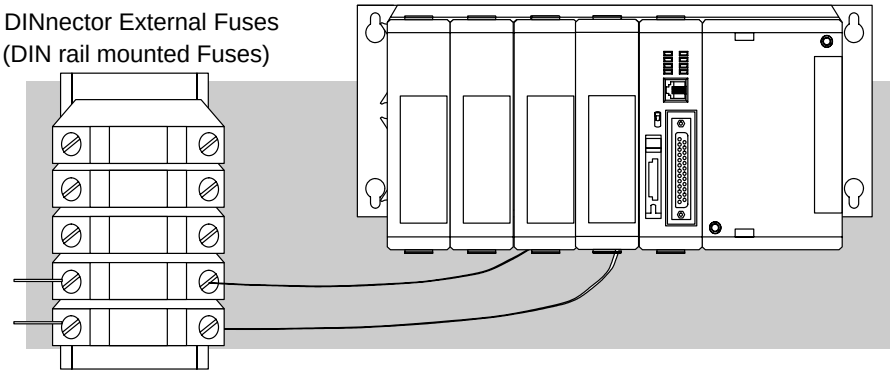
Use the following guidelines when wiring the I/O modules in your system.

1. There is a limit to the size of wire the modules can accept. The table below lists the maximum AWG for each module type. Smaller AWG is acceptable to use for each of the modules.

Module type	Maximum AWG
8 point	12 AWG
16 point	16 AWG

2. Always use a continuous length of wire, do not combine 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. To reduce the risk of having a module with a blown fuse, we suggest you add external fuses to your I/O wiring. A fast blow fuse, with a lower current rating than the I/O module fuse can be added 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 our catalog for a complete line of DINnectors, DIN rail mounted fuse blocks.

DINnector External Fuses
(DIN rail mounted Fuses)



NOTE: For modules which have soldered or non-replaceable fuses, we recommend you return your module to us and let us replace your blown fuse(s) since disassembling the module will void your warranty.

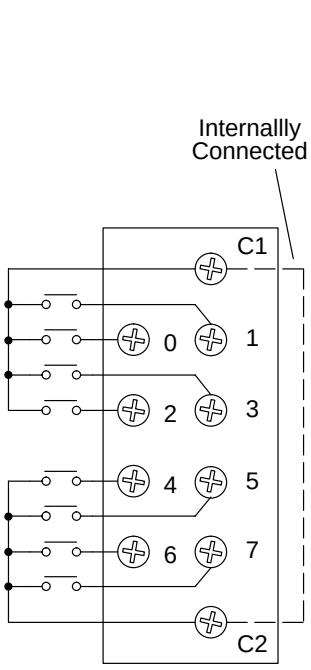
Glossary of Specification Terms

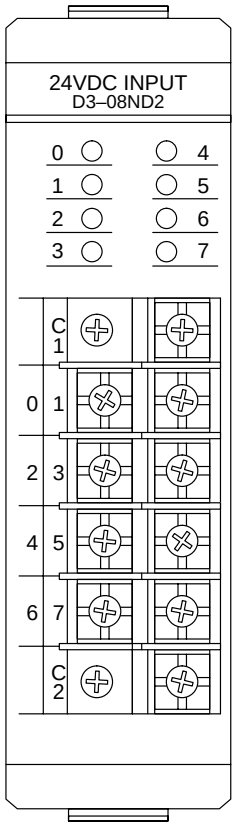
Inputs or Outputs Per Module	Indicates number of input or output points per module and designates current sinking, current sourcing, or either.
Commons Per Module	Number of commons per module and their electrical characteristics.
Input Voltage Range	The operating voltage range of the input circuit.
Output Voltage Range	The operating voltage range of the output circuit.
Peak Voltage	Maximum voltage allowed for the input circuit.
AC Frequency	AC modules are designed to operate within a specific frequency range.
ON Voltage Level	The voltage level at which the input point will turn ON.
OFF Voltage Level	The 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.
Minimum Load	The minimum load current for the output circuit to operate properly.
External DC Required	Some output modules require external power for the output circuitry.
ON Voltage Drop	Sometimes called “saturation voltage”, it is the voltage measured from an output point to its common terminal when the output is ON at max. load.
Maximum Leakage Current	The maximum current a connected maximum load will receive when the output point is OFF.
Maximum Inrush Current	The maximum current used by a load for a short duration upon an OFF to ON transition of a output point. It is greater than the normal ON state current and is characteristic of inductive loads in AC circuits.
Base Power Required	Power from the base power supply is used by the DL305 input modules and varies between different modules. The guidelines for using module power is explained in the power budget configuration section in Chapter 4–5.

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.
Terminal Type	Indicates whether the terminal type is a removable or non-removable connector or a terminal.
Status Indicators	The LEDs that indicate the ON/OFF status of an input point. These LEDs are electrically located on either the logic side or the field device side of the input circuit.
Weight	Indicates the weight of the module. See Appendix E for a list of the weights for the various DL305 components.
Fuses	Protective device for an output circuit, which stops current flow when current exceeds the fuse rating. They may be replaceable or non–replaceable, or located externally or internally.

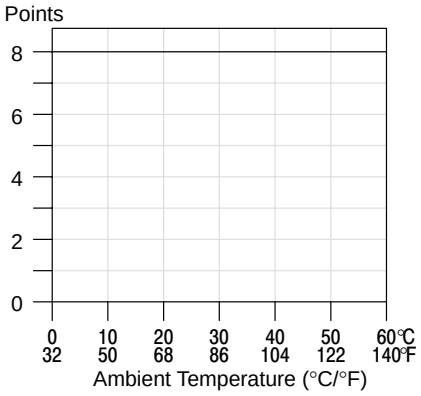
D3–08ND2, 24 VDC Input Module

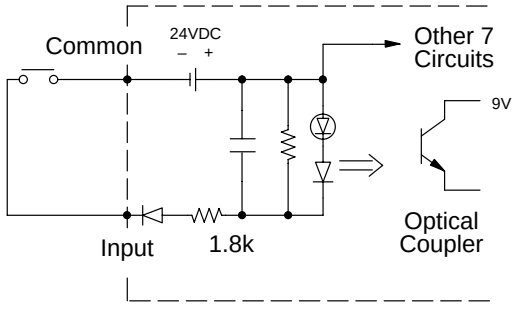
Inputs per module	8 (current sourcing)		Base power required	9V 10 mA Max 24V 14mA/ON pt. (112 mA Max)
Commons per module	2 (internally connected)			
Input voltage range	18–36VDC			
Input voltage	Internally supplied		OFF to ON response	4–15 ms
Peak voltage	40 VDC		ON to OFF response	4–15 ms
AC frequency	N/A		Terminal type	Non–removable
ON voltage level	< 3 V		Status indicators	Field side
OFF voltage level	>18 V		Weight	4.2 oz. (120 g)
Input impedance	1.8 K ohm			
Input current	12 mA Max			
Minimum ON current	7 mA			
Maximum OFF current	3 mA			





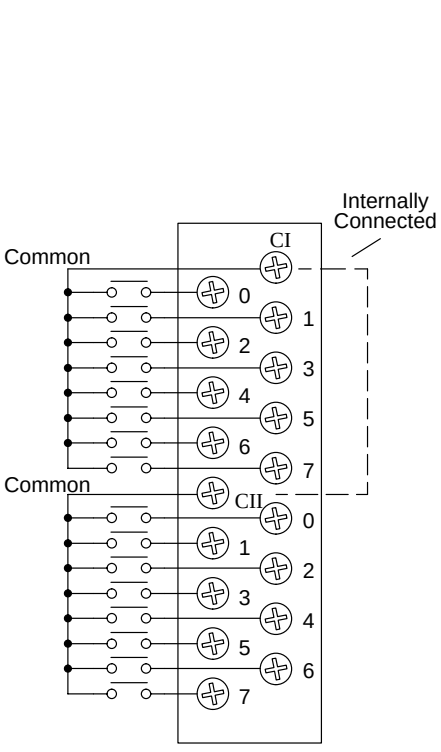
Derating Chart for D3–08ND2

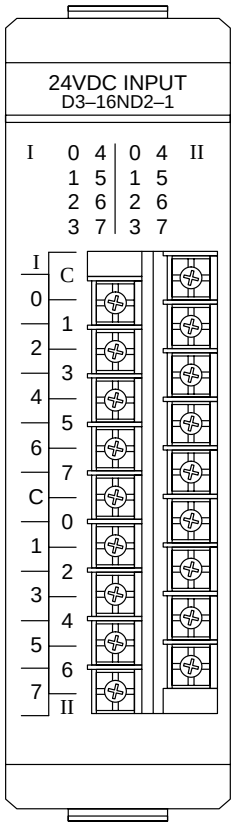




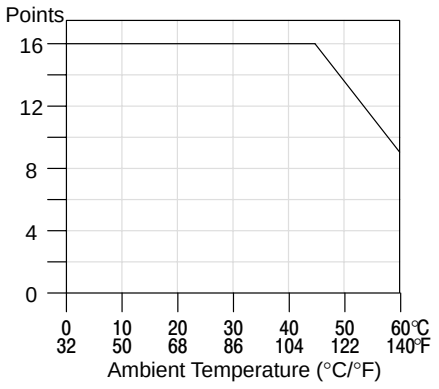
D3–16ND2–1, 24 VDC Input Module

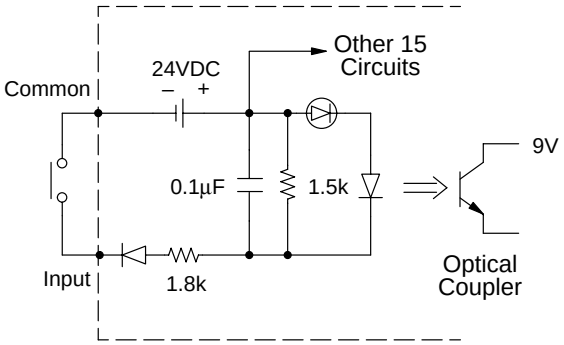
Inputs per module	16 (current sourcing)		Base power required	9V 25 mA Max 24V 14mA/ON pt. (224 mA Max)
Commons per module	2 (internally connected)			
Input voltage range	18–36VDC			
Input voltage	Internally supplied		OFF to ON response	3–15 ms
Peak voltage	36VDC		ON to OFF response	4–15 ms
AC frequency	N/A		Terminal type	Removable
ON voltage level	< 3V		Status indicators	Field side
OFF voltage level	>19 V		Weight	6.3 oz. (180 g)
Input impedance	1.8 K ohm			
Input current	20 mA Max			
Minimum ON current	5 mA			
Maximum OFF current	1 mA			





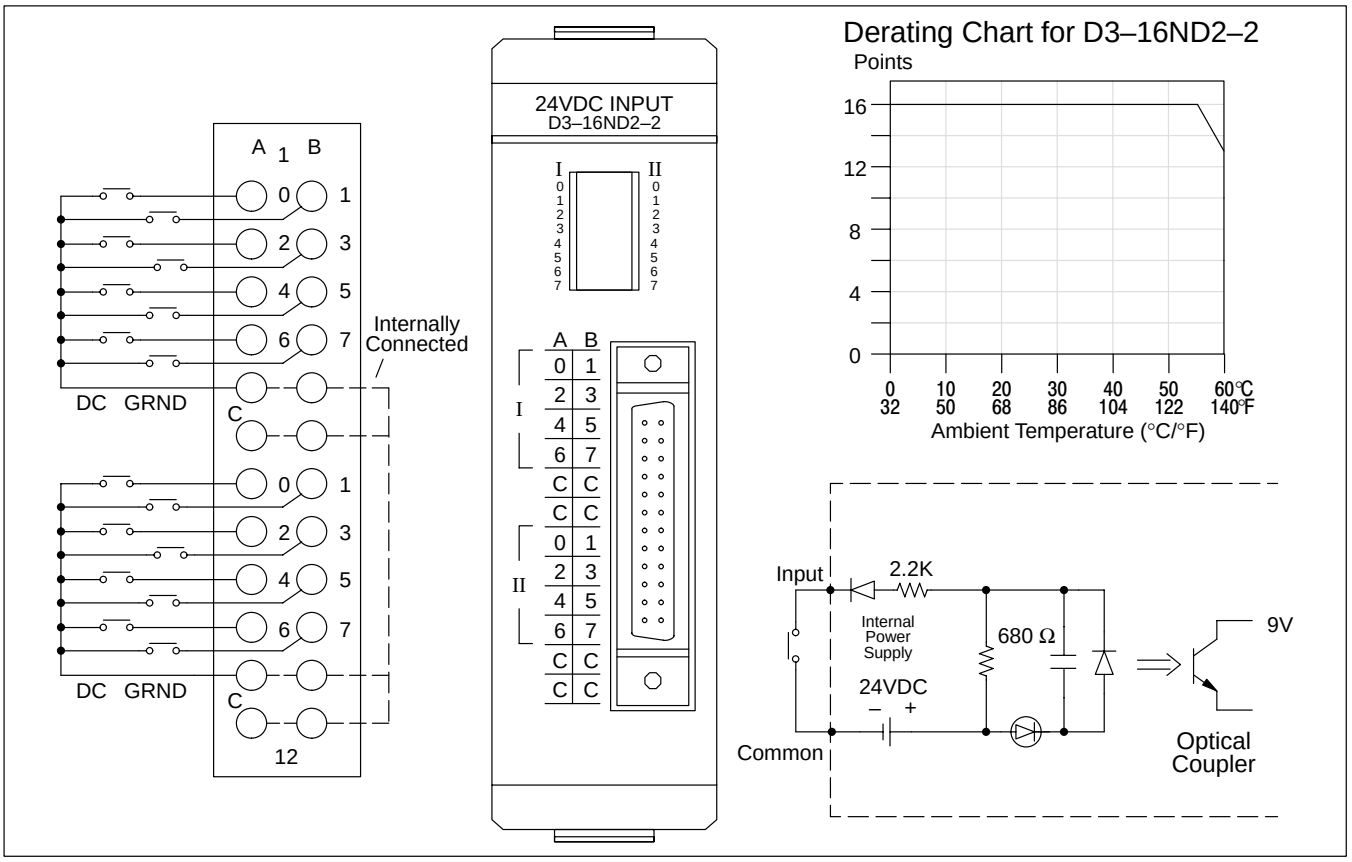
Derating Chart for D3–16ND2–1





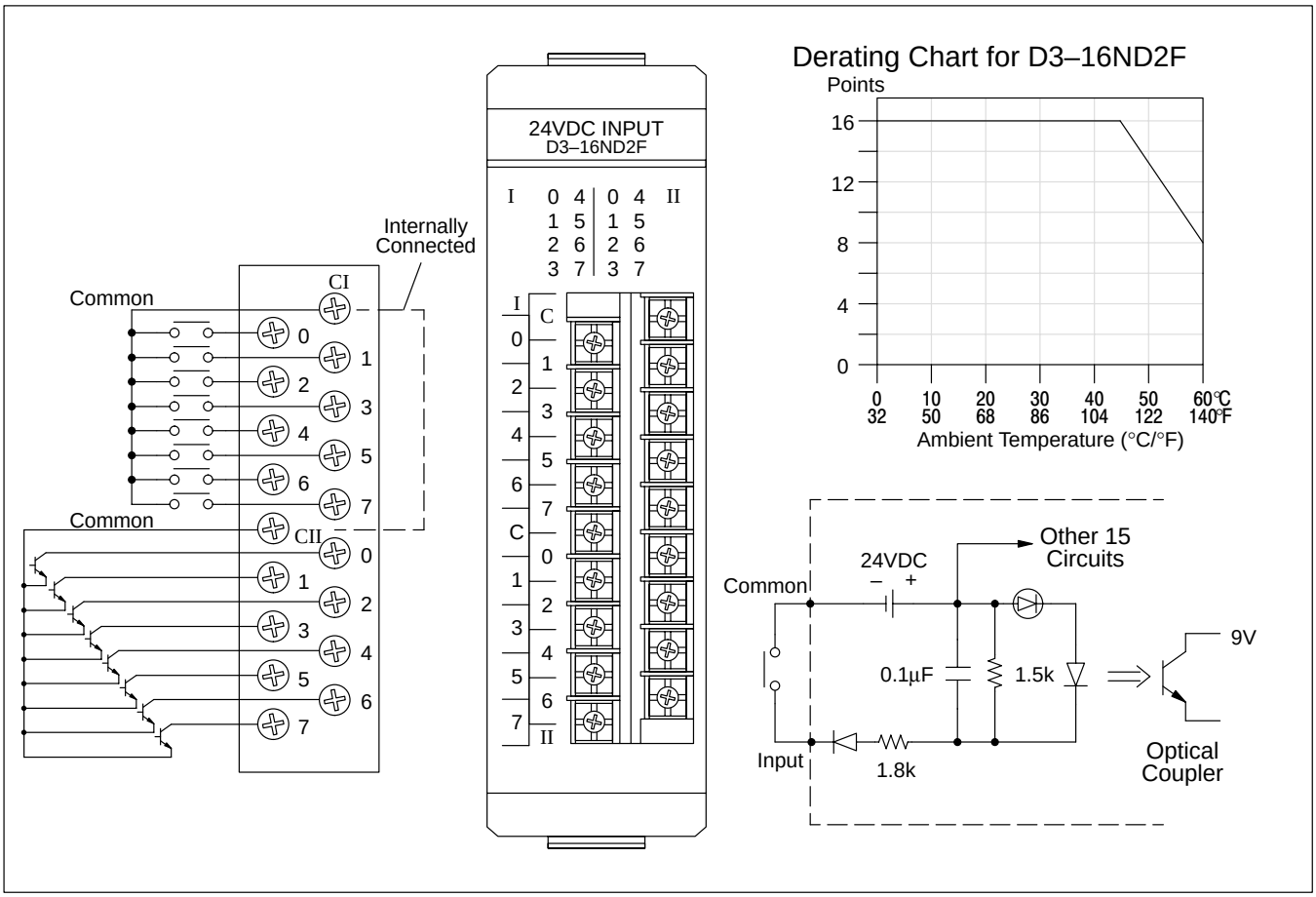
D3–16ND2–2, 24 VDC Input Module Module

Inputs per module	16 (current sourcing)		Base power required	9V 3mA+1.3mA/ON pt (24 mA Max) 24V 1mA+13mA/ON pt (209 mA Max)
Commons per module	8 internally connected			
Input voltage range	18–36 VDC			
Input voltage	Internally supplied			
Peak voltage	36 VDC		OFF to ON response	4–15 ms
AC frequency	N/A		ON to OFF response	4–15 ms
ON voltage level	< 3 V		Terminal type	24 Pin Removable connector
OFF voltage level	> 19 V			
Input impedance	2.2 K ohm		Status indicators	Field side
Input current	20 mA Max		Weight	5.3 oz. (150 g)
Minimum ON current	5 mA			
Maximum OFF current	2 mA			



D3–16ND2F, 24 VDC Fast Response Input Module

Inputs per module	16 (current sourcing)		Base power required	9V 25 mA Max 24V 14 mA/ON pt. (224 mA Max)
Commons per module	2 (internally connected)			
Input voltage range	18–36VDC			
Input voltage	Internally supplied		OFF to ON response	0.8 ms
Peak voltage	36VDC		ON to OFF response	0.8 ms
AC frequency	N/A		Terminal type	Removable
ON voltage level	< 13V		Status indicators	Field side
OFF voltage level	>19 V		Weight	6.3 oz. (180 g)
Input impedance	1.8 K ohm			
Input current	20 mA Max			
Minimum ON current	5 mA			
Maximum OFF current	1 mA			

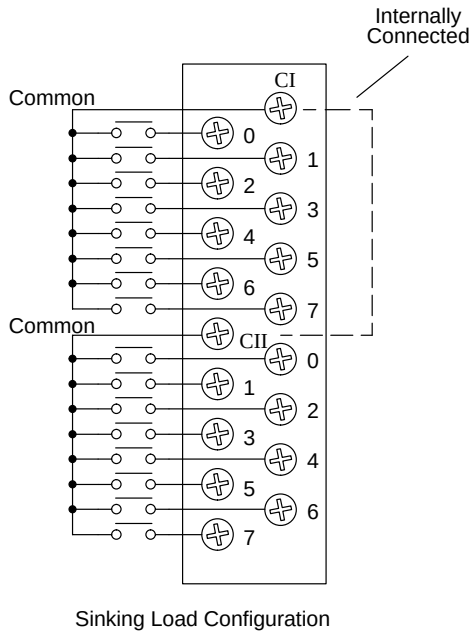


F3–16ND3F, TTL/24 VDC Fast Response Input Module

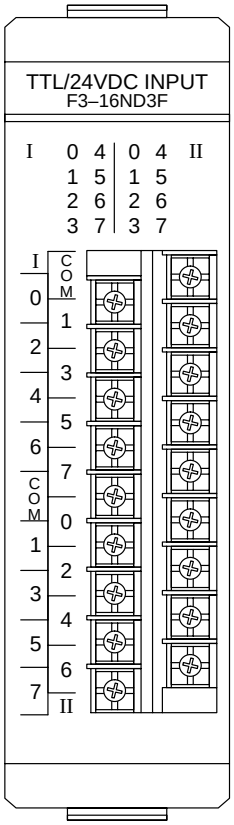
Inputs per module	16 sink/source (jumper selectable sink/source)*		Base power required	9V 148 mA Max 24V 68 mA Max
Commons per module	2 (non-isolated)		Input current	1 mA @ 5VDC 3 mA @ 12–24 DC
Input voltage range	5 VDC TTL & CMOS, 12–24 VDC (jumper selectable)*		Input impedance	4.7K
Input voltage supplied	Internal (used with sinking loads) External (used with sourcing loads)		OFF to ON response	1 ms
Peak voltage	100 VDC (35 VDC Continuous)		ON to OFF response	1 ms
AC frequency	N/A		Maximum input rate	500 Hz
ON voltage level	0–1.5VDC @ 5VDC 0–4VDC @ 12–24VDC		Minimum ON current	0.4 mA @ 5VDC 0.9 mA @ 12–24VDC
OFF voltage level	3.5–5VDC @ 5VDC 10–24VDC @12–24VDC		Maximum OFF current	0.8 mA @ 5VDC 2.2 mA @ 12–24VDC
			Terminal type	Removable
			Status indicators	Logic side
			Weight	5.4 oz. (153 g)

* 12 Inputs are jumper selectable for 5VDC/12–24VDC and Sink Load/Source Load

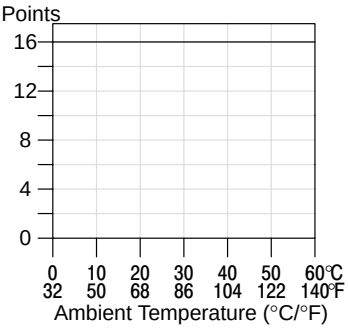
4 Inputs are jumper selectable for 5VDC/12–24VDC and Sink Load/Source Load

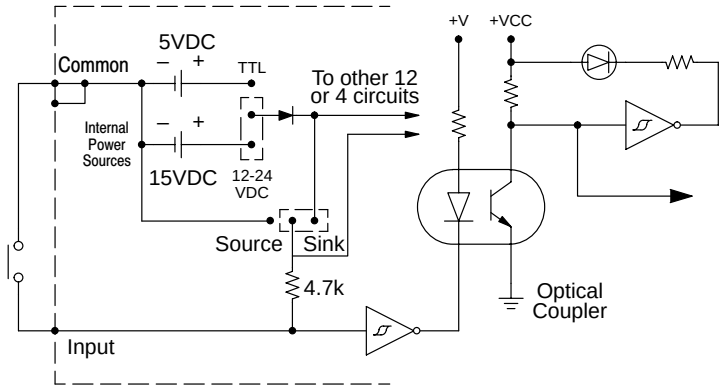


Sinking Load Configuration

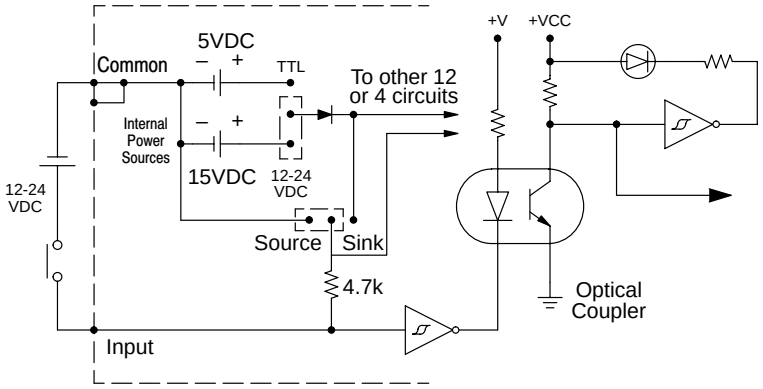


Derating Chart for F3–16ND3F





Jumper selected for 12–24VDC, sinking load configuration



Jumper selected for sourcing load configuration. An external power supply must be used in this configuration.

The DC power to sense the state of the inputs when jumpers are installed for sinking type signals is provided by the rack power supply. Sinking type inputs are turned ON by switching the input circuit to common. Source type input signals assume the ON state until the input device provides the voltage to turn the input OFF.

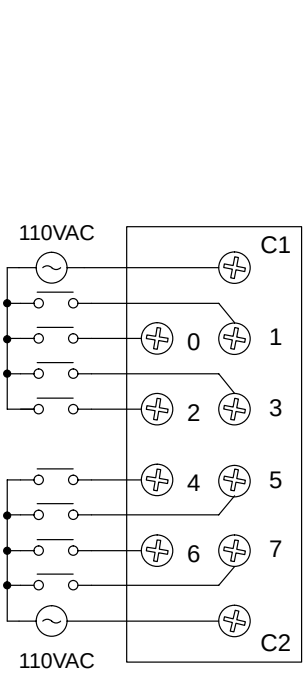
Selection of Operating Mode

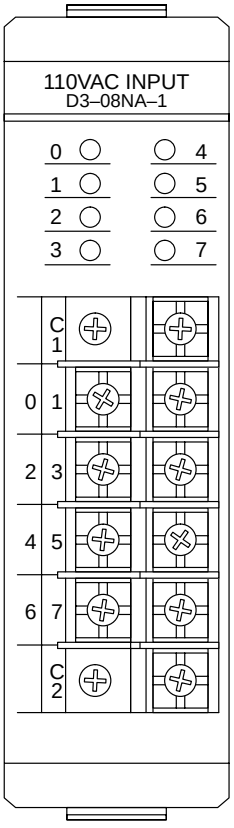
The mode of operation, either 5VDC or 12–24VDC sink or source, for each group of circuits is determined by the position of jumper plugs on pins located on the edge of the circuit board. There are four sets of pins (3 pins in each set), with two sets for each group of inputs. The first two sets of pins are used to configure the first 12 inputs (eg. 0 to 7 and 100 to 103) and are labeled 12 CIRCUITS. Above the first set of pins are the labels 12/24V and 5V. Above the second set of pins are the labels SINK and SRC (source). To select an operating mode for the first 12 circuits, place a jumper on the two pins nearest the appropriate labels. For example, to select 24VDC Sink input operation for the first 12 inputs, place a jumper on the two pins labeled 12/24V and on the two pins labeled SINK. The last two sets of pins are used to configure the last 4 inputs (eg. 104 to 107) and are labeled 4 CIRCUITS. The operating mode selected for the last group of 4 inputs can be different than the mode chosen for the first group of 12 inputs. Correct module operation requires each set of three pins have a jumper installed (four jumpers total).

NOTE: When a group of inputs are used with TTL logic, select the SINK operating mode for that group. “Standard” TTL can sink several milliamps but can source less than 1 mA.

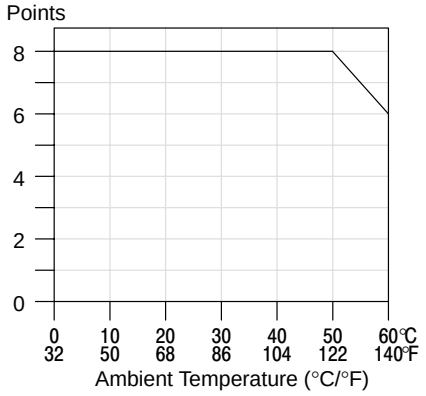
D3–08NA–1, 110 VAC Input Module

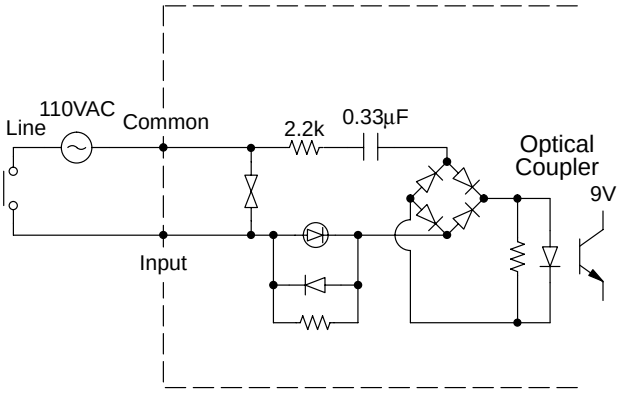
Inputs per module	8		Minimum ON current	8 mA
Commons per module	2 (isolated)		Maximum OFF current	2 mA
Input voltage range	85–132VAC		Base power required	9V 10 mA Max 24V N/A
Input voltage supply	External		OFF to ON response	10–30 ms
Peak voltage	132VAC		ON to OFF response	10–60 ms
AC frequency	47–63 Hz		Terminal type	Non–removable
ON voltage level	>80 VAC		Status indicators	Field side
OFF voltage level	<20 VAC		Weight	5 oz. (140 g)
Input impedance	10 K ohm			
Input current	15 mA @ 50 Hz 18 mA @ 60 Hz			





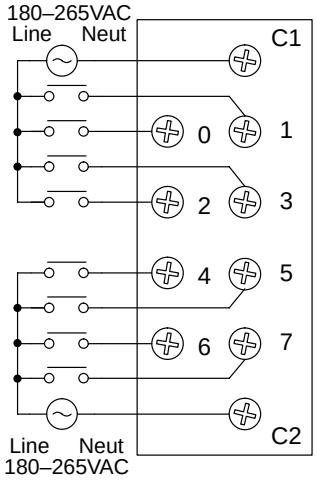
Derating Chart for D3–08NA–1

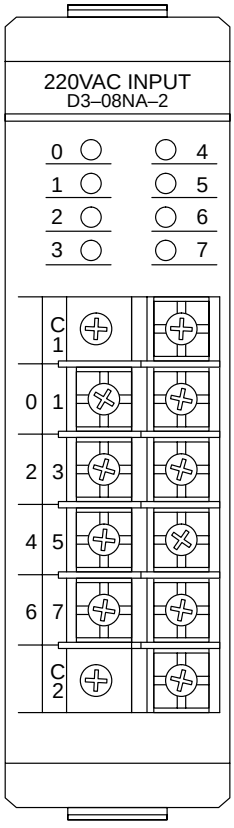


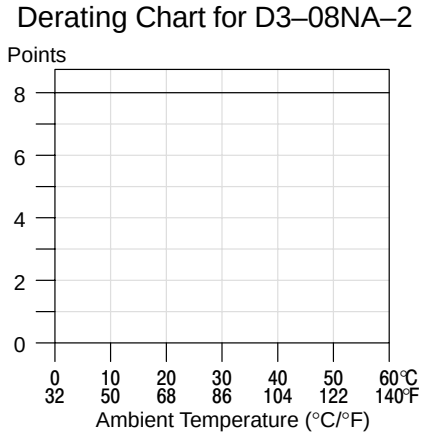


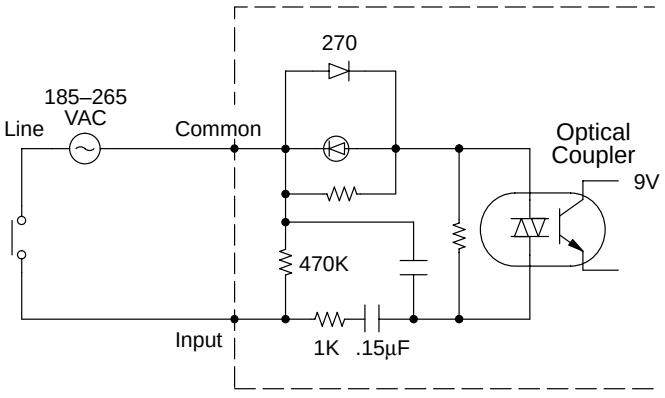
D3–08NA–2, 220 VAC Input Module

Inputs per module	8		Minimum ON current	10 mA
Commons per module	2 (isolated)		Maximum OFF current	2 mA
Input voltage range	180–265VAC		Base power required	9V 10 mA max 24V N/A
Input voltage supply	External		OFF to ON response	5–50 ms
Peak voltage	265 VAC		ON to OFF response	5–60 ms
AC frequency	50–60Hz		Terminal type	Non–removable
ON voltage level	>180 VAC		Status indicators	Field side
OFF voltage level	< 40 VAC		Weight	5 oz. (140 g)
Input impedance	18 K ohm			
Input current	13 mA @ 50 Hz 18 mA @ 60 Hz			





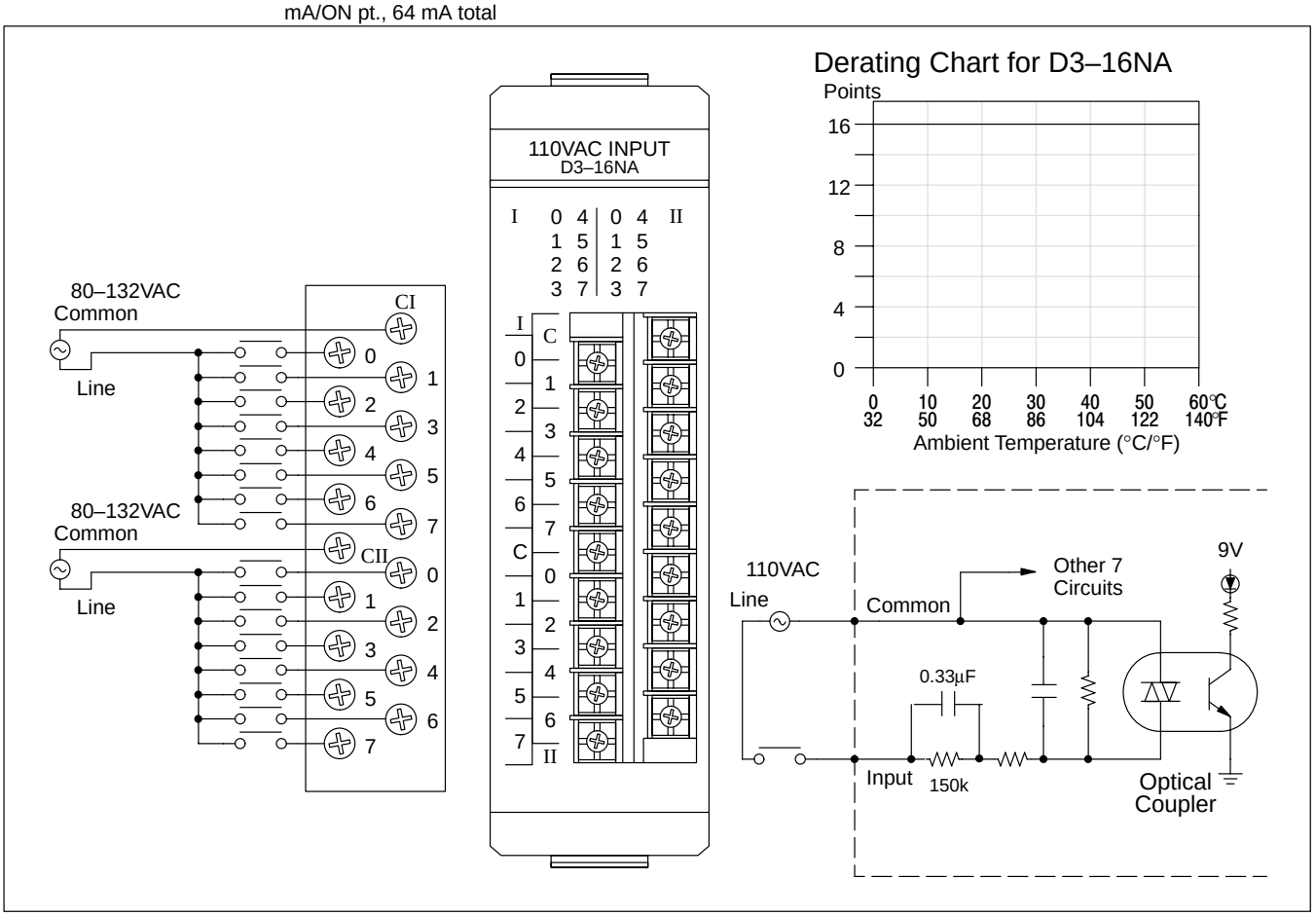




D3–16NA, 110 VAC Input Module

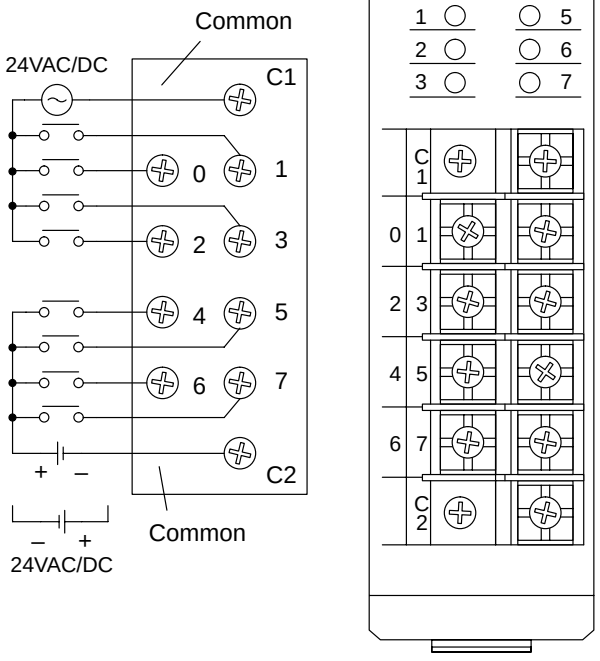
Inputs per module	16		Minimum ON current	8 mA
Commons per module	2 (isolated)		Maximum OFF current	1.5 mA
Input voltage range	80–132VAC		Base power required*	9V 6.25 mA Max/ON pt. 100mA max
Input voltage supply	External			
Peak voltage	132VAC		OFF to ON response	5–50 ms
AC frequency	50–60 Hz		ON to OFF response	5–60 ms
ON voltage level	>80 VAC		Terminal type	Removable
OFF voltage level	<15 VAC		Status indicators	Logic side
Input impedance	8 K ohm		Weight	6.4 oz. (180 g)
Input current	16 mA @ 50 Hz 25 mA @ 60 Hz			

* 9V typical values are 4 mA/ON pt., 64 mA total

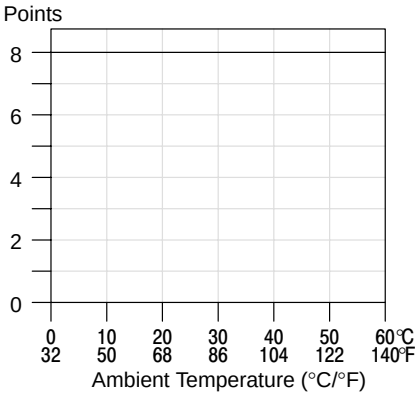


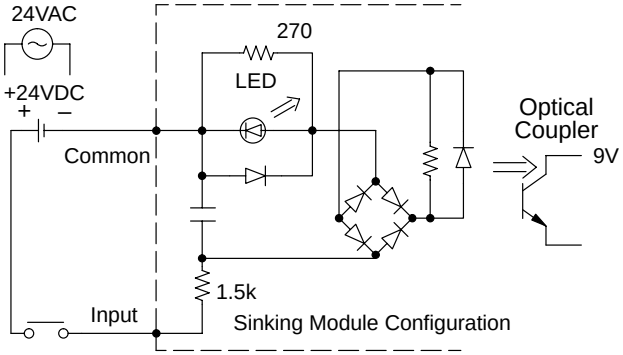
D3–08NE3, 24 VAC/DC Input Module

Inputs per module	8 (sink/source)		Base power required	9V 10 mA max 24V N/A
Commons per module	2 (isolated)			
Input voltage range	20–28 VAC/VDC		OFF to ON response	AC: 5–50 ms DC: 6–30 ms
Input voltage	External			
Peak voltage	28 VAC/VDC		ON to OFF response	AC/DC: 5–60 ms
AC frequency	47–63 Hz		Terminal type	Non–removable
ON voltage level	>20 V		Status indicators	Field side
OFF voltage level	<6V		Weight	4.2 oz. (120 g)
Input impedance	1.5 K ohm			
Input current	19 mA Max			
Minimum ON current	10 mA			
Maximum OFF current	2 mA			



Derating Chart for D3–08NE3

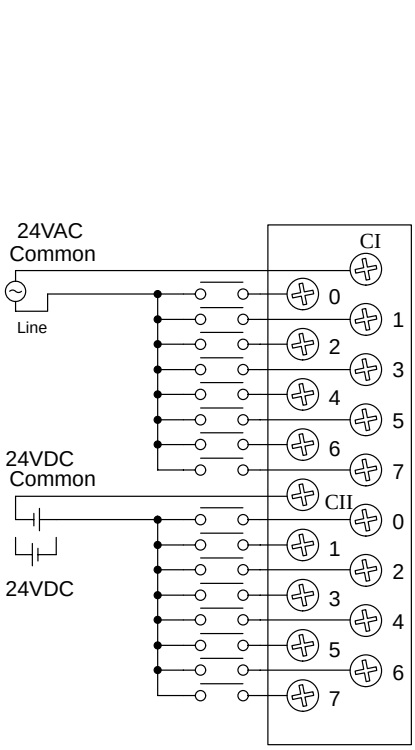


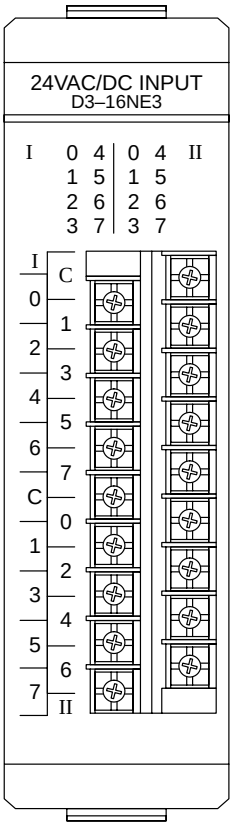


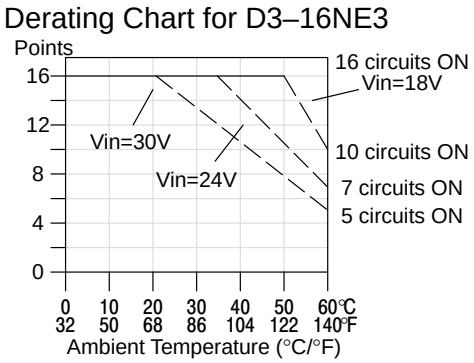
NOTE: This module can be wired in a sourcing configuration and it will be operational except there will be no module LED indication for each input.

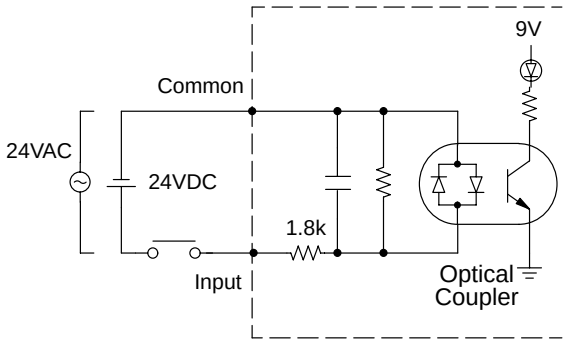
D3–16NE3, 24 VAC/DC Input Module

Inputs per module	16 (sink/source)		Base power required	9V 2.5 mA.+4.5mA/ ON pt.(130 mA max) 24V N/A
Commons per module	2 (isolated)			
Input voltage range	14–30VAC/VDC			
Input voltage supplied	External		OFF to ON response	AC 5–30 ms DC 5–25 ms
Peak voltage	30 VAC/VDC			
AC frequency	47–63 Hz		ON to OFF response	AC 5–30 ms DC 5–25 ms
ON voltage level	>14 V			
OFF voltage level	<3 V		Terminal type	Removable
Input impedance	1.8 K ohm		Status indicators	Logic side
Input current	16 mA Max		Weight	6 oz. (170 g)
Minimum ON current	7 mA			
Maximum OFF current	2 mA			





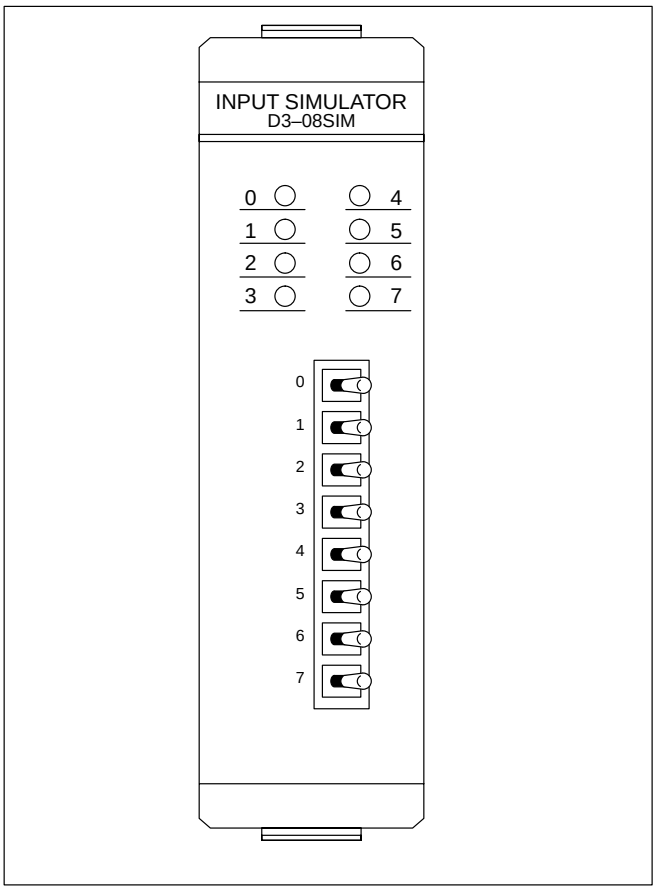




Sinking Module Configuration

D3–08SIM, Input Simulator

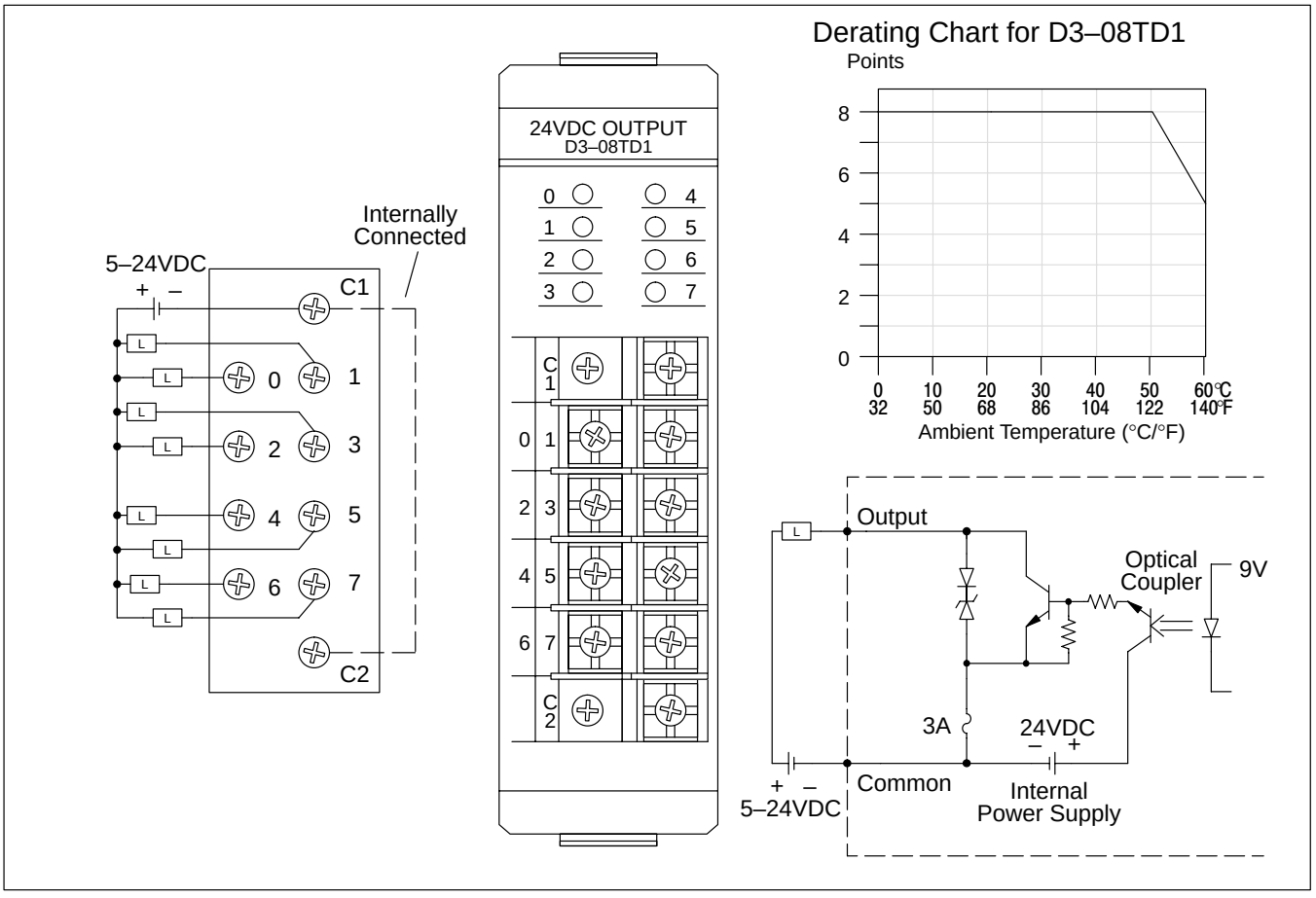
Inputs per module	8
Base Power required	10mA @ 9VDC 112mA max @ 24VDC
OFF to ON response	4–15 ms
ON to OFF response	4–15 ms
Terminal type	None
Status indicators	Switch side
Weight	3.0 oz. (85 g)



D3–08TD1, 24 VDC Output Module

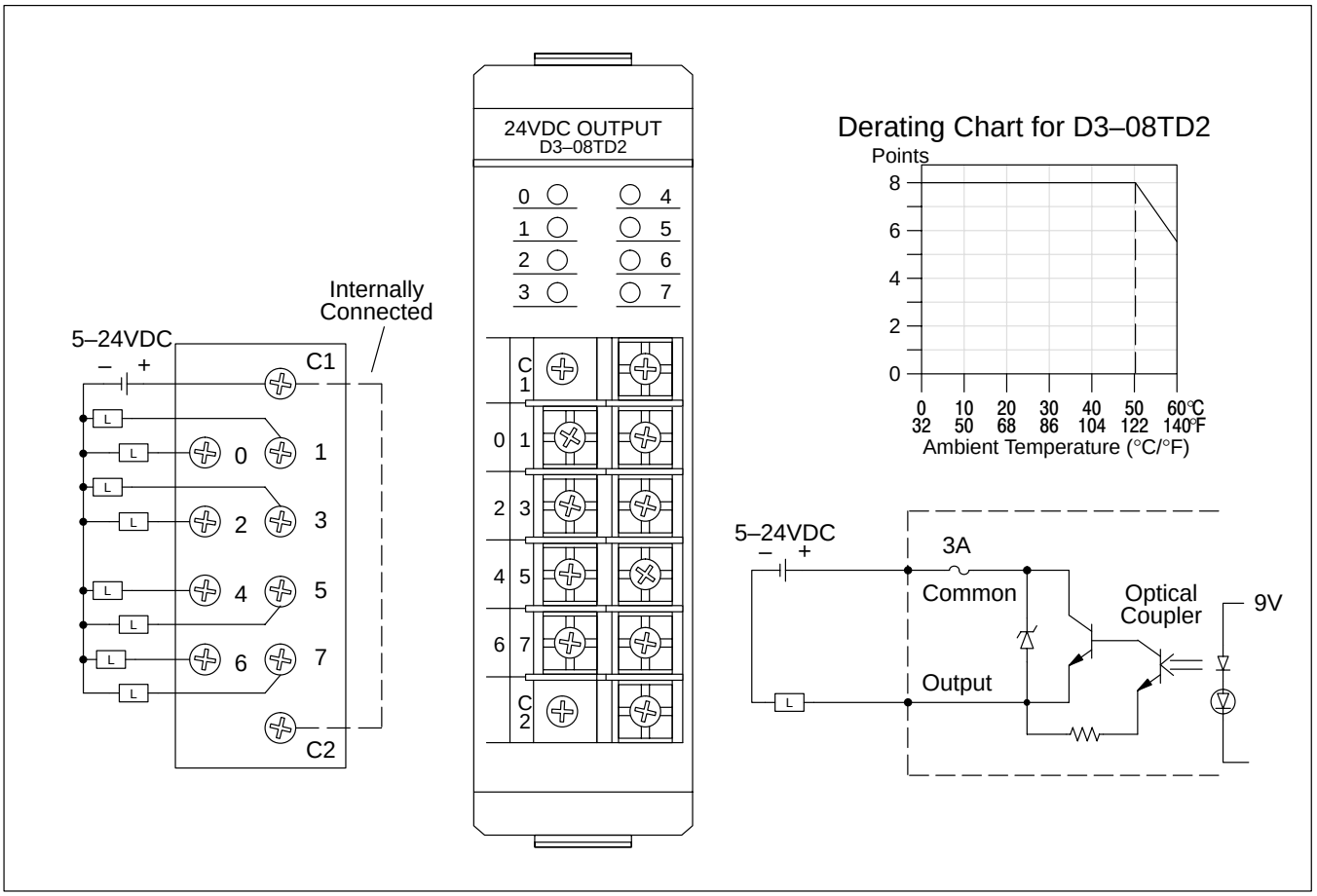
Outputs per module	8 (current sinking)
Commons per module	2(internally connected)
Operating voltage	5–24VDC
Output type	NPN (open collector)
Peak voltage	45VDC
AC frequency	N/A
ON voltage drop	0.8V @ 0.5A
Max current	0.5A / point 1.8 / common
Max leakage current	0.1 mA @ 40VDC
Max inrush current	3A / 20ms 1A / 100ms

Minimum load	1 mA
Base power required	9V 20 mA Max 24V 3mA/pt. (24mA Max)
OFF to ON response	0.1 ms
ON to OFF response	0.1 ms
Terminal type	Non-removable
Status indicators	Logic Side
Weight	4.2 oz. (120 g)
Fuses	(2) One 3A per common Non–replaceable



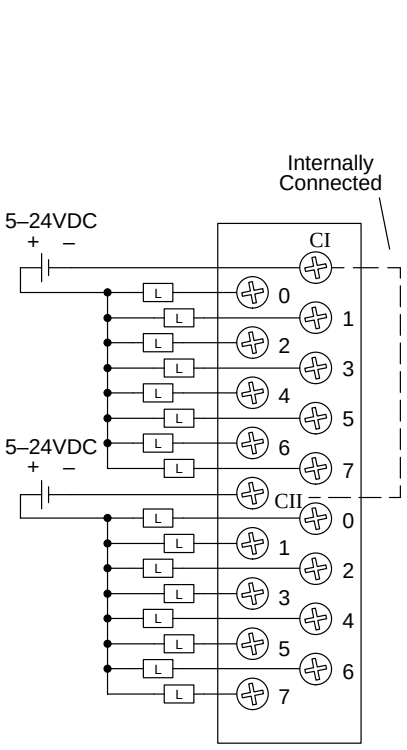
D3–08TD2, 24 VDC Output Module

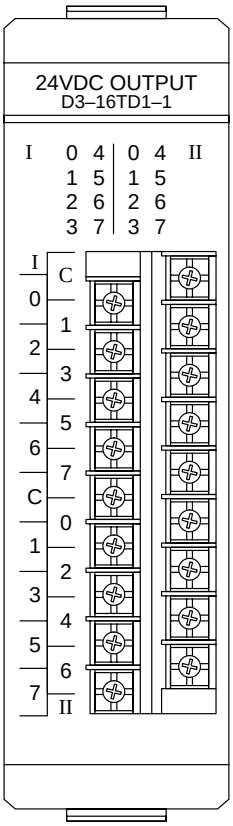
Outputs per module	8 (current sourcing)		Minimum load	1 mA
Commons per module	2 (internally connected)		Base power required	9V 30 mA Max 24V N/A
Operating voltage	5–24VDC			
Output type	NPN Transistor (emitter follower)		OFF to ON response	0.1 ms
			ON to OFF response	0.1 ms
Peak voltage	40VDC		Terminal type	Non-removable
AC frequency	N/A		Status indicators	Logic Side
ON voltage drop	1V @ 0.5A		Weight	4.2 oz. (120 g)
Max current	0.5A / point 1.8A/ common		Fuses	(2) One 3A per common Non–replaceable
Max leakage current	0.1 mA @ 24VDC			
Max inrush current	3A / 20ms 1A / 100ms			

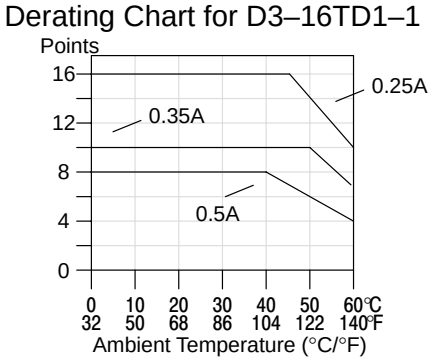


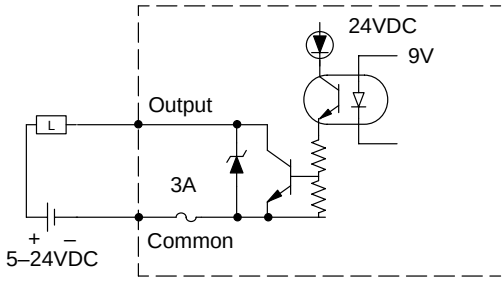
D3–16TD1–1, 24 VDC Output Module

Outputs per module	16 (current sinking)		Minimum load	1 mA
Commons per module	2 (internally connected)		Base power required	9V (40 mA Max) 3mA+2.3mA/ON pt. 24V 6 mA/ON pt. (96 mA Max)
Operating voltage	5–24VDC			
Output type	NPN transistor (open collector)			
Peak voltage	45VDC		OFF to ON response	0.1 ms
AC frequency	N/A		ON to OFF response	0.1 ms
ON voltage drop	2V @ 0.5A		Terminal type	Removable
Max current	0.5A/ point 2A/ common		Status indicators	Logic Side
			Weight	5.6 oz. (160 g)
Max leakage current	0.1mA @ 40VDC		Fuses	(2) One 3A per common Non-replaceable
Max inrush current	3A / 20 ms 1A / 100 ms			





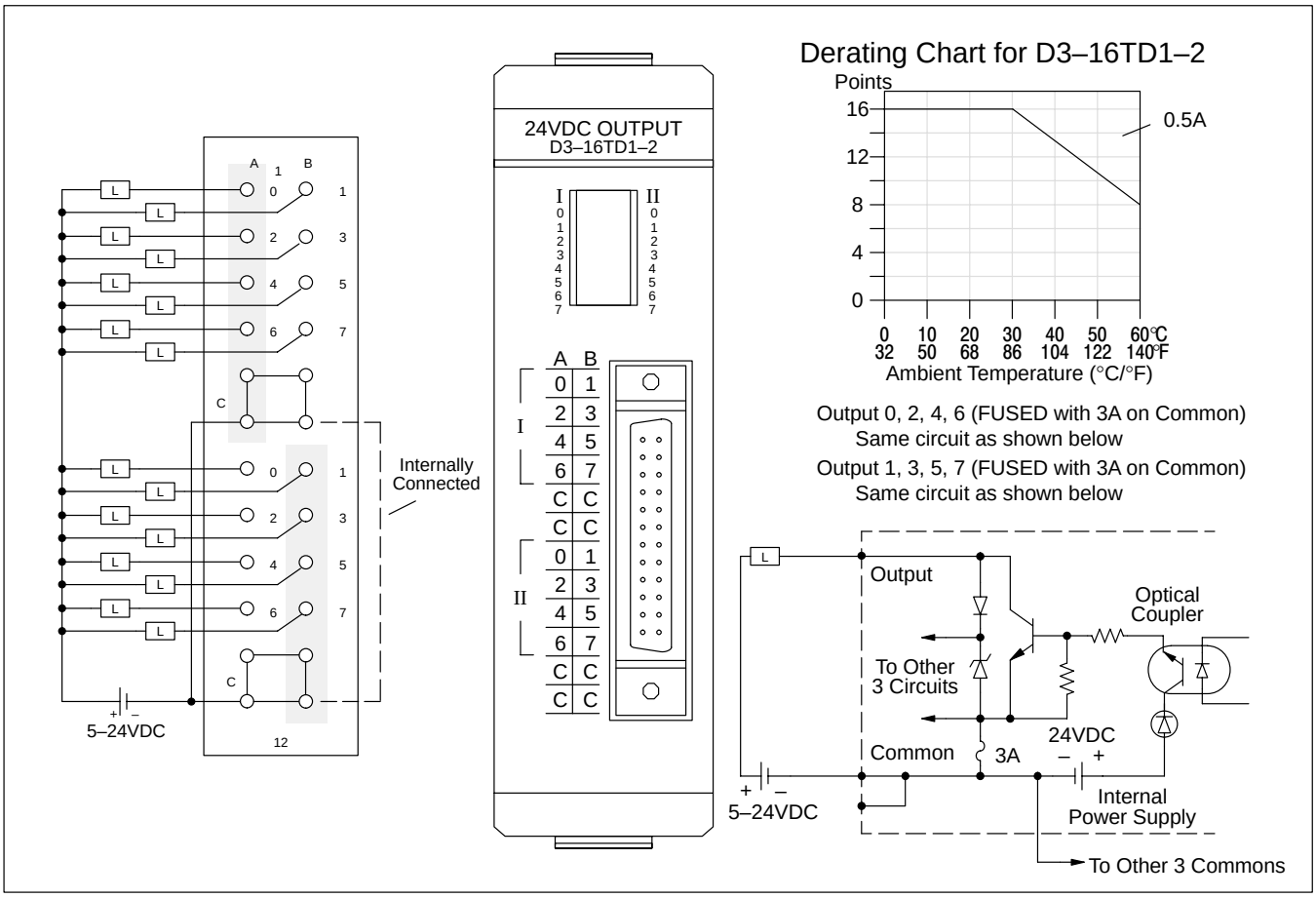




D3–16TD1–2, 24 VDC Output Module

Outputs per module	16 (current sinking)
Commons per module	4 (internally connected)
Operating voltage	5–24VDC
Output type	NPN transistor (open collector)
Peak voltage	45VDC
AC frequency	N/A
ON voltage drop	2.0V @ 0.5A
Max current	0.5A / point 1.8A common
Max leakage current	0.3 mA @ 40VDC
Max inrush current	3A / 20ms 1A / 100ms

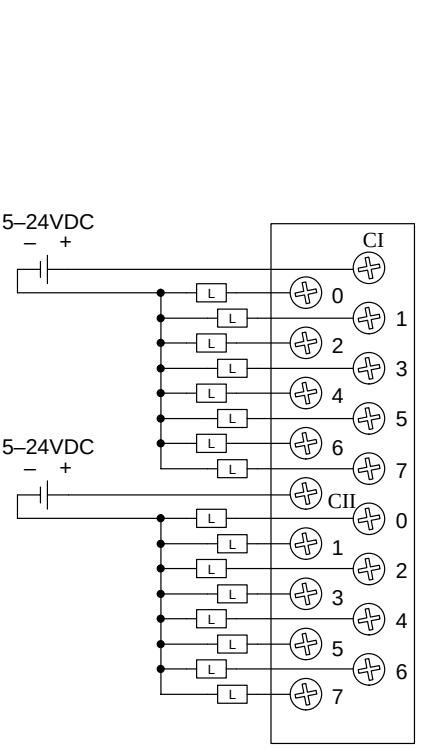
Minimum load	1 mA
Base power required	9V (40mA Max) 3mA+2.3mA/ON pt. 24V 6mA/ON pt. (96mA Max)
OFF to ON response	0.1 ms
ON to OFF response	0.1 ms
Terminal type	Removable connector
Status indicators	Logic Side
Weight	5.6 oz. (160 g)
Fuses	(4) One 3A per common Non–replaceable

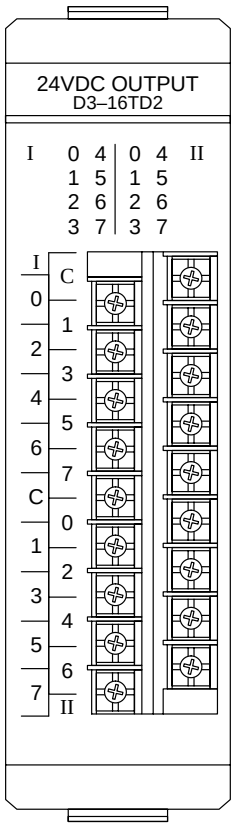


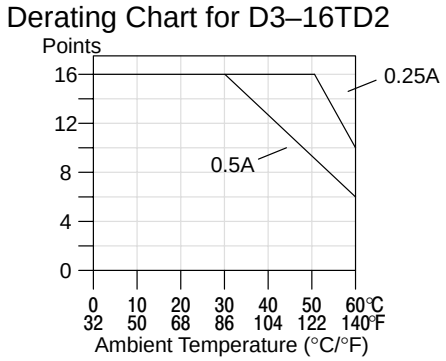
D3–16TD2, 24 VDC Output Module

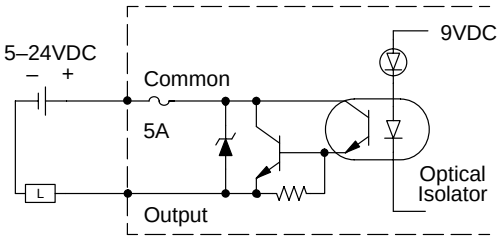
Outputs per module	16 (current sourcing)
Commons per module	2 (isolated)
Operating voltage	5–24VDC
Output type	NPN transistor (emitter follower)
Peak voltage	40VDC
AC frequency	N/A
ON voltage drop	1.5V @ 0.5A
Max current	0.5A / point 3A common
Max leakage current	0.01 mA @ 40VDC
Max inrush current	3A / 20ms 1A / 100ms

Minimum load	1 mA
Base power required	9V 7.5 mA/ON pt. (180 mA Max) 24V N/A
OFF to ON response	0.1 ms
ON to OFF response	1 ms
Terminal type	Removable
Status indicators	Logic Side
Weight	7.1 oz. (210 g)
Fuses	(2) One 5A per common Non–replaceable





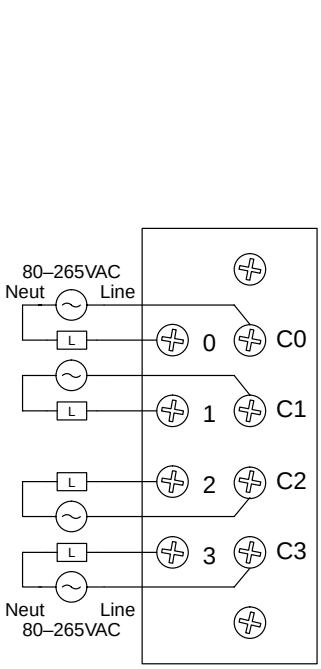


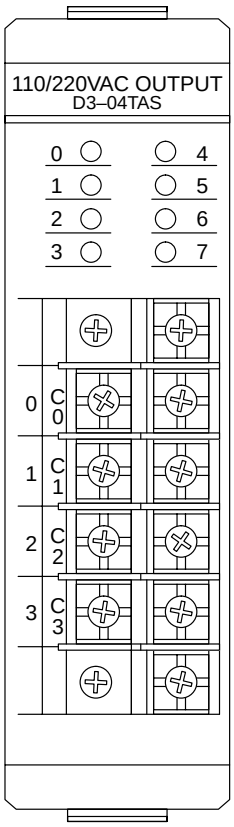


D3–04TAS, 110–220 VAC Output Module

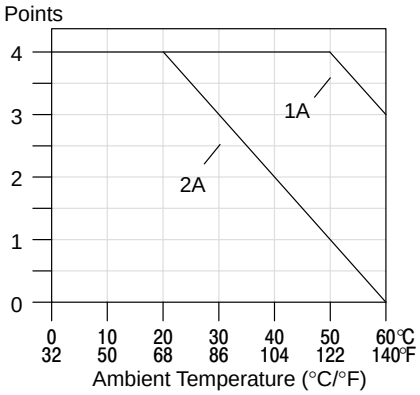
Outputs per module	4
Commons per module	4 (isolated)
Operating voltage	80–265VAC
Output type	Triac
Peak voltage	265 VAC
AC frequency	47–63 Hz
ON voltage drop	1.5 VAC @ 2A
Max current	2A / point 2A / common
Max leakage current	7 mA @ 220VAC 3.5 mA @ 110VAC
Max inrush current	20A for 16 ms 10A for 100 ms

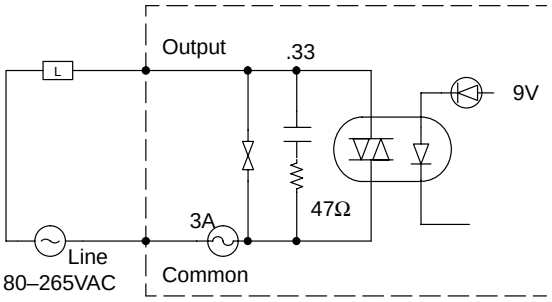
Minimum load	10 mA
Base power required	9V 12 mA Max 24V N/A
OFF to ON response	1 ms Max
ON to OFF response	10 ms Max
Terminal type	Non–removable
Status indicators	Logic Side
Weight	6.4 oz. (180 g)
Fuses	(4) One 3A per common User replaceable





Derating Chart for D3–04TAS



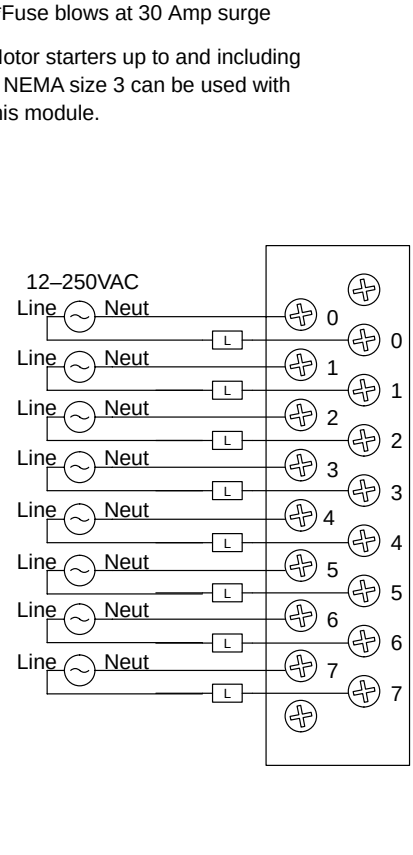


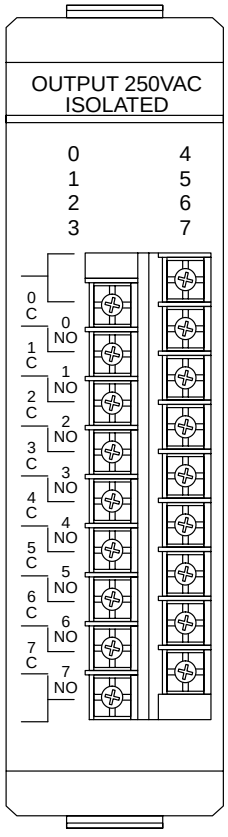
F3–08TAS, 250 VAC Isolated Output Module

Outputs per module	8 (500V point-to-point isolation)		Base power required	9V 10mA / ON pt. 80mA Max. 24V N/A
Commons per module	8 (isolated)		OFF to ON response	8 ms Max
Operating voltage	12–125 VAC 125–250 VAC requires external fuses		ON to OFF response	8 ms Max
Output type	SSR Array (TRIAC)		Terminal type	Removable
Peak voltage	400 VAC		Status indicators	Logic Side
AC frequency	47 – 440 Hz		Weight	N/A at press time
ON voltage drop	1 VAC @ 1A		Fuses BK/PCE–5 Bussman (One spare fuse included)	(8) fast blow One 5A (125V fast blow) per each circuit User replaceable
Max current	1A / point			
Max leakage current	10 μA @ 240 VAC			
Max inrush current*	20A for 16 ms 3A for 100 ms			
Minimum load	0.5 mA			

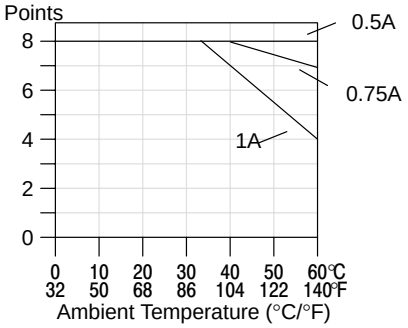
*Fuse blows at 30 Amp surge

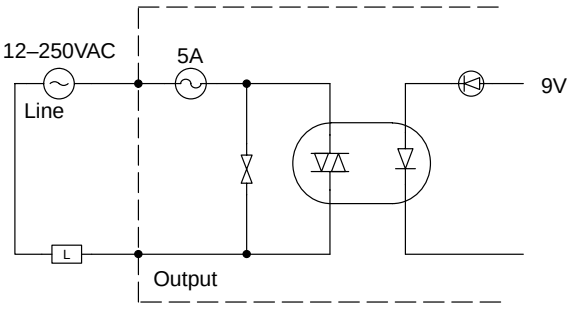
Motor starters up to and including a NEMA size 3 can be used with this module.





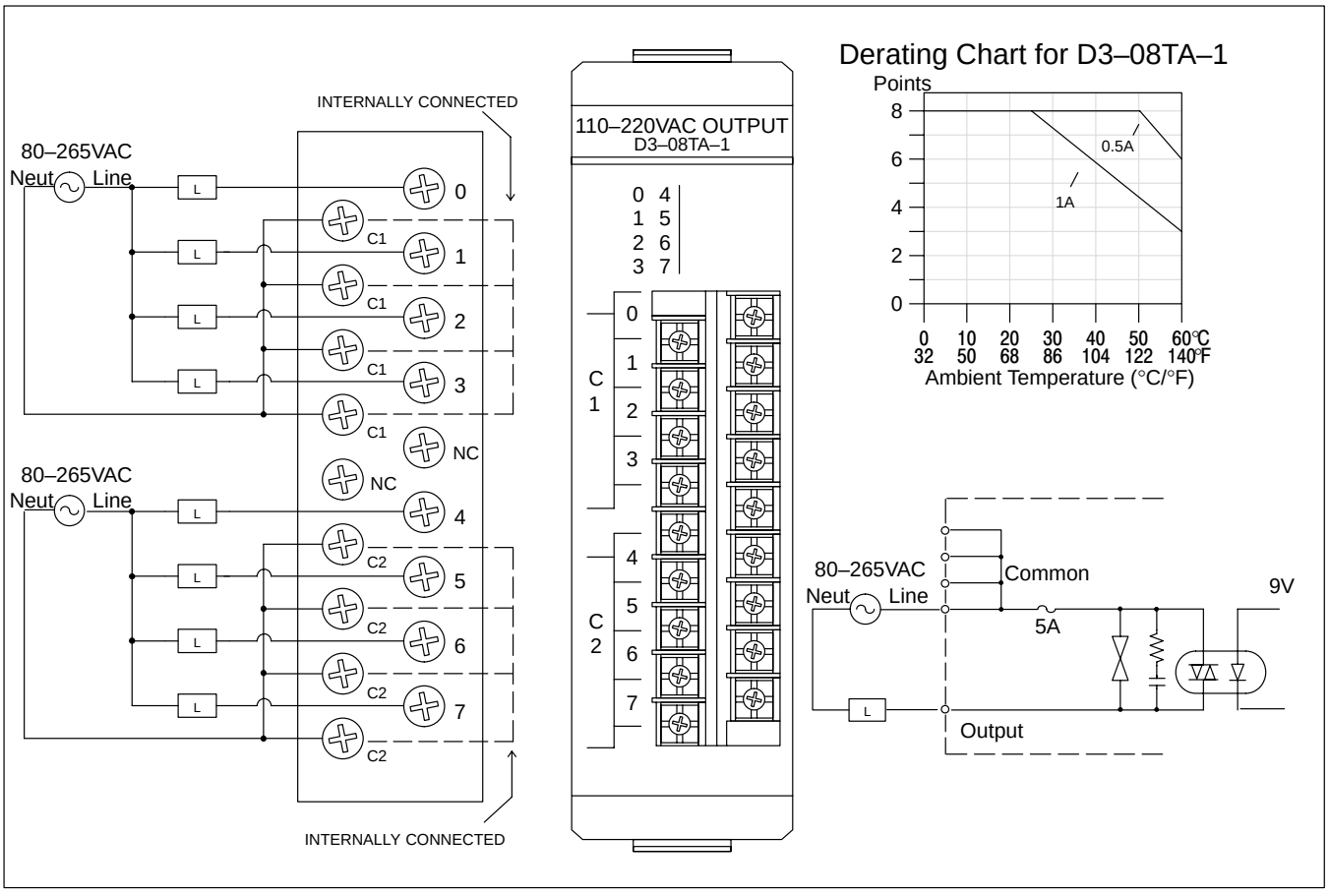
Derating Chart for F3–08TAS





D3–08TA–1, 110–220 VAC Output Module

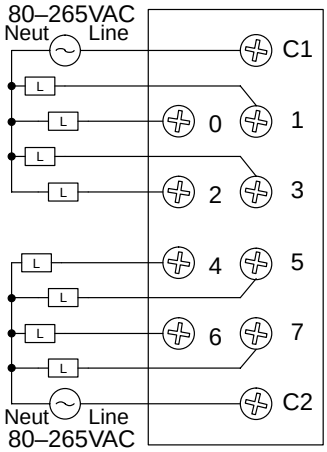
Outputs per module	8		Minimum load	25 mA
Commons per module	2 (isolated)		Base power required	9V 20mA/ON pt. (160 mA Max) 24V N/A
Operating voltage	80–265VAC			
Output type	Triac			
Peak voltage	265VAC		OFF to ON response	1 ms Max
AC frequency	47–63 Hz		ON to OFF response	8.33 ms Max
ON voltage drop	1.5 VAC @ 1A		Terminal type	Removable
Max current	1A / point 3A / common		Status indicators	Logic Side
			Weight	7.4 oz. (210 g)
Max leakage current	1.2 mA @ 220VAC 0.52 mA @ 110VAC		Fuses	(2) One 5A per common Non-replaceable
Max inrush current	10A for 16 ms 5A for 100 ms			

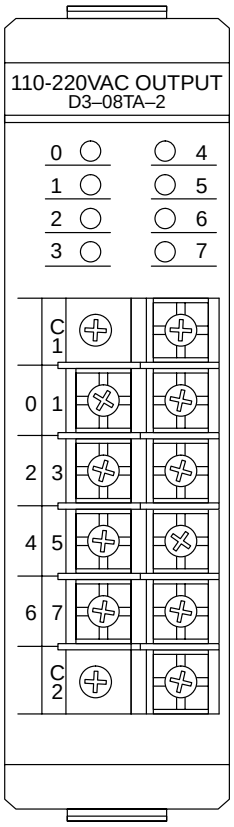


D3–08TA–2, 110–220 VAC Output Module

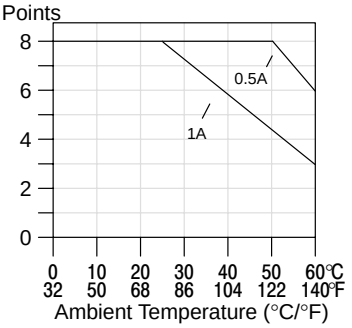
Outputs per module	8
Commons per module	2 (isolated)
Operating voltage	80–265VAC
Output type	Triac
Peak voltage	265VAC
AC frequency	47–63 Hz
ON voltage drop	1.5 VAC @ 1A
Max current	1A / point 3A / common
Max leakage current	1.2 mA @ 220VAC 0.52 mA @ 110VAC
Max inrush current	10A for 16 ms 5A for 100 ms
Minimum load	25 mA

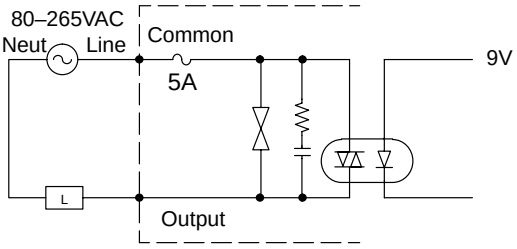
Base power required	9V 20mA/ON pt. (160 mA Max) 24V N/A
OFF to ON response	1 ms Max
ON to OFF response	8.33 ms Max
Terminal type	Non-removable
Status indicators	Logic Side
Weight	6.4 oz. (180 g)
Fuses	(2) One 5A per common Non–replaceable





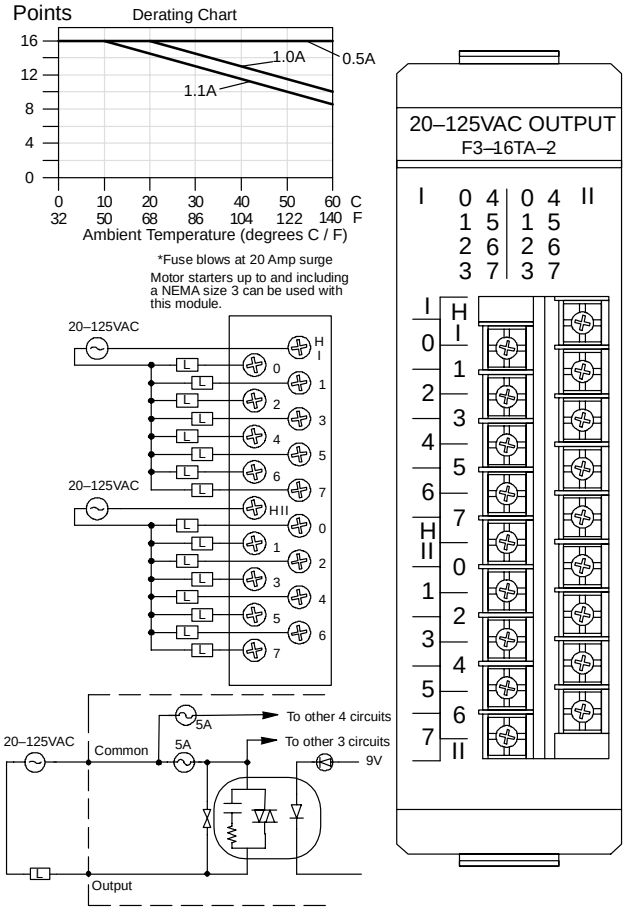
Derating Chart for D3–08TA–2





F3–16TA–2, 20–125 VAC Output Module

Outputs per module	16		Minimum load	50mA
Commons per module	2 (isolated)		Base power required	9V 14mA / ON pt. 250mA Max. 24V N/A
Operating voltage	20–125VAC		OFF to ON response	8ms Max
			ON to OFF response	8ms Max
Output type	SSR Array (TRIAC)		Terminal type	Removable
Peak voltage	140VAC		Status indicators	Logic Side
AC frequency	47 – 63Hz		Weight	7.7oz. (218g)
ON voltage drop	1.1VAC @ 1.1A		Fuses	4 (One 5A 125V fast blow per each group of four outputs) Order D3–FUSE–4 (5 per pack)
Max current	1.1A / point			
Max leakage current	0.7mA @ 125VAC			
Max inrush current*	15A for 20 ms 8A for 100 ms			

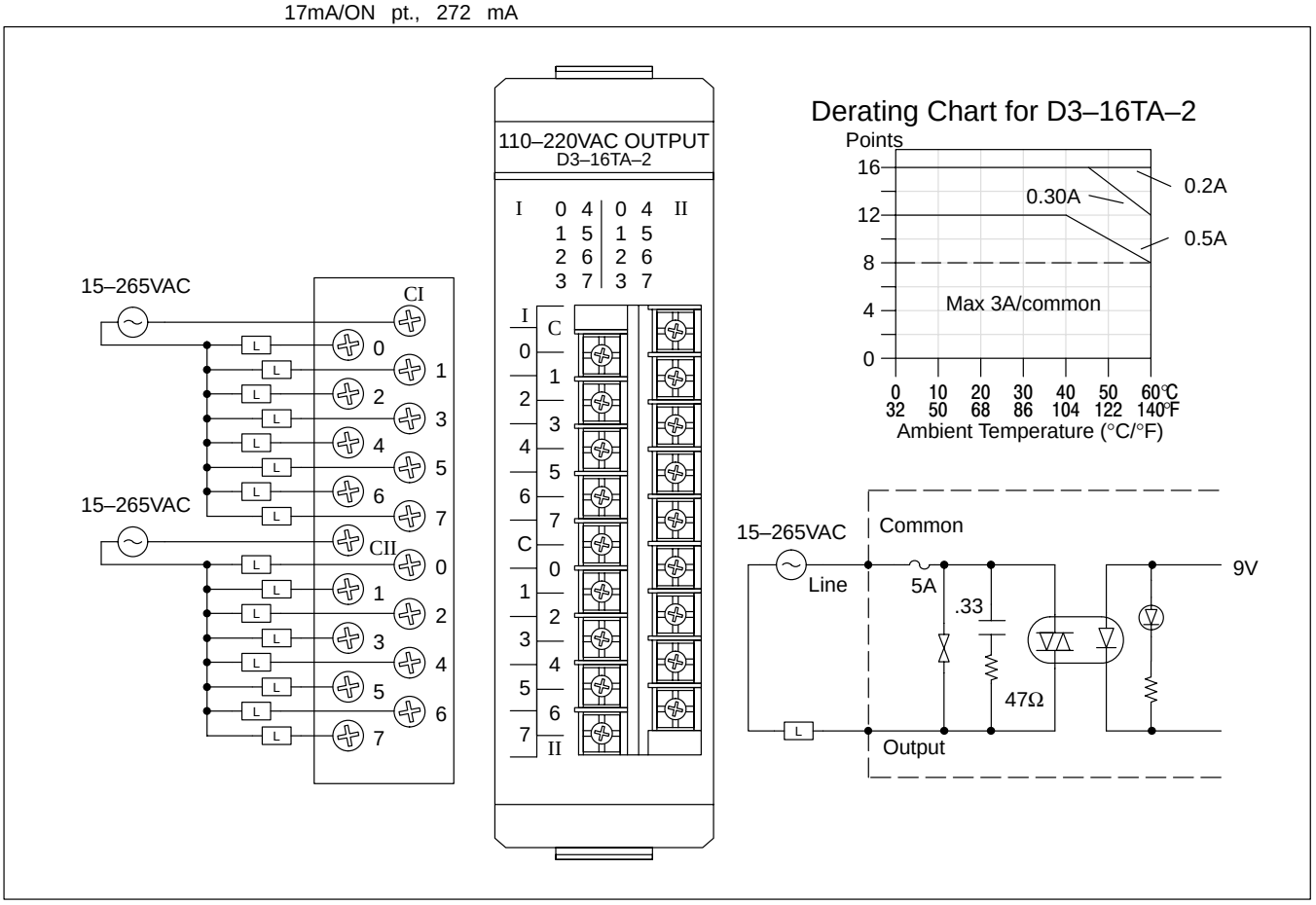


D3–16TA–2, 15–220 VAC Output Module

Outputs per module	16		Minimum load	10 mA @ 15VAC
Commons per module	2 (isolated)		Base power required *	9V 25mA Max /ON pt. 400 mA Max 24V N/A
Operating voltage	15–265 VAC			
Output type	Triac		OFF to ON response	1 ms Max
Peak voltage	265 VAC		ON to OFF response	9 ms Max
AC frequency	47–63 Hz		Terminal type	Removable
ON voltage drop	1.5 VAC @ 0.5A		Status indicators	Logic Side
Max current	0.5A / point 3A / common 6A / per module		Weight	7.2 0z. (210 g)
			Fuses	(2) One 5A per common Non–replaceable
Max leakage current	4 mA @ 265 VAC			
Max inrush current	10A for 10 ms 5A for 100 ms			

total

* 9V typical values
17mA/ON pt., 272 mA

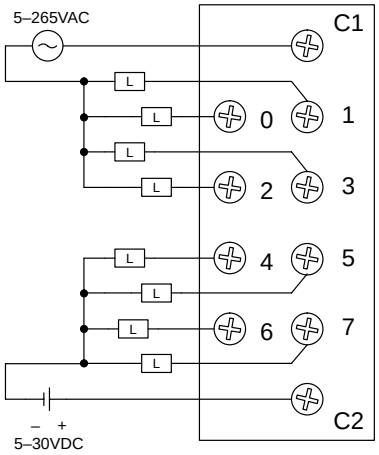


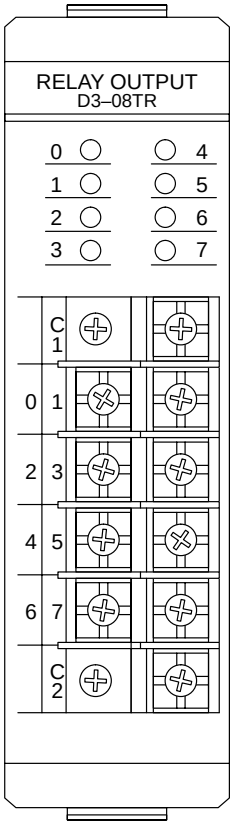
D3–08TR, Relay Output Module

Outputs per module	8		Minimum load	5 mA @ 5v
Commons per module	2 (isolated)		Base power required	9V 45 mA/ON pt. (360 mA Max) 24V N/A
Operating voltage	5–265VAC 5–30VDC			
Output type	Form A (SPST)		OFF to ON response	5 ms
Peak voltage	265VAC / 30VDC		ON to OFF response	5 ms
AC frequency	47–63 Hz		Terminal type	Non-removable
ON voltage drop	N/A		Status indicators	Logic Side
Max current	4A / point AC 5A / point DC 6A / common		Weight	7 oz. (200 g)
			Fuses	(2) One 10A per common User replaceable
Max leakage current	1 mA @ 220VAC			
Max inrush current	5A			

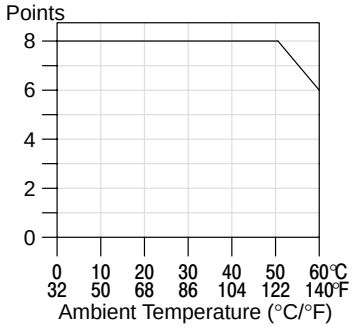
Typical Relay Life (Operations)

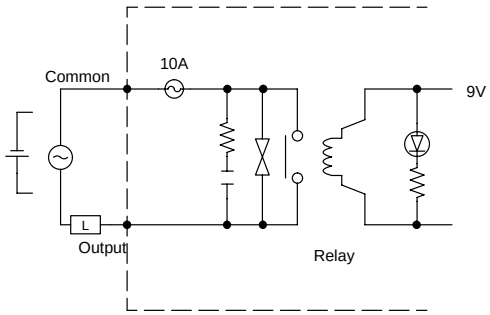
Voltage	Resistive	Solenoid	Closures
220VAC	4A	0.5A	100k
220VAC		0.05A	800k
110VAC	4A	0.5A	100k
110VAC		0.1A	650k
24VDC	5A	0.5A	100k





Derating Chart for D3–08TR



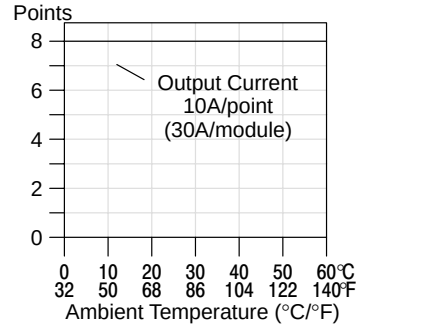


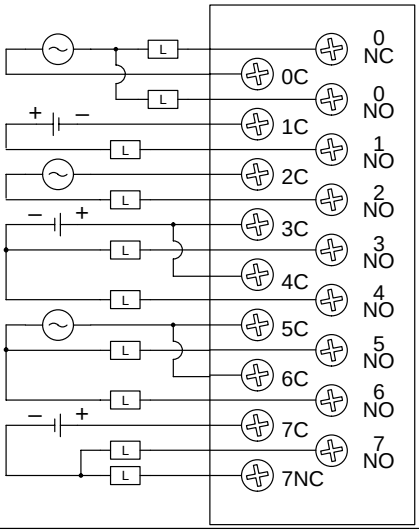
F3–08TRS–1, Relay Output Module

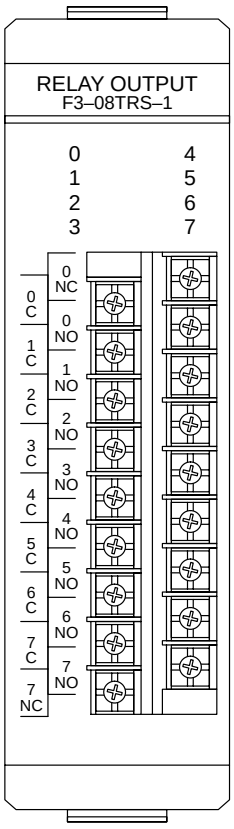
Outputs per module	8		Max leakage current	N/A
Commons per module	8 (isolated)		Max inrush current	10A Inductive
Operating voltage*	12–125 VAC 125–250 VAC requires external fuses 12–30 VDC		Minimum load	100 mA @12VDC
			Base power required	9V 37mA / ON pt. (296 mA Max) 24V N/A
Output type	6 Form A (SPST) 2 Form C (SPDT)		OFF to ON response	13 ms Max
			ON to OFF response	9 ms Max
Peak voltage	265 VAC / 120 VDC		Terminal type	Removable
AC frequency	47–63 Hz		Status indicators	Logic Side
ON voltage drop	N/A		Weight	8.9 oz. (252 g)
Max current (resistive)	10A / point AC/DC 30A / module AC/DC		Fuses	(8) One 10A (125V) per common Non-replaceable

NOTE: Contact life may be lengthened beyond those values shown by the use of an appropriate arc suppression. This technique is discussed earlier in this chapter.

Derating Chart for F3–08TRS–1

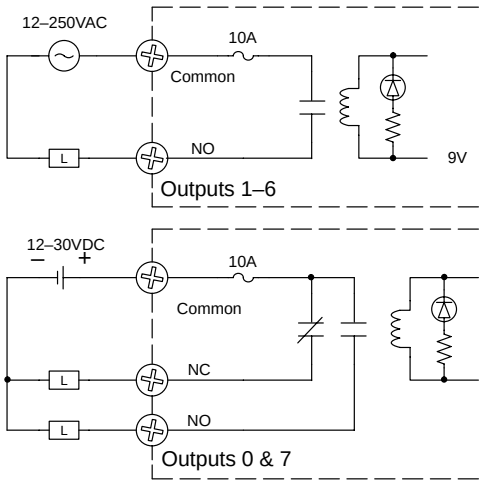






Typical Relay Life (Operations)

Maximum Resistive or Inductive Inrush Load Current	Operating Voltage		
	28VDC	120VAC	240VAC
1/4HP		25K	
10.0A	50K	50K	
5.0A	200K	100K	
3.0A	325K	125K	50K
.05A	>50M		



*Maximum DC voltage rating is 120 VDC at .5 Amp, 30,000 cycles typical

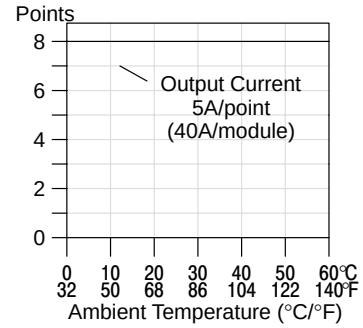
Motor starters up to and including a NEMA size 4 can be used with this module.

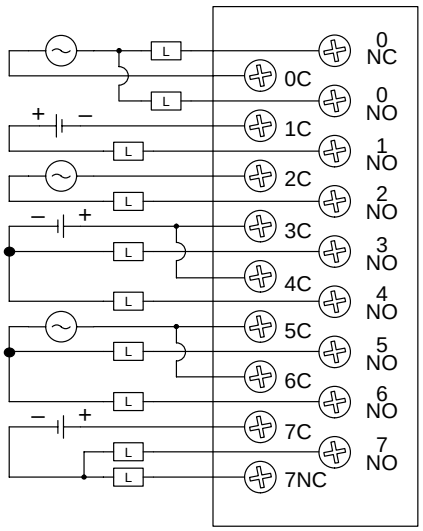
F3–08TRS–2, Relay Output Module

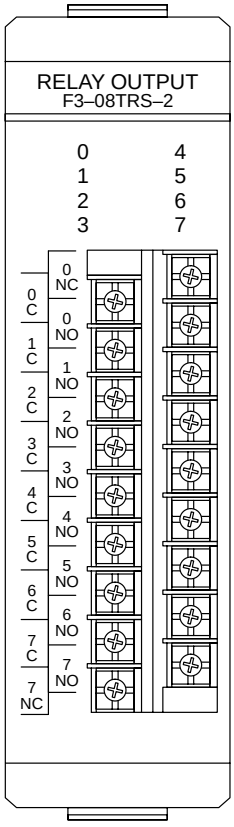
Outputs per module	8		Max leakage current	N/A
Commons per module	8 (isolated)		Max inrush current	10A Inductive
Operating voltage*	12–125 VAC 12–30 VDC		Minimum load	100 mA @12VDC
			Base power required	9V 37mA / ON pt. (296 mA Max) 24V N/A
Output type	6 Form A (SPST) 2 Form C (SPDT)		OFF to ON response	13 ms Max
			ON to OFF response	9 ms Max
Peak voltage	265 VAC / 120 VDC		Terminal type	Removable
AC frequency	47–63 Hz		Status indicators	Logic Side
ON voltage drop	N/A		Weight	9 oz. (255 g)
Max current (resistive)	5A / point AC/DC 40A / module AC/DC		Fuses 19379–K–10A Wickman	(8) One 5A (125V) per common User replaceable

NOTE: Contact life may be lengthened beyond those values shown by the use of an appropriate arc suppression. This technique is discussed earlier in this chapter.

Derating Chart for F3–08TRS–2



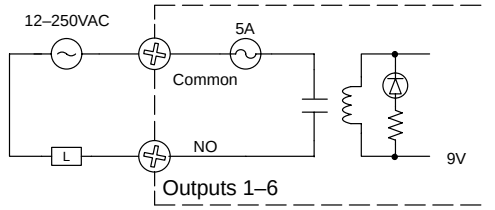


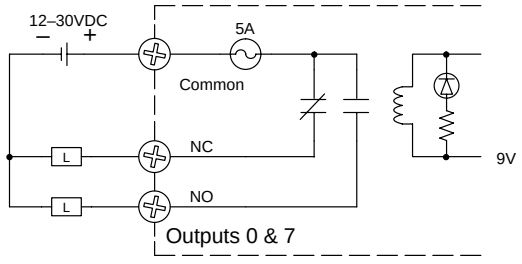


Typical Relay Life (Operations)

Maximum Resistive or Inductive Inrush Load Current	Operating Voltage		
	28VDC	120VAC	240VAC
5.0A	200K	100K	
3.0A	325K	125K	50K
.05A	>50M		

Expected mechanical relay life is 100 million operations.





*Maximum DC voltage rating is 120 VDC at .5 Amp, 30,000 cycles typical

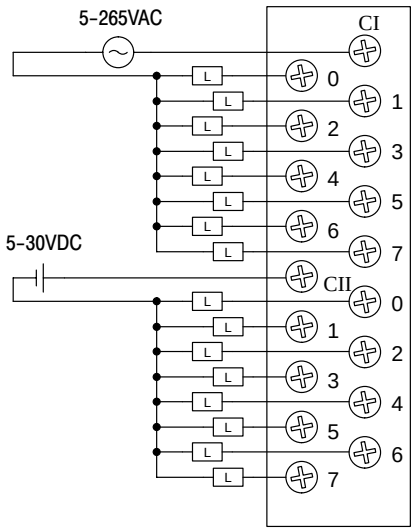
Motor starters up to and including a NEMA size 3 can be used with this module.

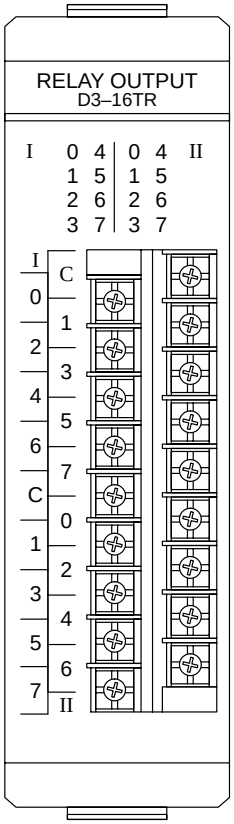
D3–16TR, Relay Output Module

Outputs per module	16		Minimum load	5 mA @ 5v
Commons per module	2 (isolated)		Base power required	9V 30 mA/ON pt. (480 mA Max) 24V N/A
Operating voltage	5–265 VAC 5–30 VDC		OFF to ON response	12 ms
Output type	16 Form A (SPST)		ON to OFF response	12 ms
Peak voltage	265 VAC / 30 VDC		Terminal type	Removable
AC frequency	47–63 Hz		Status indicators	Logic Side
ON voltage drop	N/A		Weight	8.5 oz. (248g)
Max current	2A / point AC/DC (resistive) 8A / common AC/DC		Fuses	None
Max leakage current	0.1mA @ 220 VAC			
Max inrush current	2A			

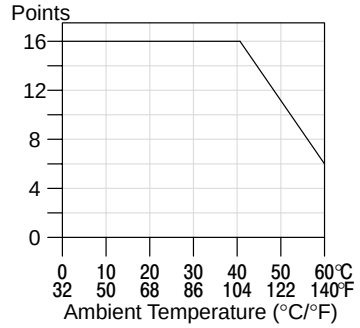
Typical Relay Life (Operations)

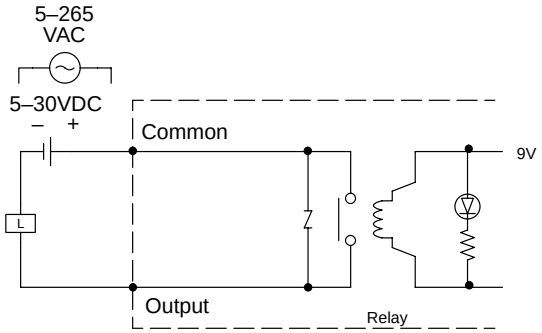
Voltage	Resistive	Solenoid	Closures
220VAC	2A	0.25A	100k
220VAC		0.03A	800k
110VAC	2A	0.25A	100k
110VAC		0.05A	650k
24VDC	2A	0.25A	100k





Derating Chart for D3–16TR





3

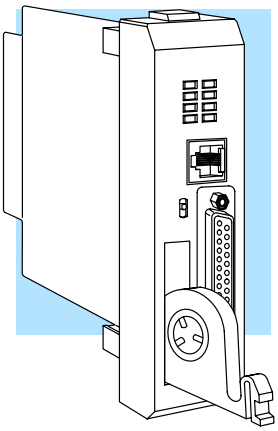
CPU Specifications and Operations

In This Chapter. . . .

Overview

The CPU is the heart of the control system. Almost all system operations are controlled by the CPU, so it is important that it is set-up and installed correctly. This chapter provides the information needed to understand:

- the differences between the different models of CPUs
- the steps required to setup and install the CPU



General CPU Features

The DL350 is a modular CPU which can be installed in 5, 8, or 10 slot bases. All I/O modules in the DL305 family will work with the CPU. The DL350 CPU offers a wide range of processing power and program RLL and Stage program instructions (see Chapter 5). It also provides extensive internal diagnostics that can be monitored from the application program or from an operator interface. The DL350 is different than the other CPUs in the DL305 family. It supports a 16 bit addressing format where the DL330/340 are 8 bit. This has enabled the DL350 to expanded its instruction set, memory, and features much like the DL205 and DL405 CPUs.

DL350 CPU Features

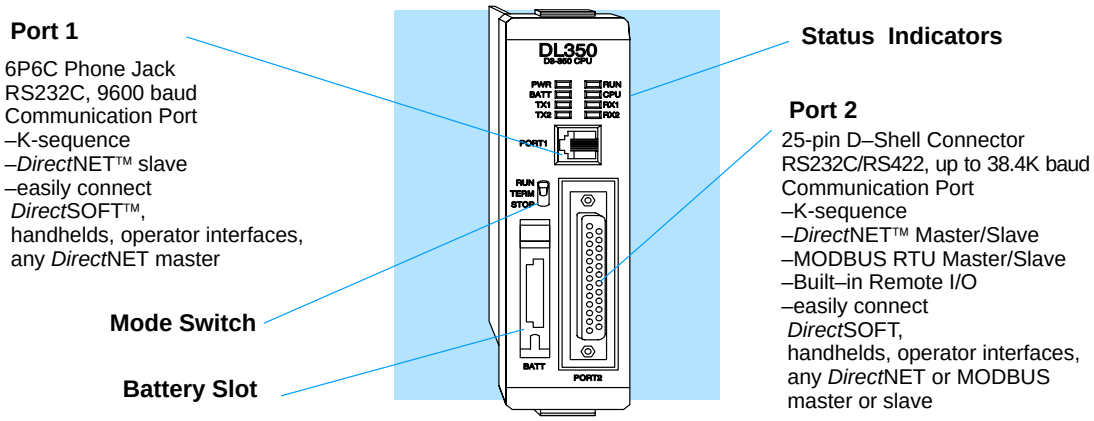
The DL350 has a maximum of 14.8K of program memory comprised of 7.6K of ladder memory and 7.2K of V-memory (data registers). It supports a maximum of 368 points of local I/O, and 880 points with remote I/O. It includes an additional internal RISC–based microprocessor for greater processing power. The DL350 has over 150 instructions, including drum timers, a print function, floating point math, and PID loop control for 4 loops.

The DL350 has a total of two communications ports. The top port is a 6 pin modular that provides a built–in RS232 communication port. It can be used for easy connection of the handheld programmer, PC, or used for a ***Direct***NET slave. The bottom port is a 25–pin RS232C/RS422 port. It will interface with ***Direct***SOFT, and operator interfaces, provides built–in Remote I/O, ***Direct***NET and MODBUS RTU Master/Slave connections.

CPU General Specifications

Feature	DL350
Total Program memory (words)	14.8K
Ladder memory (words)	7680 (Flash)
V-memory (words)	7168
Non-volatile V Memory (words)	No
Boolean execution /K	5–6 ms
RLL and RLL*PLUS* Programming	Yes
Handheld programmer	Yes
DirectSOFT™ programming for Windows™	Yes
Built-in communication ports (RS232C)	Yes
CMOS RAM	No
UVPROM	No
EEPROM	Flash
Local Discrete I/O points available	368
Remote I/O points available	512
Remote I/O Channels	1
Max Number of Remote Slaves	7
Local Analog input / output channels maximum	128 / 32
Counter Interface Module (quad., pulse out, pulse catch, etc.)	No
I/O Module Point Density	8/16
Slots per Base	5/8/10
Number of instructions available (see Chapter 5 for details)	170
Control relays	1024
Special relays (system defined)	144
Stages in RLL*PLUS*	1024
Timers	256
Counters	128
Immediate I/O	Yes
Interrupt input (hardware / timed)	No / Yes
Subroutines	Yes
Drum Timers	Yes
For/Next Loops	Yes
Math	Integer,Floating Point
PID Loop Control, Built In	Yes
Time of Day Clock/Calendar	Yes
Run Time Edits	Yes
Internal diagnostics	Yes
Password security	Yes
System error log	Yes
User error log	Yes
Battery backup	Yes (optional)

CPU Hardware Features



Mode Switch Functions

The mode switch on the DL350 CPUs provide 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 programing package or operator interface). 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 programming or monitoring device.

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 (Stop Program)	CPU is forced into the STOP mode. No change or monitoring is allowed by the programming/monitoring device.

There are two ways to change the CPU mode.

1. Use the CPU mode switch to select the operating mode.
2. Place the CPU mode switch in the TERM position and use a programming device to change operating modes. In this position, you can change between Run and Program modes.

Status Indicators

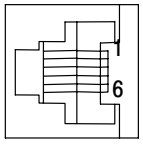
The status indicator LEDs on the CPU front panels have specific functions which can help in programming and troubleshooting.

Indicator	Status	Meaning
PWR	ON	Power good
RUN	ON	CPU is in Run Mode
CPU	ON	CPU self diagnostics error
BATT	ON	CPU battery voltage is low
TX1	ON	Transmitting Data from Port 1
RX1	ON	Receiving Data at port 1
TX2	ON	Transmitting Data from Port 2
RX2	ON	Receiving Data at Port 2

Port 1 Specifications

The operating parameters for Port 1 on the DL350 CPU are fixed.

- 6 Pin female modular (RJ12 phone jack) type connector
- ***Direct***Net (slave), K–sequence protocol
- RS232C, 9600 baud
- Connect to ***Direct***SOFT, D2–HPP, DV1000 or ***Direct***NET master



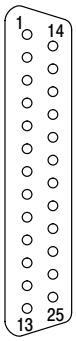
6-pin Female
Modular Connector

Port 1 Pin Descriptions (DL350 only)		
1	0V	Power (–) connection (GND)
2	5V	Power (+) connection
3	RXD	Receive Data (RS232C)
4	TXD	Transmit Data (RS232C
5	5V	Power (+) connection
6	0V	Power (–) connection (GND)

Port 2 Specifications

Port 2 on the DL350 CPU is located on the 25 pin D-shell connector. It is configurable using AUX functions on a programming device.

- 25 Pin female D type connector
- Protocol: K sequence, ***Direct***NET Master/Slave, MODBUS RTU Master/Slave, Remote I/O, non–procedure
- RS232C, non-isolated, distance within 15 m (approx. 50 feet)
- RS422C, non-isolated, distance within 1000 m
- Up to 38.4K baud
- Address selectable (1–90)
- Connects to ***Direct***SOFT, operator interfaces, any ***Direct***NETor MODBUS master or slave



25-pin Female
D Connector

Port 2 Pin Descriptions (DL350 CPU)		
1	not used	
2	TXD	Transmit Data (RS232C)
3	RXD	Receive Data (RS232C)
4	RTS	Ready to Send (RS–232C)
5	CTS	Clear to Send (RS–232C)
6	not used	
7	0V	Power (–) connection (GND)
8	0V	Power (–) connection (GND)
9	RXD +	Receive Data + (RS–422)
10	RXD –	Receive Data (RS–422)
11	CTS +	Clear to Send + (RS422)
12	TXD +	Transmit Data + (REMIO)
13	TXD –	Transmit Data – (REMIO)

Port 2 Pin Descriptions (Cont'd)		
14	TXD +	Transmit Data + (RS–422
15	not used	
16	TXD –	Transmit Data – (RS–422)
17	not used	
18	RTS –	Request to Send – (RS–422)
19	RTS +	Request to Send – (RS–422)
20	not used	
21	not used	
22	not used	
23	CTS –	Clear to Send – (RS–422)
24	RXD +	Receive Data + (REMIO)
25	RXD –	Receive Data – (REMIO)

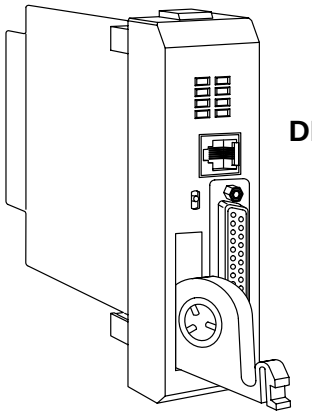
Using Battery Backup

An optional lithium battery is available to maintain the system RAM retentive memory when the DL305 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 shutdown 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 DL350 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.
3. Close the battery door making sure that it locks securely in place.
4. Make a note of the date the battery was installed.



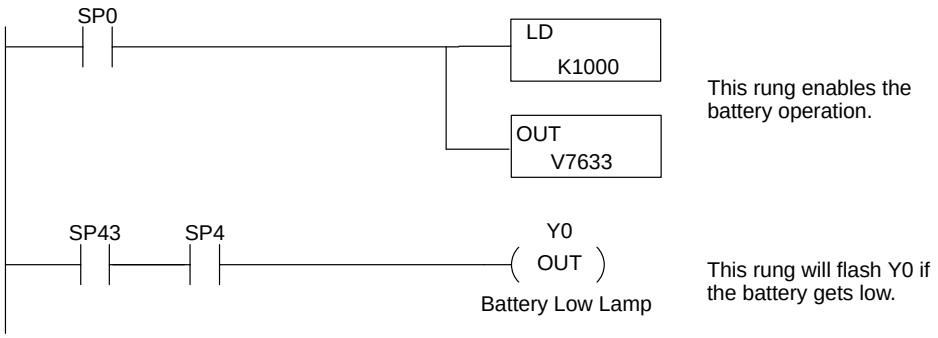
DL350

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

Enabling the Battery Backup

The battery can be enabled by setting bit 12 in V7633 ON (see example below). In this mode the battery Low LED will come on when the battery voltage is less than 2.5VDC (SP43) and error E41 will occur. In this mode the CPU will maintain the data in C,S,T,CT, and V memory when power is removed from the CPU, provided the battery is good. The use of a battery can also determine which operating mode is entered when the system power is connected. See CPU Setup, which is discussed later in this chapter.

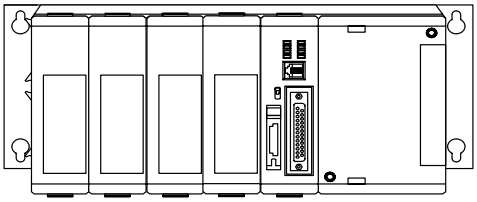
If you have installed a battery, the battery circuit can be disabled by turning off bit 12 in V7633. However, if you have a battery installed and select “No Battery” operation, the battery LED will not turn on if the battery voltage is low.



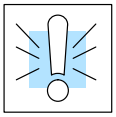
CPU Setup

Installing the CPU

The CPU **must** be installed in the first slot in the base (closest to the power supply). You cannot install the CPU in any other slot. When inserting the CPU into the base, align the PC board with the grooves on the top and bottom of the base. Push the CPU straight into the base until it is firmly seated in the backplane connector.



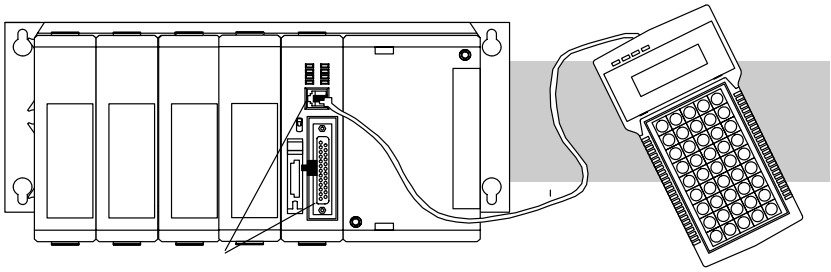
CPU must reside in first slot!



WARNING: To minimize the risk of electrical shock, personal injury, or equipment damage, always disconnect the system power before installing or removing any system component.

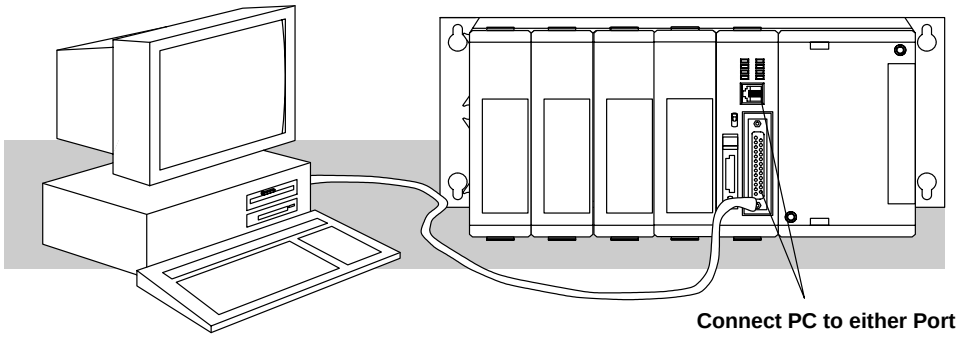
Connecting the Programming Devices

The Handheld programmer is connected to the CPU with a handheld programmer cable. You can connect the Handheld to port 1 on a DL350 CPU. The handheld programmer is shipped with a cable. The cable is approximately 6.5 feet (200 cm).



Connect Handheld to Port 1

If you are using a Personal Computer with the ***Direct***SOFT™ programming package, you can use either the top or bottom port.



Connect PC to either Port

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 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 different CPUs and the Handheld Programmer. Note, the Handheld Programmer may have additional AUX functions that are not supported with the DL305 CPUs.

AUX Function and Description		350	HPP
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	×	–
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 CPU Network Address	✓	–
57	Set Retentive Ranges	✓	–
58	Test Operations	×	–
59	Bit Override	×	–
5B	Counter Interface Config.	×	–
5C	Display Error History	✓	–

AUX Function and Description		350	HPP
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	✓	–

✓ supported

× not supported

– not applicable

Clearing an Existing Program

Before you enter a new program, you should 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

Setting the Clock and Calendar

The DL350 also has a Clock / Calendar that can be used for many purposes. If you need to use this feature there are also AUX functions available that allow you set the date and time. For example, you would use AUX 52, Display/Change Calendar to set the time and date with the Handheld Programmer. With ***Direct***SOFT you would use the PLC Setup menu options using K–Sequence protocol only.

The CPU uses the following format to display the date and time.

- Date — Year, Month, Date, Day of week (0 – 6, Sunday thru Saturday)
- Time — 24 hour format, Hours, Minutes, Seconds

Handheld Programmer Display

```
      23:08:17  97/05/20
```

You can use the AUX function to change any component of the date or time. However, the CPU will not automatically correct any discrepancy between the date and the day of the week. For example, if you change the date to the 15th of the month and the 15th is on a Thursday, you will also have to change the day of the week (unless the CPU already shows the date as Thursday). The day of the week can only be set using the handheld programmer.

Initializing System Memory

The DL350 CPU maintains system parameters in a memory area 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.

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 change from program to program without ever initializing system memory.

Remember, this AUX function will reset all system memory. If you have set special parameters such as retentive ranges, etc. they will be erased when AUX 54 is used. Make sure you that you have considered all ramifications of this operation before you select it.

Setting the CPU Network Address

The DL350 CPU has a built in ***Direct***NET port. You can use the Handheld Programmer to set the network address for the port and the port communication parameters. The default settings are:

- Station Address 1
- Hex Mode
- Odd Parity
- 9600 Baud

The ***Direct***NET Manual provides additional information about choosing the communication settings for network operation.

Setting Retentive Memory Ranges

The DL350 CPU provides 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	DL350	
	Default Range	Avail. Range
Control Relays	C1000 – C1777	C0 – C1777
V Memory	V1400 – 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.

WARNING: The DL350 CPU does 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. If the retentive ranges are important for your application, make sure you obtain the optional battery.

Password Protection

The DL350 CPU allows you to use a password to help minimize the risk of unauthorized program and/or data changes. The DL350 offers multi–level passwords for even more security. Once you enter a password you can “lock” the CPU 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 CPUs 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 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.

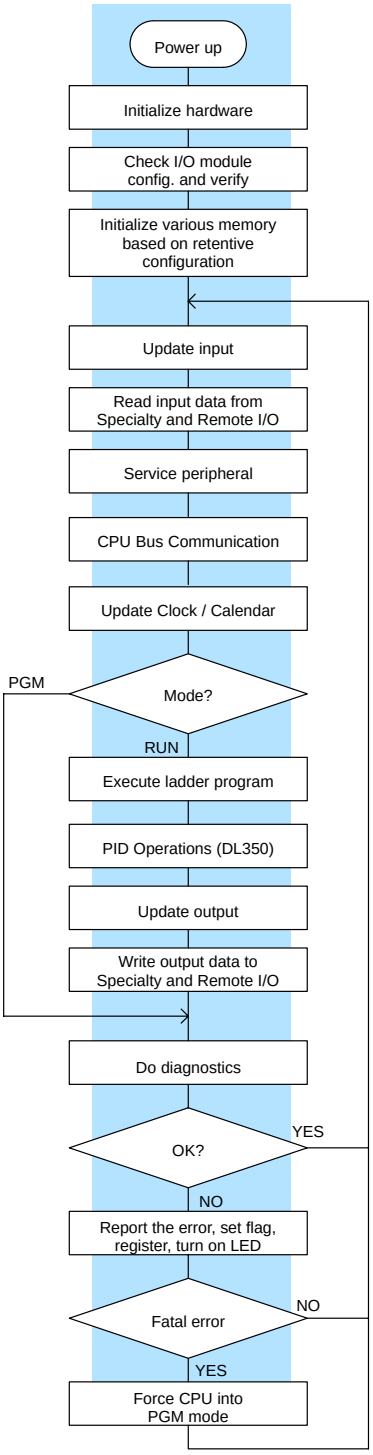
For more information on passwords, see the appropriate appendix on auxiliary functions.

WARNING: Make sure you remember your password. If you forget your password you will not be able to access the CPU. The CPU must be returned to the factory to have the password removed.

CPU Operation

Achieving the proper control for your equipment or process requires a good understanding of how DL350 CPUs control all aspects of system operation. The flow chart below shows the main tasks of the CPU operating system. In this section, we will investigate four aspects of CPU operation:

- CPU Operating System — the CPU manages all aspects of system control.
- CPU Operating Modes — The three primary modes of operation are Program Mode, Run Mode, and Test Mode.
- CPU Timing — The two important areas we discuss are the I/O response time and the CPU scan time.
- CPU Memory Map — The CPUs memory map shows the CPU addresses of various system resources, such as timers, counters, inputs, and outputs.



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.

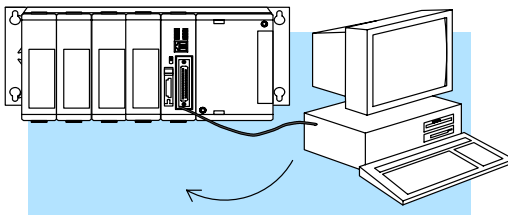
The outputs are only updated in Run mode. In program mode, they are in the off state.

In Run Mode, the CPU executes the user ladder program. Immediately afterwards, any PID loops which are configured are executed (DL350 only). Then the CPU writes the output results of these two tasks to the appropriate output points.

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 Operation

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



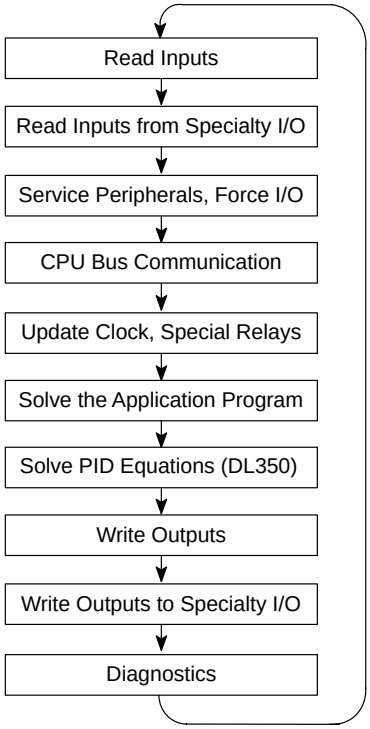
Download Program

You can use the mode switch on the DL350 CPU to select Program Mode operation. Or, with the switch in TERM position, you can use a programming device such as the Handheld Programmer to place the CPU in Program Mode.

Run Mode Operation

In Run Mode, the CPU executes the application program, does PID calculations for configured PID loops (DL350 only), 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
- Update timer/counter preset values
- Update Variable memory locations



Run Mode operation can be divided into several key areas. It is very important you understand how each of these areas of execution can affect the results of your application program solutions.

You can use the mode switch to select Run Mode operation. Or, with the mode switch in TERM position, you can use a programming device, such as the Handheld Programmer to place the CPU in Run Mode.

You can also edit the program during Run Mode. The Run Mode Edits are not “bumpless”. Instead, the CPU 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.

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.

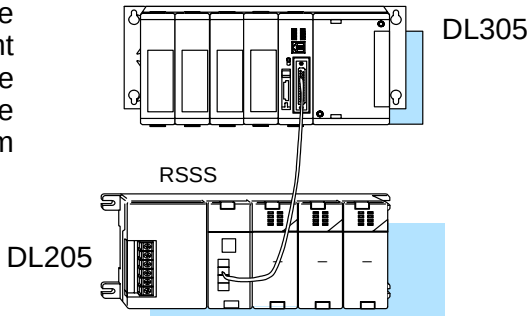
Read Inputs

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 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 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.

Read Inputs from Specialty and Remote I/O

After the CPU reads the inputs from the input modules, it reads any input point data from any Specialty modules that are installed. This is also the portion of the scan that reads the input status from Remote I/O racks.



NOTE: It may appear the Remote I/O point status is updated every scan. This is not quite true. The CPU will receive information from the Remote I/O Master module every scan, but the Remote Master may not have received an update from all the Remote slaves. Remember, the Remote I/O link is managed by the Remote Master, not the CPU.

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.

- Forcing from a peripheral – not a permanent force, good only for one scan

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.

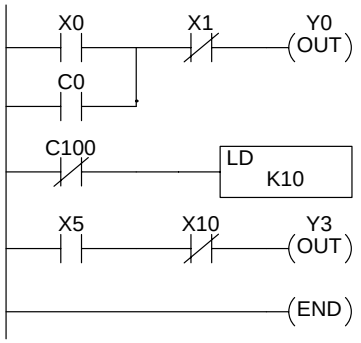
Update Clock, Special Relays, and Special Registers

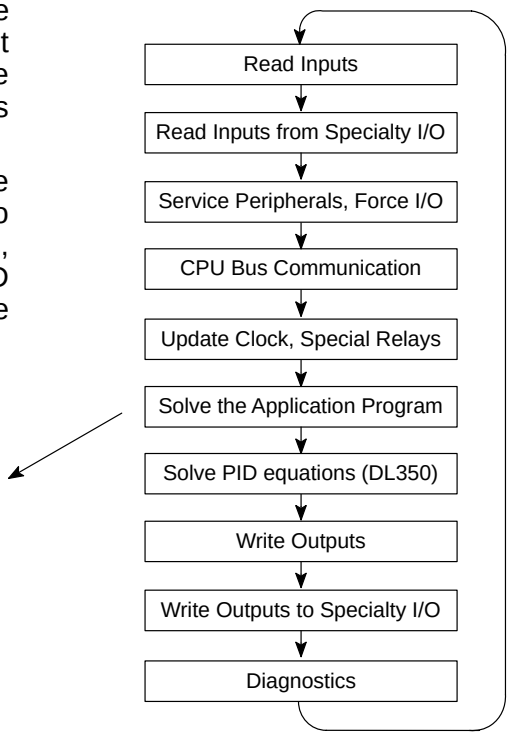
The DL350 CPUs has an internal real-time clock and calendar timer which is accessible to the application program. Special V-memory locations hold this information. This portion of the execution cycle makes sure these locations get updated on every scan. Also, there are several different Special Relays, such as diagnostic relays, etc., 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 input conditions and the system outputs.

The CPU begins with the first rung of the ladder program, evaluating it from left to right and from top to bottom. It continues, rung by rung, until it encounters the END coil instruction. At that point, a new image for the outputs is complete.





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

You may recall the CPU may have obtained and stored forcing information when it serviced the peripheral devices. If any I/O points or memory data have been forced, the output image register also contains this information.

NOTE: If an output point was used in the application program, the results of the program solution will overwrite any forcing information that was stored. For example, if Y0 was forced on by the programming device, and a rung containing Y0 was evaluated such that Y0 should be turned off, then the output image register will show that Y0 should be off. Of course, you can force output points that are not used in the application program. In this case, the point remains forced because there is no solution that results from the application program execution.

Solve PID Loop Equations

The DL350 CPU can process up to 4 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 located in the local CPU base or the local expansion bases. Remember, the CPU also made sure 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 and Remote 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. For example, this is the portion of the scan that writes the output status from the image register to the Remote I/O racks.

NOTE: It may appear the Remote I/O point status is updated every scan. This is not quite true. The CPU will send the information to the Remote I/O Master module every scan, but the Remote Master will update the actual remote modules during the next communication sequence between the master and slave modules. Remember, the Remote I/O link communication is managed by the Remote Master, not the CPU.

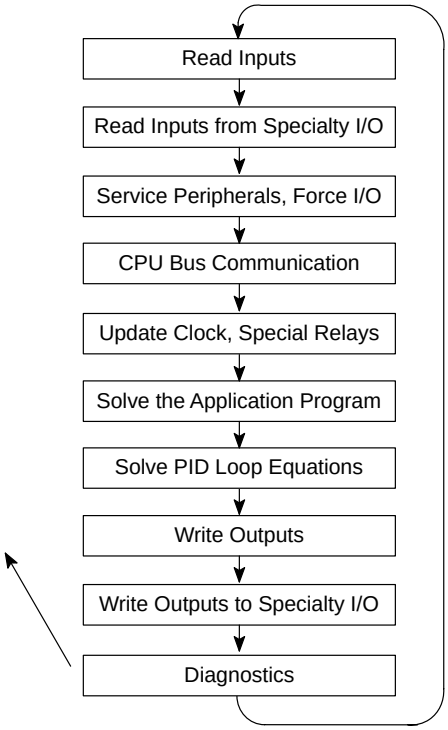
Diagnostics

During this part of the scan, the CPU performs all system diagnostics and other tasks, such as:

- calculating the scan time
- updating special relays
- resetting the watchdog timer

The DL350 CPU automatically detects and reports many different error conditions. Appendix B contains a listing of the various error codes available with the DL305 system.

One of the more important diagnostic tasks is the scan time calculation and watchdog timer control. The DL350 CPU has a “watchdog” timer that stores the maximum time allowed for the CPU to complete the solve application segment of the scan cycle. The default value set from the factory is 200 mS. If this time is exceeded the CPU will enter the Program Mode, turn off all outputs, and report the error. For example, the Handheld Programmer displays “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. There is also an RSTWT instruction that can be used in the application program to reset the watch dog timer during the CPU scan.

I/O Response Time

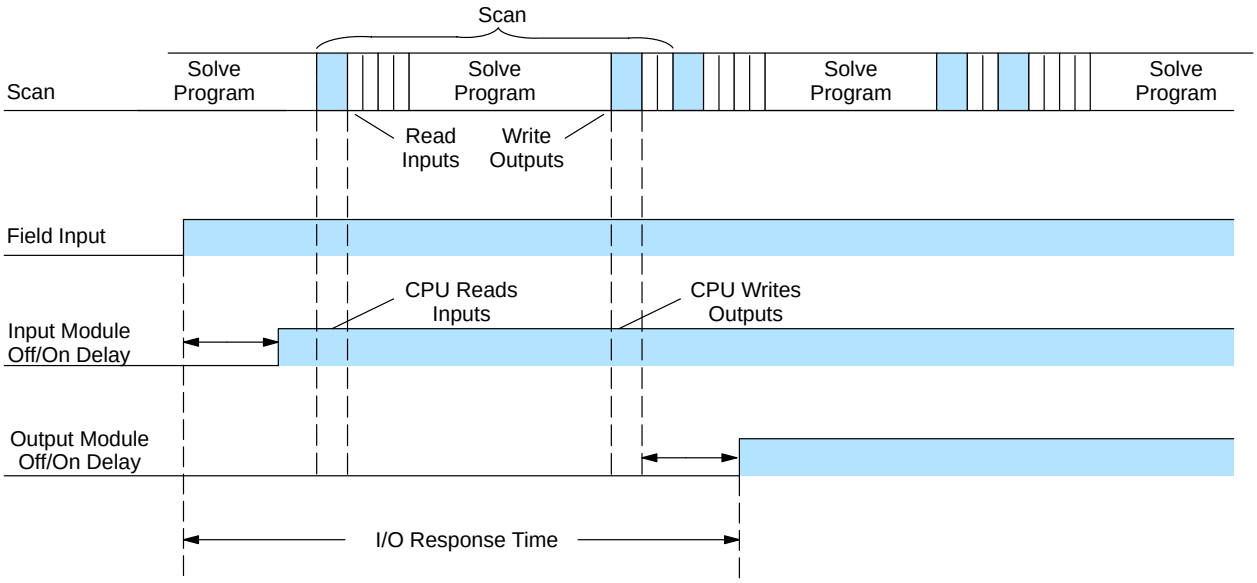
Is Timing Important for Your Application?

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 practically instantaneously. However, some applications do require extremely fast update times. There are four things that can affect the I/O response time:

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

Normal Minimum I/O Response

The I/O response time is shortest when the module senses the input change 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.



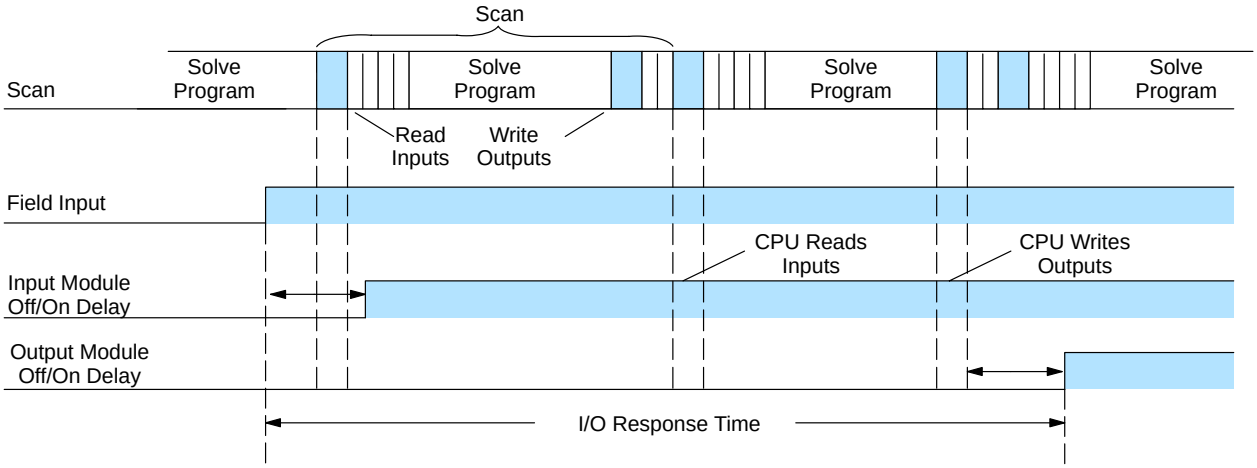
In this case, you can calculate the response time by simply adding the following items.

Input Delay + Scan Time + Output Delay = Response Time

Normal Maximum I/O Response

The I/O response time is longest when the module senses the input change after the Read Inputs portion of the execution cycle. In this case the new input status does not get 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

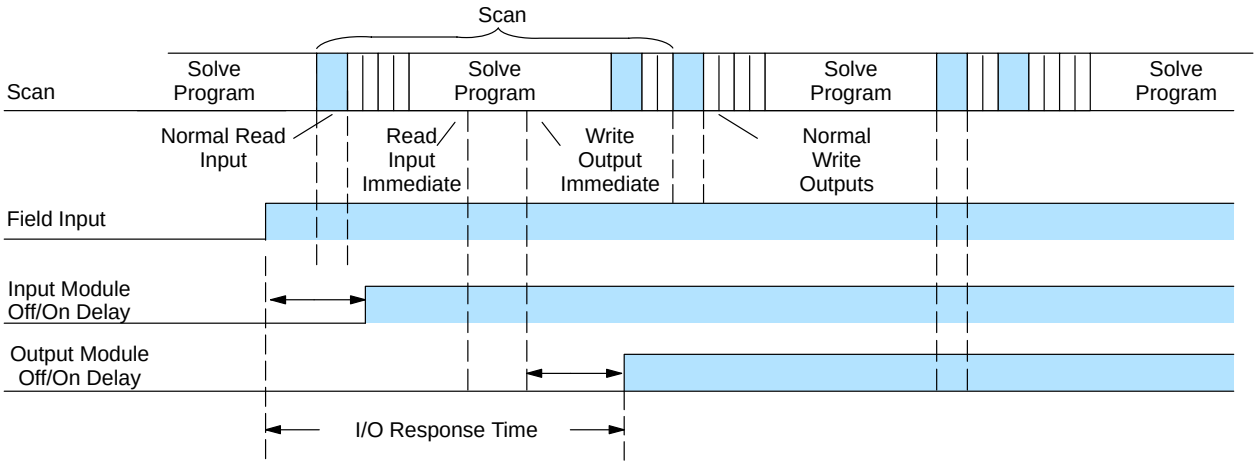


Improving Response Time

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

- Choose instructions with faster execution times
- Use immediate I/O instructions (which update the I/O points during the ladder program execution segment)
- Choose modules that have faster response times

Immediate I/O instructions are probably the most useful technique. The following example shows immediate input and output instructions, and their effect.



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 is calculated by adding the time for the immediate input instruction, the immediate output instruction, and all instructions in between.

NOTE: When the immediate instruction reads the current status from a module, it uses the results to solve that one instruction without updating the image register. Therefore, any regular instructions that follow will still use image register values. Any immediate instructions that follow will access the module again to update the status.

CPU Scan Time Considerations

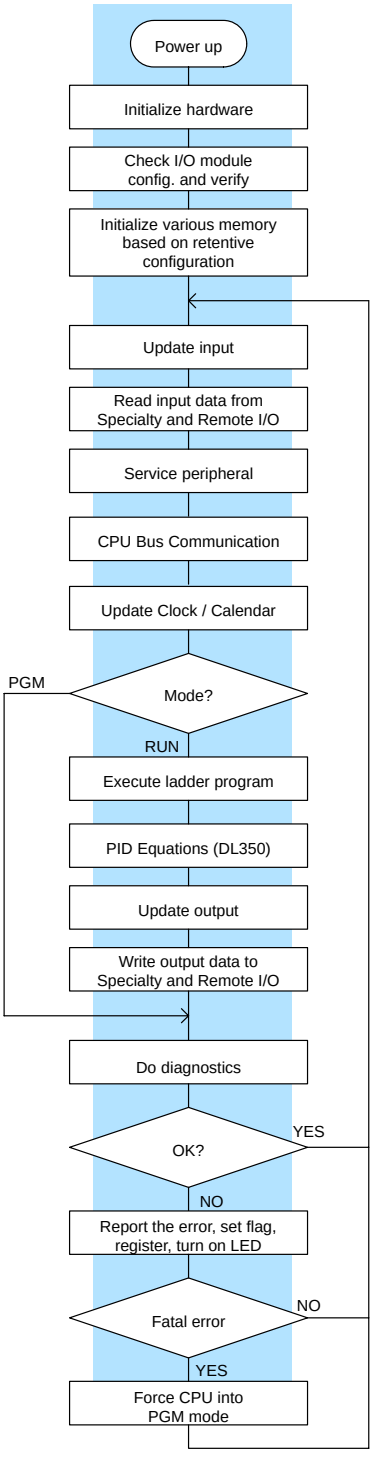
The scan time covers all the cyclical tasks that are performed by the operating system. You can use ***Direct***SOFT or the Handheld Programmer to display the minimum, maximum, and current scan times that have occurred since the previous Program Mode to Run Mode transition. This information can be very important when evaluating the performance of a system.

As 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 only one you really 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 modules and system configuration, such as expansion or remote I/O, can also affect the scan time. However, these things are usually dictated by the application.

For example, if you have a need to count pulses at high rates of speed, then you'll probably have to use a High-Speed Counter module. Also, if you have I/O points that need to be located several hundred feet from the CPU, then you need remote I/O because it's much faster and cheaper to install a single remote I/O cable than it is to run all those signal wires for each individual I/O point.

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



Intialization Process

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 Service Request	DL350
Minimum	1.2 µs
Maximum	1.5– µs

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)		DL350
Nothing Connected	Min. & Max.	0 µs
Port 1	Send Min. / Max.	6.8/12.6 µs
	Rec. Min. / Max.	9.2/972 ms
Port 2	Send Min. / Max.	6.8/12.6 µs
	Rec. Min. / Max.	9.2/972 ms

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.

NOTE: Some specialty modules can have a considerable impact on the CPU scan time. If timing is critical in your application, consult the module documentation for any information concerning the impact on the scan time.

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		DL350
Program Mode	Minimum	79.0 µs
	Maximum	79.0 µs
Run Mode	Minimum	79.0 µs
	Maximum	79.0 µs

Diagnostics

The DL305 CPUs perform many types of system diagnostics. The amount of time required depends on many things, such as the number of I/O modules installed, etc. The following table shows the minimum and maximum times that can be expected.

Diagnostic Time	DL350
Minimum	104.0 µs
Maximum	139.6 µs

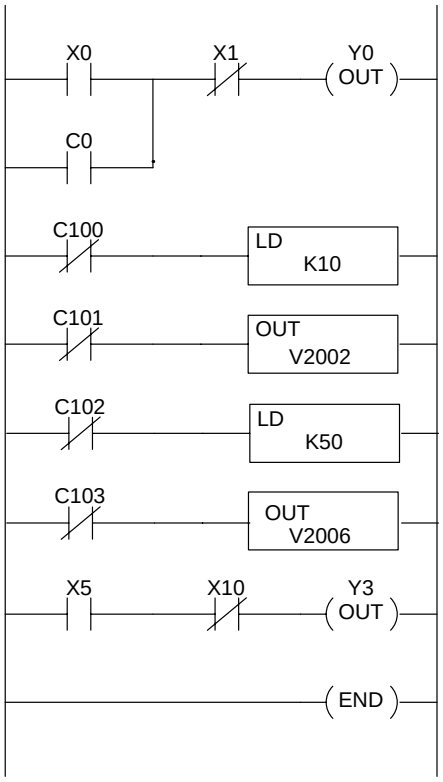
Application Program Execution

The CPU processes the program from the top (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.

You can add the execution times for all the instructions in your program to find the total program execution time.

For example, the execution time for a DL350 running the program shown would be calculated as follows.



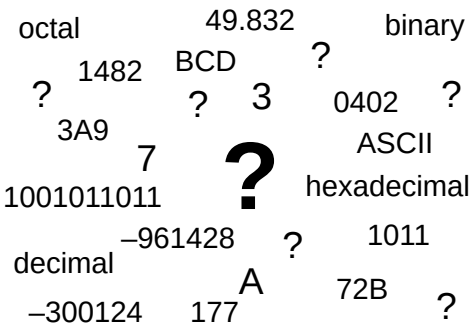
Instruction	Time
STR X0	1.4µs
OR C0	1.0µs
ANDN X1	1.2µs
OUT Y0	7.95µs
STRN C100	1.6µs
LD K10	62µs
STRN C101	1.6µs
OUT V2002	21.0µs
STRN C102	1.6µs
LD K50	62µs
STRN C103	1.6µs
OUT V2006	21.0µs
STR X5	1.4µs
ANDN X10	1.2µs
OUT Y3	7.95µs
END	16µs
TOTAL	210.5µs

Appendix C provides a complete list of instruction execution times for the DL350 CPU.

Program Control Instructions — the DL350 CPU offers 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 effect 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 PLC***Direct*** PLCs for the first time, please take a moment to study how our PLCs use numbers. You will find that each PLC manufacturer has their own conventions on the use of numbers in their PLCs. Take a moment to familiarize yourself with how numbers are used in PLC***Direct*** PLCs. The information you learn here applies to all our PLCs!

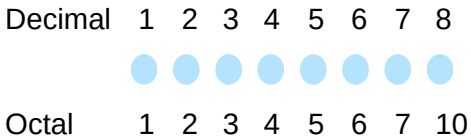


PLCs store and manipulate numbers in binary form: ones and zeros. So why do we have numbers in so many different forms? Numbers have meaning, and some *representations* are more convenient than others for particular purposes. Sometimes we use numbers to represent a size or amount of something. Other numbers refer to locations or addresses, or to time. In science we attach engineering units to numbers to give a particular meaning.

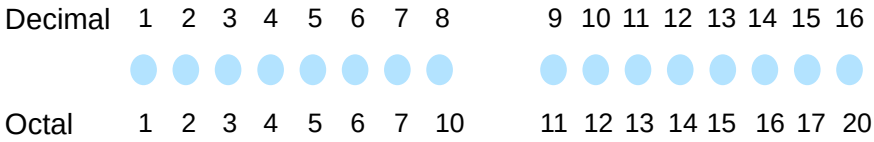
PLC Resources

PLCs offer a fixed amount of resources, depending on the model and configuration. The word “resources” includes 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.

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).

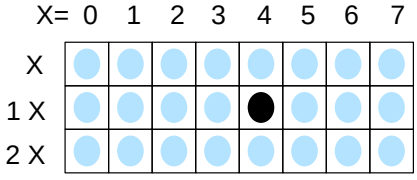


In the figure below, ther are two groups of eight circles. Counting in octal ther are “20” items, meaning 2 groups of eight, plus 0 individuals Avoid saying “twenty”, say “two–zero octal”. This makes a clear distinction between number systems.



After *counting* PLC resources, it’s time to *access* PLC resources (there is 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.

The circles are in an array of square containers to the right. To access a resource, the 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 (called “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. The word “significant”, refers 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 the data registers are rarely programmmed one bit at a time. Instructions or viewing tools work with binary, decimal, octal, and hexadecimal numbers. All of these are converted and stored as binary for us.

A frequently-asked question is “How do I tell if a number is binary, octal, BCD, or hex”? The answer is that we usually cannot tell 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 a storage box... that’s all. It does not convert or move the data on its own.

Binary-Coded Decimal Numbers

Since humans naturally count in decimal, we prefer to enter and view PLC data in decimal as well (via operator interfaces). 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.

BCD number	4				9				3				6			
	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
V-memory storage	0	1	0	0	1	0	0	1	0	0	1	1	0	1	1	0

In a pure binary sense, a 16-bit word represents numbers from 0 to 65535. In storing BCD numbers, the range is reduced to 0 to 9999. Many math instructions use BCD data, and ***Direct***SOFT and the handheld programmer allow us to enter and view data in BCD. Special RLL instructions convert from BCD to binary, or visa–versa.

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 a convenient way for humans to view full binary data.

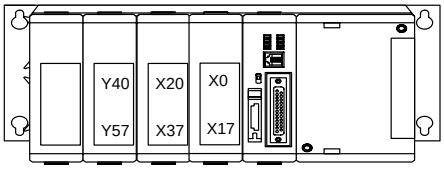
Hexadecimal number	A				7				F				4			
V-memory storage	1	0	1	0	0	1	1	1	1	1	1	1	0	1	0	0

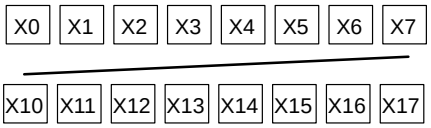
Memory Map

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 the DL350 CPU. A memory map overview follows the memory descriptions.

Octal Numbering System

All memory locations or areas 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.



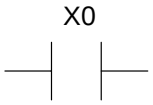


Discrete and Word Locations

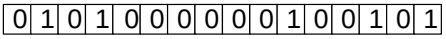
As you examine the different memory types, you'll notice two types of memory in the DL350, discrete and word memory. Discrete memory is one bit that can be either a 1 or a 0. Word memory is referred to as V memory (variable) and is a 16-bit location normally used to manipulate data/numbers, store data/numbers, etc.

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

Discrete – On or Off, 1 bit

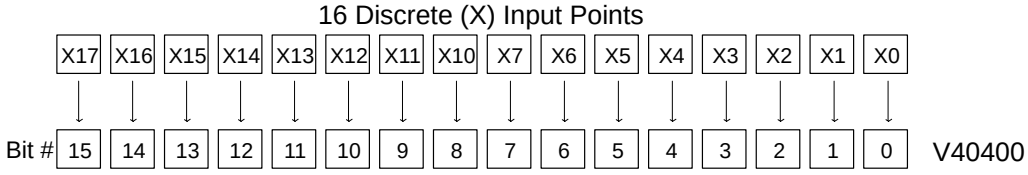


Word Locations – 16 bits



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 the DL350 CPU in this chapter.

Input Points (X Data Type)

The discrete input points are noted by an X data type. There are up to 512 discrete input points available with the DL350 CPU. In this example, the output point Y0 will be turned on when input X0 energizes.

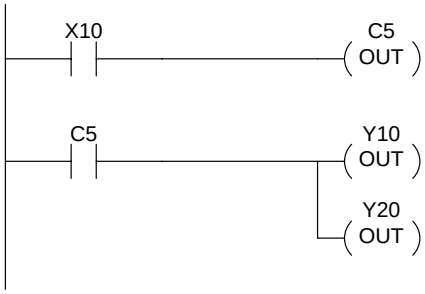
Output Points (Y Data Type)

The discrete output points are noted by a Y data type. There are up to 512 discrete output points available with the DL350 CPU. In this example, output point Y1 will turn 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. They are internal to the CPU. 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 has 16 consecutive discrete locations.

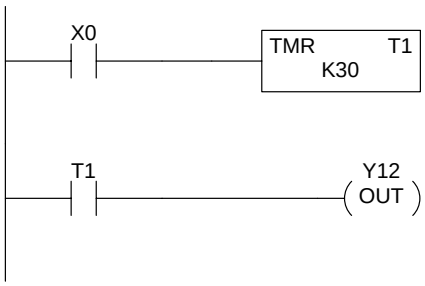
In this example, memory location C5 will energize when input X10 turns on. The second rung shows a simple example of how to use a control relay as an input.



Timers and Timer Status Bits (T Data type)

The amount of timers available depends on the model of CPU you are using. The tables at the end of this section provide the number of timers for the DL350. Regardless of the number of timers, you have access to timer status bits that 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.

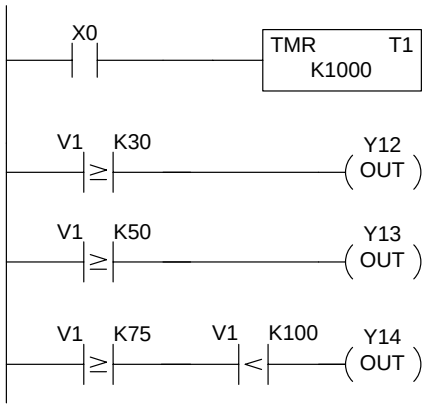
When input X0 turns on, timer T1 will start. When the timer reaches the preset of 3 seconds (K of 30) timer status contact T1 turns on. When T1 turns on, output Y12 turns on.



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, etc.

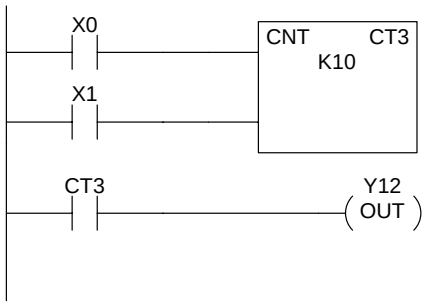
The primary reason for this is programming flexibility. The example shows how you can use relational contacts to monitor several time intervals from a single timer.



Counters and Counter Status Bits (CT Data type)

The amount of counters available depends on the model of CPU you are using. The tables at the end of this section provide the number of counters for the DL350. Regardless of the number of counters, you have access to 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.

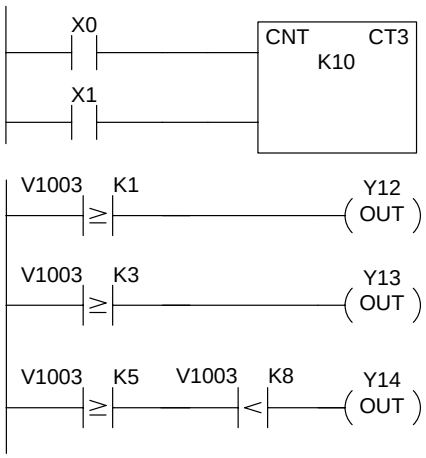
Each time contact X0 transitions from off to on, the counter increments by one. If X1 comes on, the counter is reset to zero. When the counter reaches the preset of 10 counts (K of 10) counter status contact CT3 turns on. When CT3 turns on, output Y12 turns on.



Counter Current Values (V Data Type)

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.

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

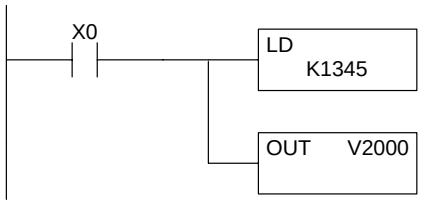


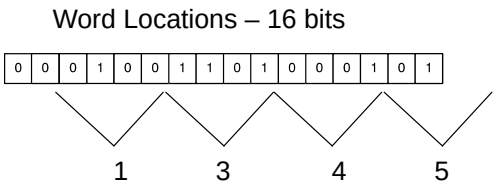
Word Memory (V Data Type)

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.

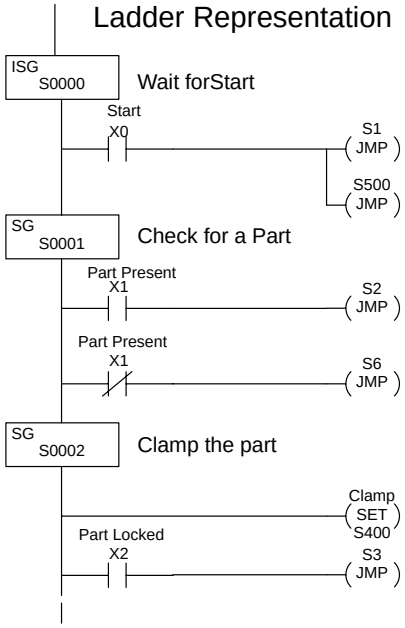




Stages (S Data type)

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.



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

DL350 System V-memory

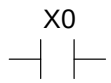
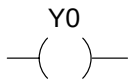
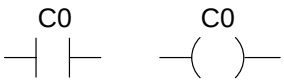
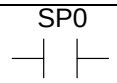
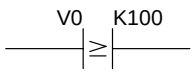
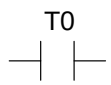
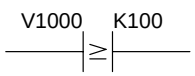
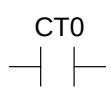
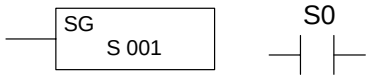
System V-memory	Description of Contents	Default Values / Ranges
V7620–V7627	Locations for DV–1000 operator interface parameters	
V7620	Sets the V-memory location that contains the value.	V0 – V3777
V7621	Sets the V-memory location that contains the message.	V0 – V3777
V7622	Sets the total number (1 – 16) of V-memory locations to be displayed.	1 – 16
V7623	Sets the V-memory location that contains the numbers to be displayed.	V0 – V3777
V7624	Sets the V-memory location that contains the character code to be displayed.	V0 – V3777
V7625	Contains the function number that can be assigned to each key.	V-memory for X, Y, or C
V7626	Reserved	0,1,2,3,12
V7627	Reserved	Default=0000
V7630–V7632	Reserved	–
V7633	User defined timer interrupt/operation of battery/Binary instruction sign flag* Bit 0–7 40H Setting Interrupt Bit 12 ON with battery sign flag. ON use sign flag – OFF no sign flag Bit 15 Binary instruction sign flag. ON use sign flag – OFF no sign flag	
V7634	User defined timer interrupt	
V7640	Loop Table Beginning address	V1400–V7340
V7641	Number of Loops Enabled	1–4
V7642	Error Code – V–memory Error Location for Loop Table	
V7643–V7647	Reserved	
V7650	Port 2 End–code setting Setting (A55A), Nonprocedure communications start.	
V7651	Port 2 Data format –Non–procedure communications format setting.	
V7652	Port 2 Format Type setting – Non–procedure communications type code setting.	
V7653	Port 2 Terminate–code setting – Non–procedure communications Termination code setting.	
V7654	Port 2 Store V–mem address – Non–procedure communication data store V–Memory address.	
V7655	Port 2 Setup area –0–7 Comm protocol (flag 0) 8–15 Comm time out/response delay time (flag 1)	
V7656	Port 2 setup area – 0–15 Communication (flag2, flag 3)	
V7657	Port 2 setup area – Bit to select use of parameter	
V7660–V7707	Set–up Information	
V7710–V7717	Reserved	
V7720–V7722	Locations for DV–1000 operator interface parameters.	
V7720	Titled Timer preset value pointer	
V7721	Title Counter preset value pointer	
V7722	HiByte-Titled Timer preset block size, LoByte-Titled Counter preset block size	
V7730–V7737	For slot 0 to 7 D3–DCM	
V7747	Location contains a 10ms counter. This location increments once every 10ms.	
V7750	Reserved	

System V-memory	Description of Contents
V7751	Fault Message Error Code — stores the 4-digit code used with the FAULT instruction when the instruction is executed.
V7752	Reserved
V7753	Reserved
V7754	Reserved
V7755	Error code — stores the fatal error code.
V7756	Error code — stores the major error code.
V7757	Error code — stores the minor error code.
V7760–V7762	Reserved
V7763–V7764	Location for syntax error information.
V7765	Scan — stores the total number of scan cycles that have occurred since the last Program Mode to Run Mode transition.
V7766	Contains the number of seconds on the clock. (00 to 59).
V7767	Contains the number of minutes on the clock. (00 to 59).
V7770	Contains the number of hours on the clock. (00 to 23).
V7771	Contains the day of the week. (Mon, Tue, etc.).
V7772	Contains the day of the month (1st, 2nd, etc.).
V7773	Contains the month. (01 to 12)
V7774	Contains the year. (00 to 99)
V7775	Scan — stores the current scan time (milliseconds).
V7776	Scan — stores the minimum scan time that has occurred since the last Program Mode to Run Mode transition (milliseconds).
V7777	Scan — stores the maximum scan time that has occurred since the last Program Mode to Run Mode transition (milliseconds).

The following system control relays are valid only for D3–350 CPU remote I/O setup on Communications Port 2.

System CRs	Description of Contents
C740	Completion of setups – ladder logic must turn this relay on when it has finished writing to the Remote I/O setup table
C741	Erase received data – turning on this flag will erase the received data during a communication error.
C743	Re-start – Turning on this relay will resume after a communications hang-up on an error.
C750 to C757	Setup Error – The corresponding relay will be ON if the setup table contains an error (C750 = master, C751 = slave 1... C757=slave 7
C760 to C767	Communications Ready – The corresponding relay will be ON if the setup table data is valid (C760 = master, C761 = slave 1... C767=slave 7

DL350 Memory Map

Memory Type	Discrete Memory Reference (octal)	Word Memory Reference (octal)	Qty. 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	
Timer Current Values	None	V0 – V377	256	
Timer Status Bits	T0 – T377	V41100 – V41117	256	
Counter Current Values	None	V1000 – V1177	128	
Counter Status Bits	CT0 – CT177	V41140 – V41147	128	
Data Words	none	V1400 – V7377 V10000–V17777	3072 4096	None specific, used with many instructions
Stages	S0 – S1777	V41000 – V41077	1024	
System parameters	None	V7400–V7777	256	System specific, used for various purposes

X Input / Y Output Bit Map

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

MSB								DL350 Input (X) and Output (Y) Points							LSB	X Input Address	Y Output Address
17	16	15	14	13	12	11	10	7	6	5	4	3	2	1	0		
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

Control Relay Bit Map

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

MSB								DL350 Control Relays (C)							LSB	Address
17	16	15	14	13	12	11	10	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

MSB Additional DL350 Control Relays (C) LSB																Address
17	16	15	14	13	12	11	10	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

Stage™ Control / Status Bit Map

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

MSB								DL350 Stage (S) Control Bits							LSB	Address
17	16	15	14	13	12	11	10	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

MSB								DL350 Additional Stage (S) Control Bits (continued)							LSB	Address
17	16	15	14	13	12	11	10	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

Timer and Counter Status Bit Maps

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

MSB								DL350 Timer (T) and Counter (CT) Contacts							LSB	Timer Address	Counter Address
17	16	15	14	13	12	11	10	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	V41140
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V41101	V41141
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V41102	V41142
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V41103	V41143
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V41104	V41144
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V41105	V41145
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V41106	V41146
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V41107	V41147

This portion of the table shows additional Timer contacts available with the DL350.

MSB								DL350 Additional Timer (T) Contacts							LSB	Timer Address
17	16	15	14	13	12	11	10	7	6	5	4	3	2	1	0	
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

4

System Design and Configuration

In This Chapter. . . .

- DL305 System Design Strategies
- Module Placement
- Calculating the Power Budget
- Expansion I/O
- Remote I/O
- Network Connections to MODBUS and ***Direct***NET
- Network Slave Operation
- Network Master Operation

DL305 System Design Strategies

I/O System Configurations

The DL350 CPU offers the following ways to add I/O to the system:

- **Local I/O** – consists of I/O modules located in the same base as the CPU.
- **Remote I/O** – consists of I/O modules located in bases which are serially connected to the bottom port on a DL350 CPU.
- **Expansion I/O** – consists of I/O modules located in expansion bases located close to the local base. Expansion cables connect them to the local CPU base's serial bus in a daisy–chain fashion.

A DL305 system can be developed using many different arrangements of these configurations. All I/O configurations use the standard complement of DL305 I/O modules and bases.

Networking Configurations

The DL350 CPU offers the following way to add networking to the system:

- **DL350 Communications Port** – The DL350 CPU has a 25–Pin connector on Port 2 that provides a built–in RTU MODBUS connection.
- **MODBUS Master Module**– MODBUS master modules can be used in any slot for connecting as a master to a MODBUS network.
- **MODBUS Slave Module**– MODBUS slave modules can be used in any slot for connecting as a slave to a MODBUS network.

Module/Unit	Master	Slave
DL350 CPU	***Direct***NET MODBUS RTU	***Direct***NET K–Sequence MODBUS RTU

Base Configurations

The DL305 system currently offers two types of bases. Both types come in 5, 8, or 10 slot configurations. All DL305 CPUs will work in either type of base. The xxxxx–1 bases are designed to compliment the features of the DL350 CPU, however all other DL305 CPUs will work in these bases. You can also mix the bases in a system. By mixing the bases or by installing the DL350 in an conventional base, you will loose some of the features of the CPU. The DL350 will revert back to 8–bit addressing and will virtually function like a DL340 CPU. This section will focus on the xxxxx–1 bases using the DL350 CPU. If you will be using the DL350 in a conventional base or if you are mixing bases in a system, refer to Appendix F for base, I/O, and module placement information.

The xxxxx–1 bases support a 8 bit parallel bus that allows the use of intelligent modules when using the DL350 CPU. The addressing scheme is simplified and also extends the number of I/O points you can use. You will have a bigger power budget to work with due to the increase in the power supply capacity to 2.0A.

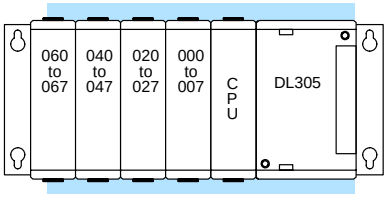
Module Placement

Slot Numbering

The DL305 bases each provide different numbers of slots for use with the I/O modules. You may notice the bases refer to 5-slot, 8-slot, etc. One of the slots is dedicated to the CPU, so you always have one less I/O slot. For example, you have four I/O slots with a 5-slot base. The I/O slots are numbered 0 – 3. The CPU slot always contains a CPU and is not numbered.

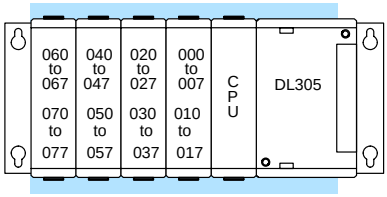
The examples below show the I/O numbering for a 5 slot local CPU base with 8 point I/O and a 5 slot local CPU base with 16 point I/O.

5 Slot Base Using 8 Point I/O Modules



Slot Number: 3—2—1—0

5 Slot Base Using 16 Point I/O Modules



Slot Number: 3—2—1—0

I/O Module Placement Rules

There are some limitations that determine where you can place certain types of modules. Some modules require certain locations and may limit the number or placement of other modules. The table on pages 4-6 and 4-7 should clear up any gray areas in the explanation and you will probably find the configuration you intend to use in your installation.

In all of the configurations mentioned the number of slots from the CPU that are to be used can roll over into an expansion base if necessary. For example if a rule states a module must reside in one of the six slots adjacent to the CPU, and the system configuration is comprised of two 5 slot bases, slots 1 and 2 of the expansion base are valid locations.

The following table provides the general placement rules for the DL305 components.

Module	Restriction
CPU	The CPU must reside in the first slot of the local CPU base. The first slot is the closest slot to the power supply.
16 Point I/O Modules	Any slot.
Analog Modules	Any slot.
ASCII Basic Modules	Any slot.
High Speed Counter	The D3–350 CPU does not support a high speed counter module.

I/O Configuration

I/O addresses use octal numbering, starting in the slot next to the CPU. The addresses are assigned in groups of 16 for each slot regardless of what module is in the slot. The discrete input and output modules can be mixed in any order, but there may be restrictions placed on some specialty modules.

Calculating the Power Budget

Managing your Power Resource

When you determine the types and quantity of I/O modules you will be using in the DL305 system it is important to remember there is a limited amount of power available from the power supply. We have provided a chart to help you easily see the amount of power available with each base. The following chart will help you calculate the amount of power you need with your I/O selections. At the end of this section you will also find an example of power budgeting and a worksheet for your own calculations.

WARNING: It is *extremely* important to calculate the power budget. If you exceed the power budget, the system may operate in an unpredictable manner which may result in a risk of personal injury or equipment damage.

Base Power Specifications

This chart shows the amount of current available for the three voltages supplied on the new xxxxx–1 bases. Use these currents when calculating the power budget for your system.

Bases	5V Power Supplied in Amps	9V Power Supplied in Amps	24V Power Supplied in Amps	Auxiliary 24 VDC Output at Base Terminal
D3–05B–1	1.0A (50°C) 0.7A (60°C)	2.0	0.6	100mA max
D3–05BDC	1.4A (50°C) 0.7A (60°C)	0.8	0.6	None
D3–08B–1	1.0A (50°C) 0.7A (60°C)	2.0	0.6	100mA max
D3–10B–1	1.0A (50°C) 0.7A (60°C)	2.0	0.6	100mA max
D3–10BDC	1.4A (50°C) 0.7A (60°C)	1.7	0.6	None

I/O Points Required for Each Module

Each type of module requires a certain number of I/O points. This is also true for the specialty modules, such as analog, counter interface, etc. The table on page 4–5 lists the number and type of I/O points required for each module.

Module Power Requirements

The next three pages show the amount of maximum current required for each of the DL305 modules. The column labeled “External Power Source Required” is for module operation and is not for field wiring. Use these currents when calculating the power budget for your system. If 24 VDC is needed for external devices, the 24 VDC (100mA maximum) output at the base terminal strip may be used as long as the power budget is not exceeded.

	I/O Points Required	5V Power Required (mA)	9V Power Required in (A)	24V Power Required (mA)	External Power Source Required
CPUs					
D3–350		500	20	0	None
DC Input Modules					
D3–08ND2	8	0	10	112	None
D3–16ND2–1	16	0	25	224	None
D3–16ND2–2	16	0	24	209	None
D3–16ND2F	16	0	25	224	None
F3–16ND3F	16	0	148	68	None
AC Input Modules					
D3–08NA–1	8	0	10	0	None
D3–08NA–2	8	0	10	0	None
D3–16NA	16	0	100	0	None
AC/DC Input Modules					
D3–08NE3	8	0	10	0	None
D3–16NE3	16	0	130	0	None
DC Output Modules					
D3–08TD1	8	0	20	24	None
D3–08TD2	8	0	30	0	None
D3–16TD1–1	16	0	40	96	None
D3–16TD1–2	16	0	40	96	None
D3–16TD2	16	0	180	0	None
AC Output Modules					
D3–04TAS	8	0	12	0	None
F3–08TAS	8	0	80	0	None
D3–08TA–1	8	0	96	0	None
D3–08TA–2	8	0	160	0	None
F3–16TA–1	16	0	160	0	None
D3–16TA–2	16	0	400	0	None

	I/O Point Required	5V Power Required in mA	9V Power Required in mA	24V Power Required in mA	External Power Source Required
Relay Output Modules					
D3–08TR	8	0	360	0	None
F3–08TRS–1	8	0	296	0	None
F3–08TRS–2	8	0	296	0	None
D3–16TR	16	0	480	0	None
Analog					
D3–04AD	16	0	55	0	24VDC @ 65mA max
F3–04ADS	16	0	183	50	None
F3–08AD	16	0	25	37	None
F3–08TEMP	16	0	25	37	None
F3–08THM–n	16	0	50	34	None
F3–16AD	16	0	33	47	None
D3–02DA	16	0	80	0	24VDC @ 170mA max
F3–04DA–1	16	0	144	108	None
F3–04DA–2	16	0	144	108	None
F3–04DAS	16	0	154	145	None
Communications and Networking					
FA–UNICON	0	0	0	0	(24 VDC or 5 VDC) @ 100mA
ASCII BASIC Modules					
F3–AB128–R	16	0	205	0	None
F3–AB128–T	16	0	205	0	None
F3–AB128	16	0	90	0	None
F3–AB64	16	0	90	0	None
Specialty Modules					
D3–08SIM	8	0	10	112	None
D3–HSC	16	0	70	0	None
Programming					
D2–HPP		200	50	0	Optional

Power Budget Calculation Example

The following example shows how to calculate the power budget for the DL305 system.

Base # 0	Module Type	5 VDC (mA)	9 VDC (mA)	Auxiliary Power Source 24 VDC Output (mA)
Available Base Power	**D3–05B**	**1000**	**2000**	**600**
CPU Slot	D3–350	+500	+ 120	
Slot 0	D3–16NE3	+ 0	+ 130	+ 0
Slot 1	D3–16NE3	+ 0	+ 130	+ 0
Slot 2	F3–16TA–1	+ 0	+ 160	+ 0
Slot 3	F3–16TA–1	+ 0	+ 160	+ 0
Slot 4				
Slot 5				+ 0
Slot 6				+ 0
Slot 7				+ 0
Other				
Handheld Prog	D2–HPP	+ 200	+ 200	+ 0
Total Power Required		**700**	**900**	**0**
Remaining Power Available		1000–700=300	2000–900=1100	600 – 0 = **600**

1. Use the power budget table to fill in the power requirements for all the system components. First, enter the amount of power supplied by the base. Next, list the requirements for the CPU, any I/O modules, and any other devices, such as the Handheld Programmer or the DV–1000 operator interface. Remember, even though the Handheld or the DV–1000 are not installed in the base, they still obtain their power from the system. Also, make sure you obtain any *external* power requirements, such as the 24VDC power required by the analog modules.
2. Add the current columns starting with Slot 0 and put the total in the row labeled "**Total power required**"**.**
3. Subtract the row labeled "**Total power required**" from the row labeled "**Available Base Power**". Place the difference in the row labeled **"Remaining Power Available**".
4. If **"Total Power Required"** is greater than the power available from the base, the power budget will be exceeded. It will be unsafe to used this configuration and you will need to restructure your I/O configuration.

WARNING: It is *extremely* important to calculate the power budget. If you exceed the power budget, the system may operate in an unpredictable manner which may result in a risk of personal injury or equipment damage.

Power Budget Calculation Worksheet

This blank chart is provided for you to copy and use in your power budget calculations.

Base # 0	**Module Type**	**5 VDC (mA)**	**9 VDC (mA)**	**Auxiliary Power Source 24 VDC Output (mA)**
Available Base Power				
CPU Slot				
Slot 0				
Slot 1				
Slot 2				
Slot 3				
Slot 4				
Slot 5				
Slot 6				
Slot 7				
Other				
Handheld Prog	D2–HPP			
Total Power Required				
Remaining Power Available				

1. Use the power budget table to fill in the power requirements for all the system components. First, enter the amount of power supplied by the base. Next, list the requirements for the CPU, any I/O modules, and any other devices, such as the Handheld Programmer or the DV–1000 operator interface. Remember, even though the Handheld or the DV–1000 are not installed in the base, they still obtain their power from the system. Also, make sure you obtain any *external* power requirements, such as the 24VDC power required by the analog modules.
2. Add the current columns starting with Slot 0 and put the total in the row labeled "**Total power required**"**.**
3. Subtract the row labeled "**Total power required**" from the row labeled "**Available Base Power**". Place the difference in the row labeled **"Remaining Power Available**".
4. If **"Total Power Required"** is greater than the power available from the base, the power budget will be exceeded. It will be unsafe to used this configuration and you will need to restructure your I/O configuration.

WARNING: It is *extremely* important to calculate the power budget. If you exceed the power budget, the system may operate in an unpredictable manner which may result in a risk of personal injury or equipment damage.

Local I/O Expansion

Base Uses Table

It is helpful to understand how you can use the various DL305 bases in your control system. The following table shows how the bases can be used.

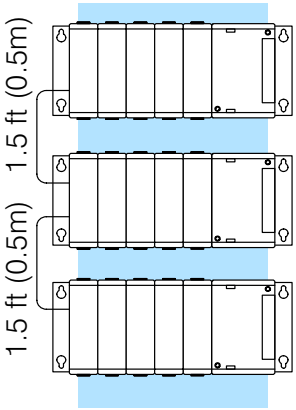
Base Part #	Number of Slots	Can Be Used As A Local CPU Base	Can Be Used As An Expansion Base
D3–05B–1	5	Yes	Yes
D3–05BDC–1	5	Yes	Yes
D3–08B–1	8	Yes	Yes
D3–08BDC–1	8	Yes	Yes
D3–10B–1	10	Yes	Yes
D3–10BDC–1	10	Yes	Yes

Local/Expansion Connectivity

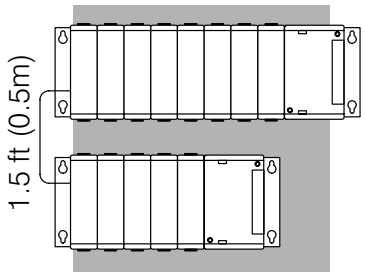
The configurations below show the valid combinations of local and expansion bases using the DL350 CPU.

NOTE: You should use one of the configurations listed below when designing an expansion system. If you use a configuration not listed below the system will not function properly.

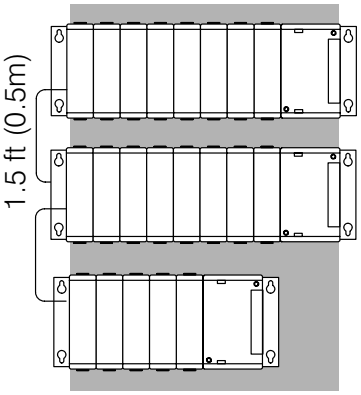
5 slot local CPU base with a maximum of two 5 slot expansion bases



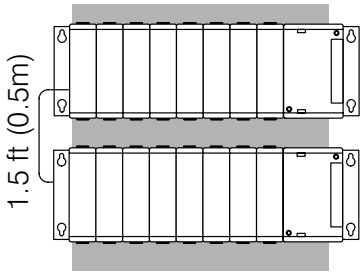
8 slot local CPU base with a 5 slot expansion base

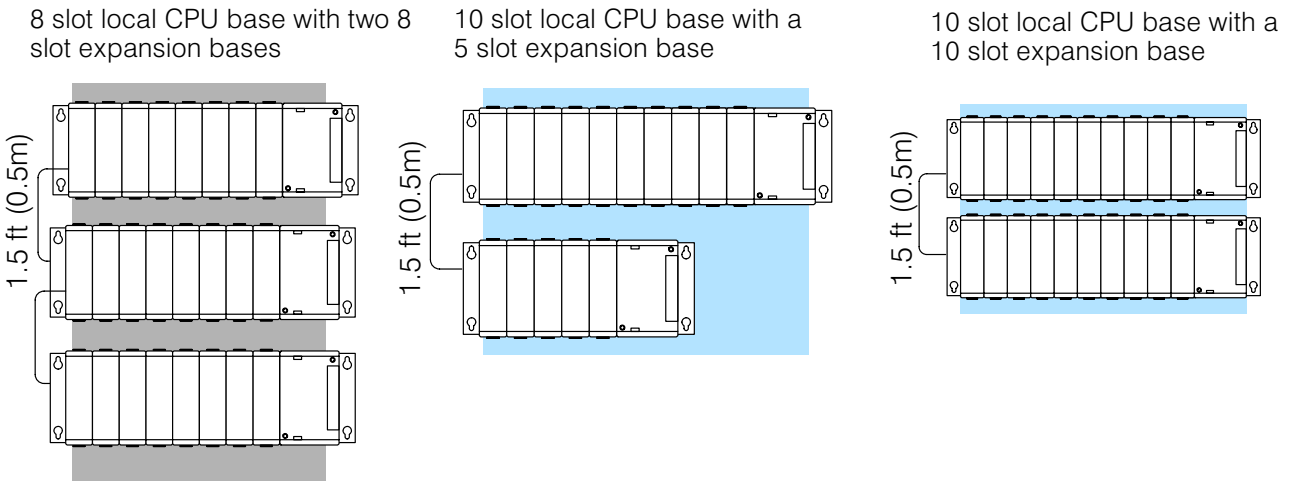


8 slot local CPU base with a 8 slot and 5 slot expansion base



8 slot local CPU base with a 8 slot expansion base

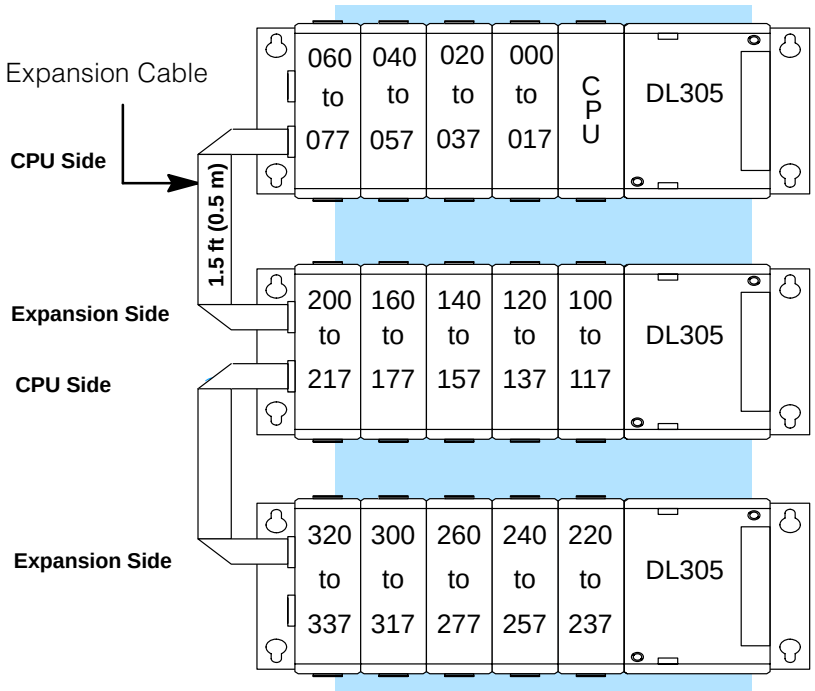




Connecting Expansion Bases

The local CPU base is connected to the expansion base using a 1.5 ft. cable (D3–EXCBL). The base must be connected as shown in the diagram below.

The top expansion connector on the base is the input from a previous base. The bottom expansion connector on the base is the output to an expansion base. The expansion cable is marked with “CPU Side” and “Expansion Side”. The“ CPU Side” of the cable is connected to the bottom port of the base and the “Expansion Side” of the cable is connected to the top port of the next base.

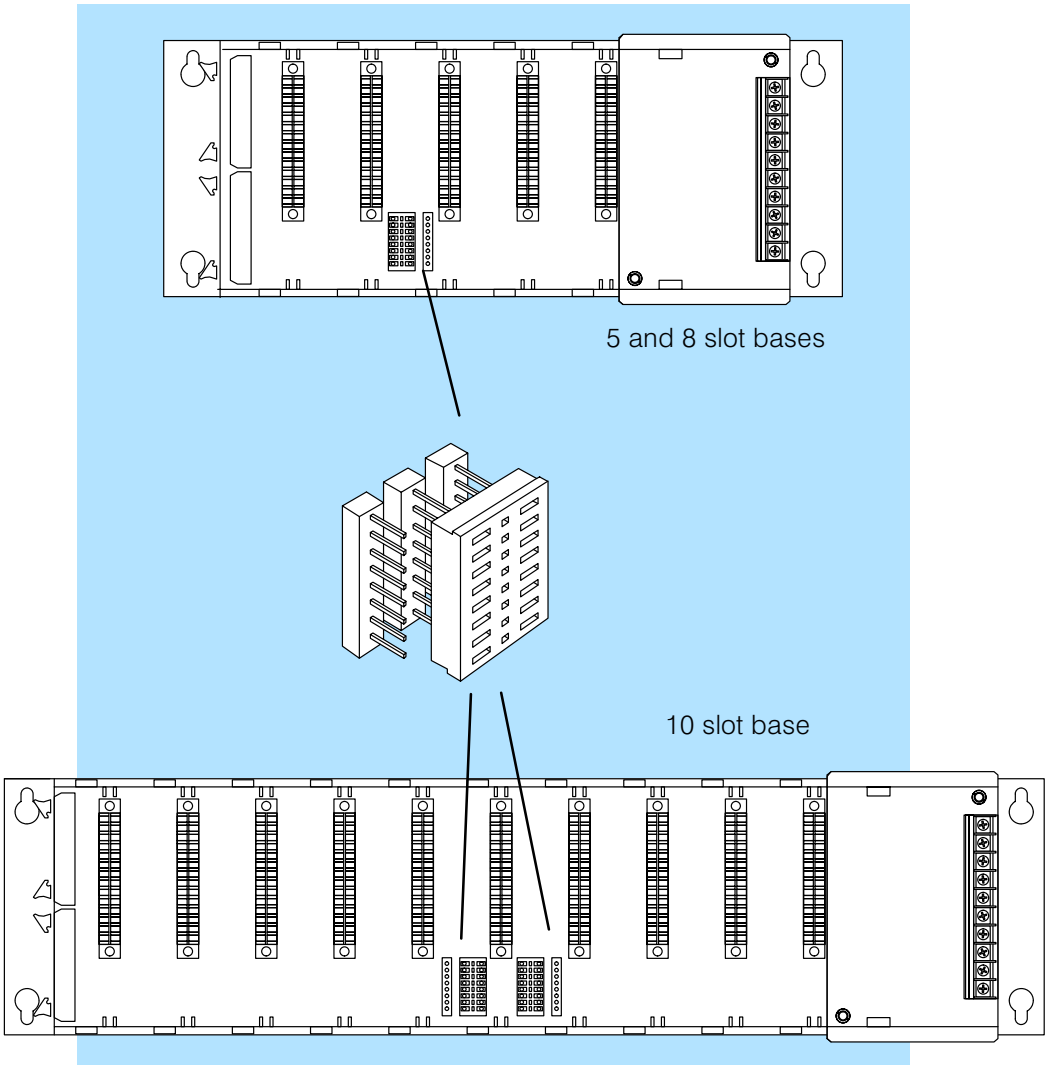


Note: Avoid placing the expansion cable in the same wiring tray as the I/O and power source wiring.

Setting the Base Switches

Jumper Switch

The 5, and 8 slot bases have a jumper switch between slot 3 and 4 used to set the base to local CPU base or expansion base. The 10 slot base has two jumpers, one is located between slots 4 and 5 and the other is located between slot 5 and 6. The second switch sets I/O addressing ranges for the DL330/340 CPUs. This switch should always be bridged to the right hand position for the DL350 CPU.



I/O Configurations with a 5 Slot Local CPU Base

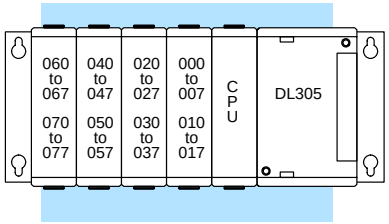
Switch settings

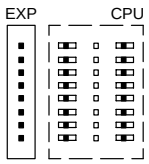
The 5 slot base has a jumper switch on the inside of the base between slots 3 and 4 which allows you to select:

Type of Base	Switch Position
Local CPU	right side bridged
First Expansion	left side bridged
Last Expansion	right side bridged

5 Slot Base

Total I/O:
8 pt. modules 32
16 pt. modules 64



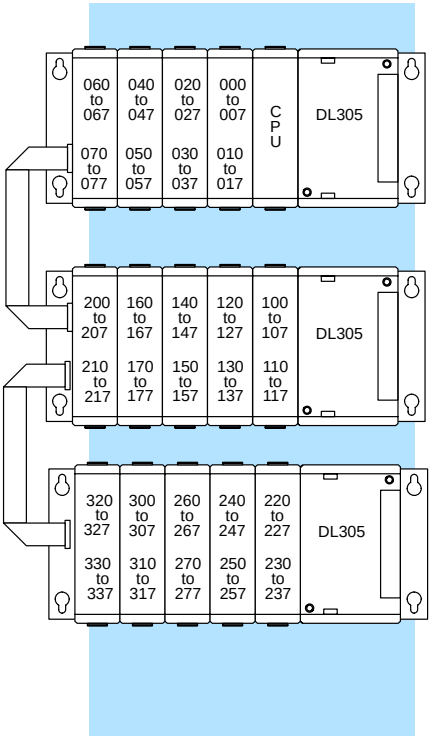


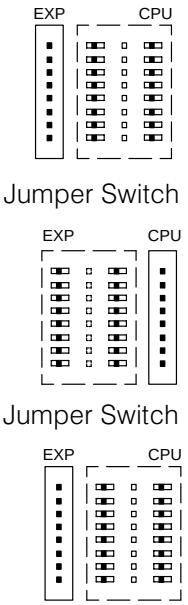
Jumper Switch

5 Slot Base and up to two 5 Slot Expansion Bases

Total I/O:
1 Expansion base
8 pt. modules – 72
16 pt. modules – 144

2 Expansion Bases
8 pt. modules – 112
16 pt modules – 224

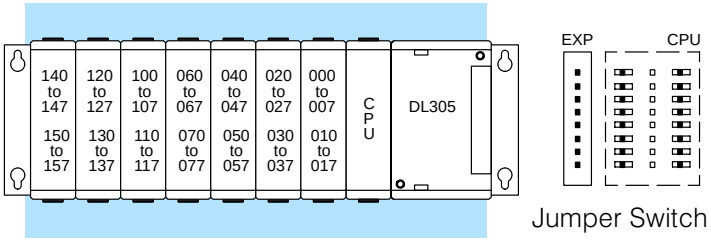




I/O Configurations with an 8 Slot Local CPU Base

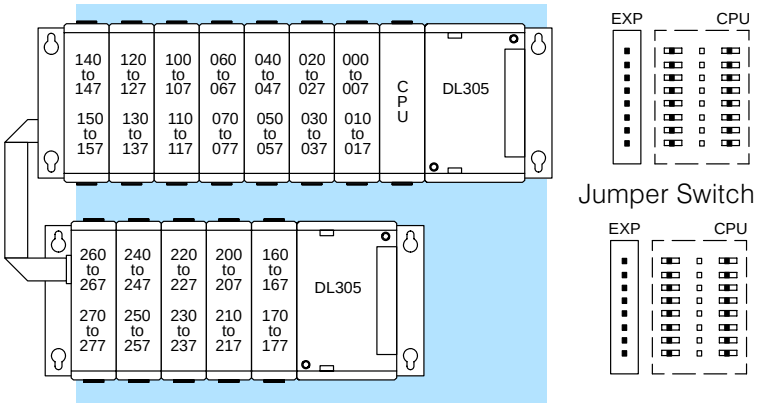
8 Slot Base

Total I/O:
8 pt. modules – 56
16 pt. modules – 112



8 Slot Base and 5 Slot Expansion Base

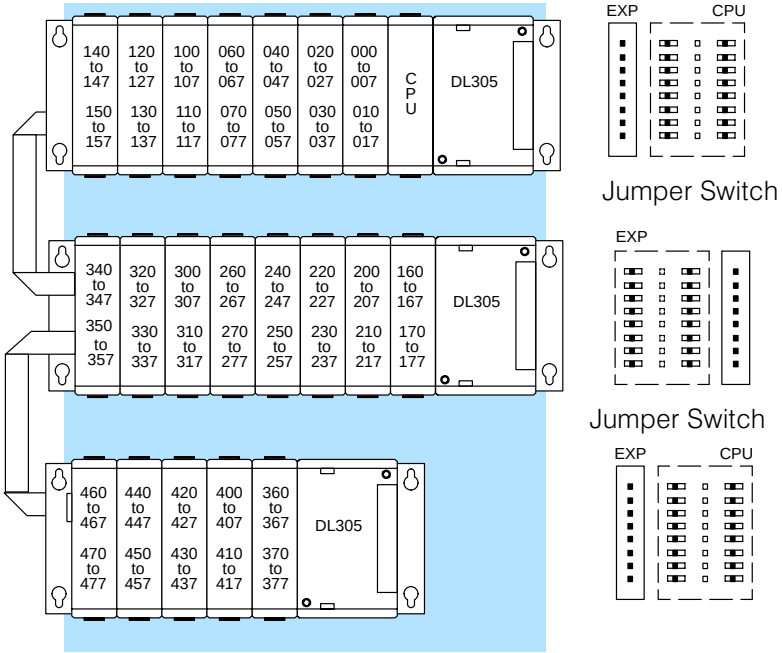
Total I/O:
8 pt modules – 96
16 pt modules – 192



8 Slot Base and One 8 slot and one 5 slot Expansion Bases

Total I/O:
1 Expansion Base
8 pt modules – 120
16 pt modules – 240

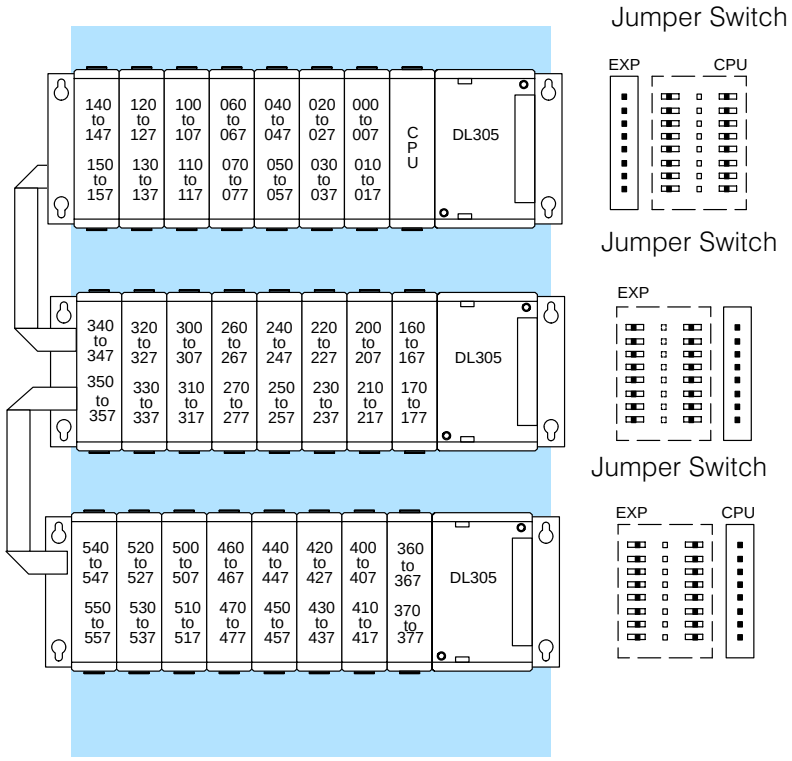
2 Expansion Bases
1 – 8 slot 1 – 5 slot
8 pt. modules – 160
16 pt. modules – 320



8 Slot Base and two 8 slot Expansion Bases

Total I/O:

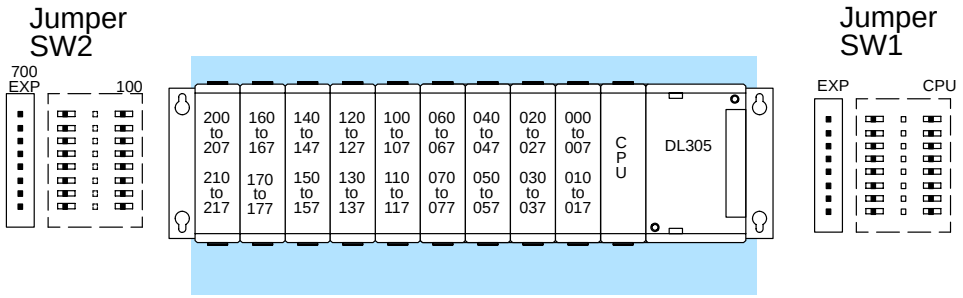
2 Expansion Bases
2 – 8 slot
8 pt. modules – 184
16 pt. modules – 368



I/O Configurations with a 10 Slot Local CPU Base

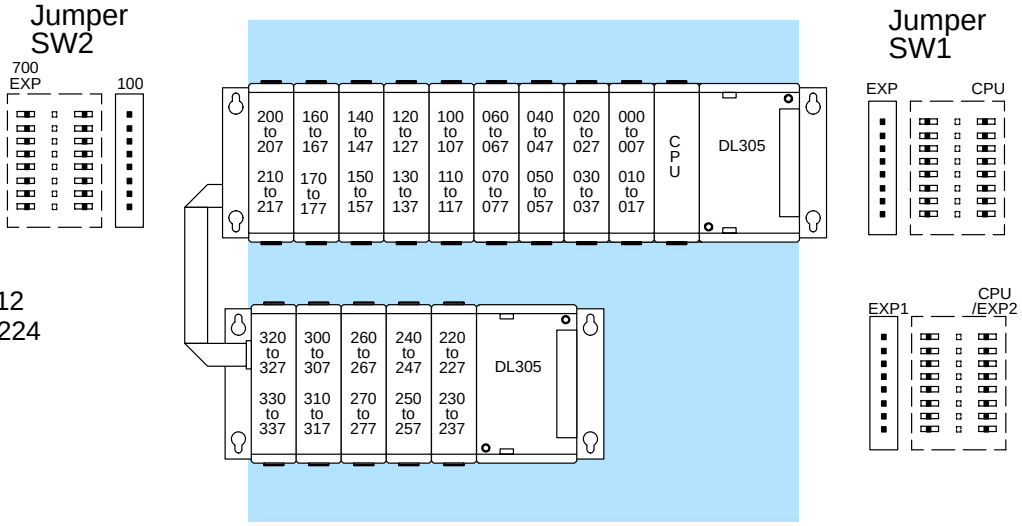
10 Slot Base

Total I/O:
8 pt. modules – 72
16 pt. modules – 144



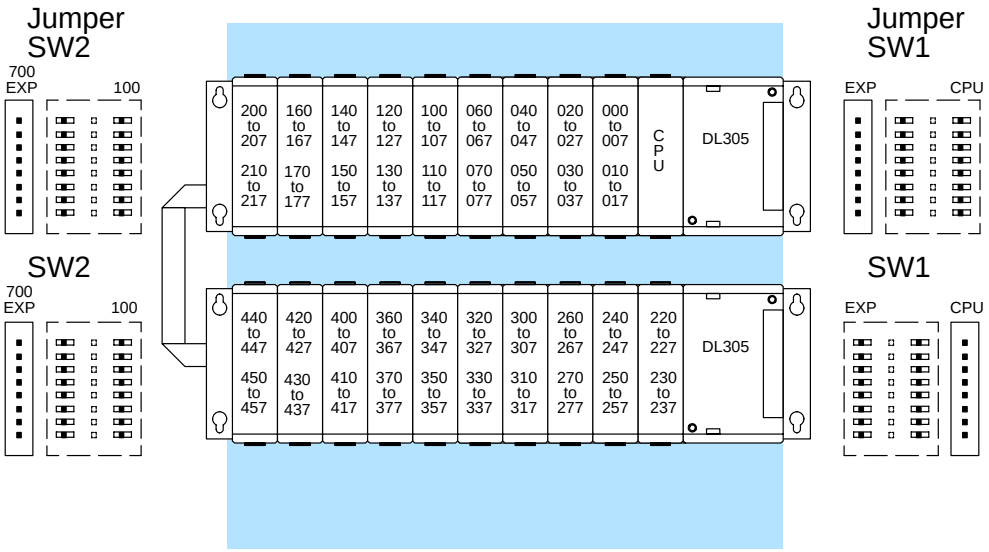
10 Slot Base and 5 Slot Expansion Base with 16 Point I/O

Total I/O:
8 pt. modules – 112
16 pt. modules – 224



10 Slot Base and 10 Slot Expansion Base with 16 Point I/O

Total I/O:
8 pt. modules – 152
16 pt. modules – 304



Remote I/O Expansion

How to Add Remote I/O Channels

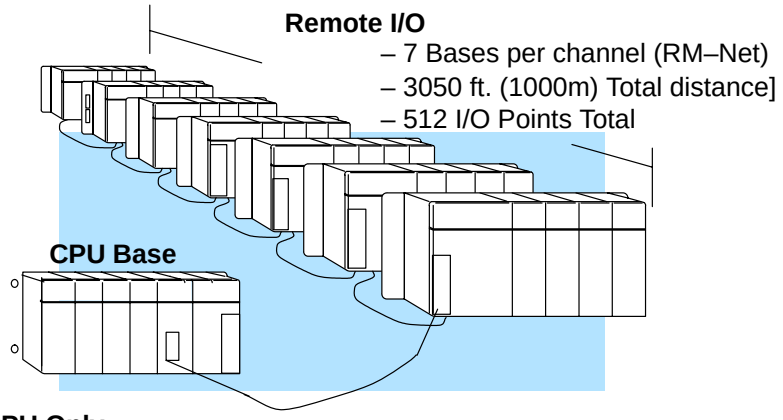
Remote I/O is useful for a system that has a sufficient number of sensors and other field devices located a relative long distance away (up to 1000 meters, or 3050 feet) from the more central location of the CPU. The DL350 supports a built–in Remote master, however the DL305 family does not have any Remote I/O modules. Therefore, you must use a DL205 or DL405 base for the slave channels. The methods of adding remote I/O are:

- **DL350 CPU**: The CPU's comm port 2 features a built-in Remote I/O channel.

	DL350
Maximum number of Remote Masters supported in the local CPU base (1 channel per Remote Master)	1
CPU built-in Remote I/O channels	1
Maximum I/O points supported by each channel	512
Maximum Remote I/O points supported	512
Maximum number of remote I/O bases per channel (RM–NET)	7

Remote I/O points map into different CPU memory locations, therefore it does not reduce the number of local I/O points. Refer to the DL205 Remote I/O manual for details on remote I/O configuration and numbering. Configuring the built-in remote I/O channel is described in the following section.

The following figure shows 1 CPU base with seven remote bases. The remote bases can be DL205 or DL405 bases.



DL350 CPU Only
RM–Net

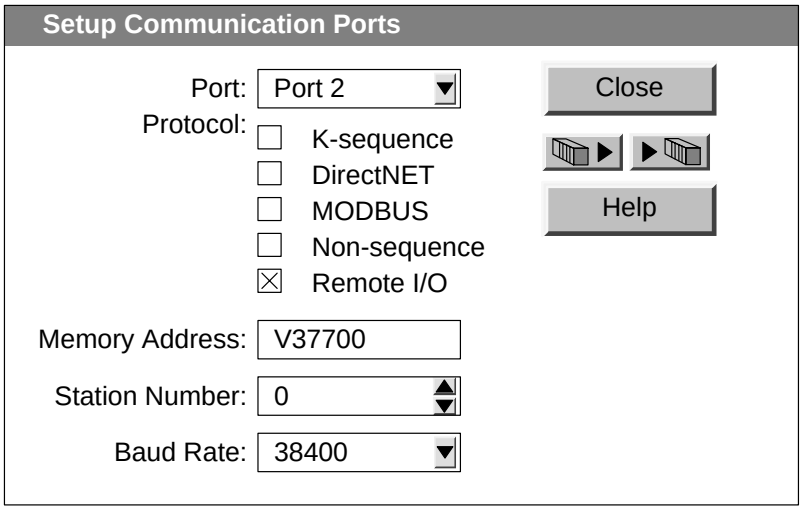
Configuring the CPU's Remote I/O Channel

This section describes how to configure the DL350's built-in remote I/O channel. Additional information is in the Remote I/O manual, D2–REMIO–M, which you will need in configuring the Remote slave units on the network.

The DL350 CPU's built-in remote I/O channel has the same capability as the DL250 and DL450 CPUs. It can communicate with up to seven remote bases containing a maximum of 512 I/O points, at a maximum distance of 1000 meters.

You may recall from the CPU specifications in Chapter 3 that the DL350's Port 2 is capable of several protocols. To configure the port using the Handheld Programmer, use AUX 56 and follow the prompts, making the same choices as indicated below on this page. To configure the 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 "Remote I/O" (called "M–NET" on the HPP), and then you'll see the dialog box shown below.



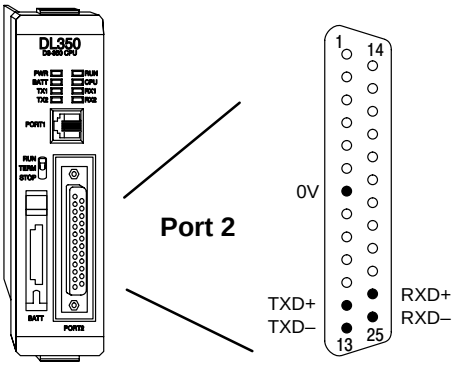
- **Memory Address:** Choose a V-memory address to use as the starting location of a Remote I/O configuration table (V37700 is the default). This table is separate and independent from the table for any Remote Master(s) in the system.
- **Station Number:** Choose "0" as the station number, which makes the DL350 the master. Stations numbers 1–7 are reserved for remote slaves.
- **Baud Rate:** The baud rates 19200 and 38400 baud are available. Choose 38400 initially as the remote I/O baud rate, and revert to 19200 baud if you experience data errors or noise problems on the link. Important: You must configure the baud rate on the Remote Slaves (via DIP switches) to match the baud rate selection for the CPU's Port 2.

Then click the button indicated to send the Port 2 configuration to the CPU, and click Close.

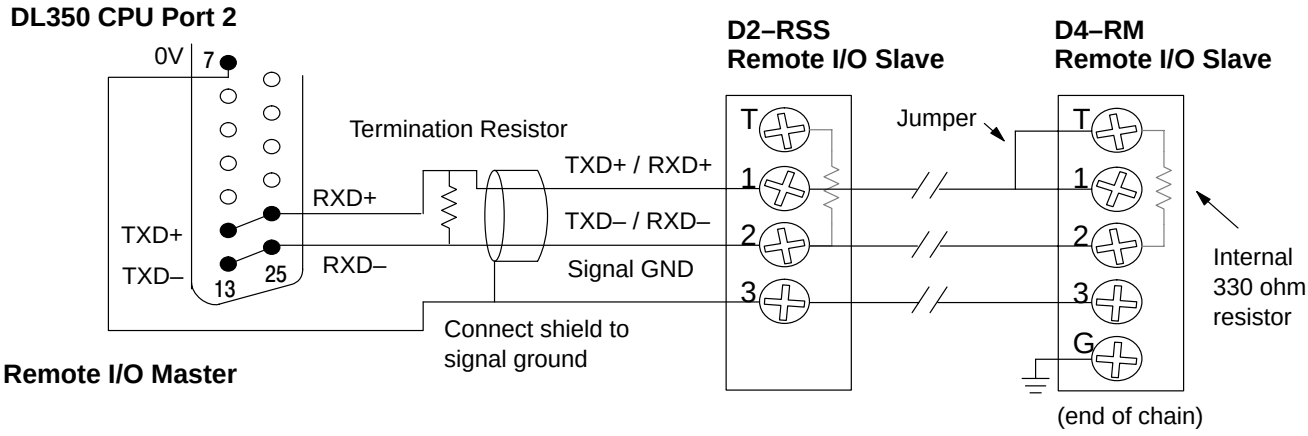
The next step is to make the connections between all devices on the Remote I/O link.

The location of the Port 2 on the DL350 is on the 25-pin connector , as pictured to the right.

- Pin 7 Signal GND
- Pin 12 TXD+
- Pin 13 TXD–
- Pin 24 RXD+
- Pin 25 RXD–



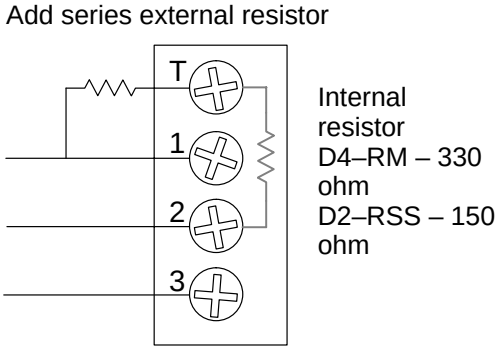
Now we are ready to discuss wiring the DL350 to the remote slaves on the remote base(s). The remote I/O link is a 3-wire, half-duplex type. Since Port 2 of the DL350 CPU is a 5-wire port, we must jumper its transmit and receive lines together as shown below (converts it to 3-wire, half-duplex).



The twisted/shielded pair connects to the DL350 Port 2 as shown. Be sure to connect the cable shield wire to the signal ground connection. A termination resistor must be added externally to the CPU, as close as possible to the connector pins. Its purpose is to minimize electrical reflections that occur over long cables. Be sure to add the jumper at the last slave to connect the required internal termination resistor.

Ideally, the two termination resistors at the cables opposite ends and the cable's rated impedance will all three match. For cable impedances greater than 330 ohms, add a series resistor at the last slave as shown to the right. If less than 330 ohms, parallel a matching resistance across the slave's pins 1 and 2 instead.

Remember to size the termination resistor at Port 2 to match the cables rated impedance. *The resistance values should be between 100 and 500 ohms.*



Configure Remote I/O Slaves

After configuring the DL350 CPU's Port 2 and wiring it to the remote slave(s), use the following checklist to complete the configuration of the remote slaves. Full instructions for these steps are in the Remote I/O manual.

- Set the baud rate to match CPU's Port 2 setting.
- Select a station address for each slave, from 1 to 7. Each device on the remote link *must* have a unique station address. There can be only one master (address 0) on the remote link.

Configuring the Remote I/O Table

The beginning of the configuration table for the built-in remote I/O channel is the memory address we selected in the Port 2 setup.

The table consists of blocks of four words which correspond to each slave in the system, as shown to the right. The first four table locations are reserved.

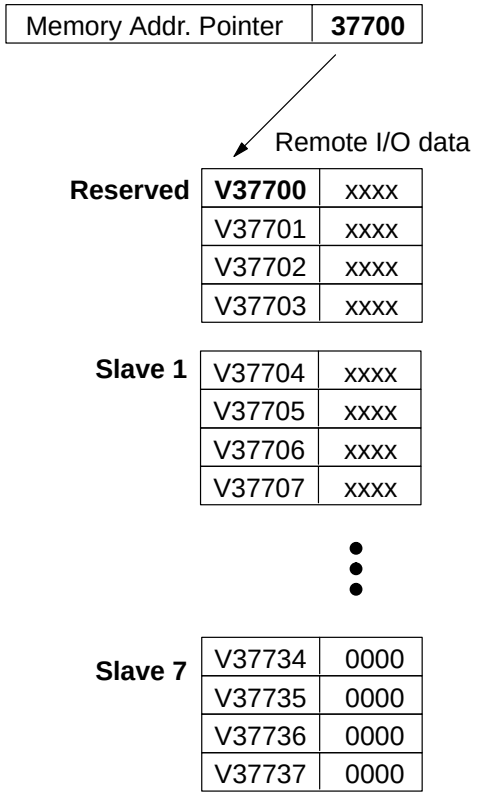
The CPU reads data from the table after powerup, interpreting the four data words in each block with these meanings:

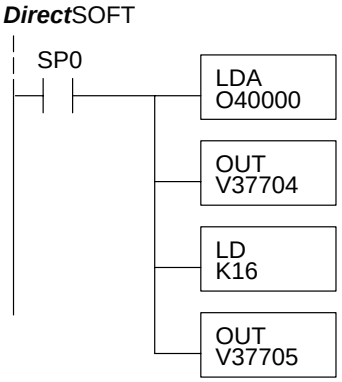
1. Starting address of slave's input data
2. Number of slave's input points
3. Starting address of outputs in slave
4. Number of slave's output points

The table is 32 words long. If your system has fewer than seven remote slave bases, then the remainder of the table must be filled with zeros. For example, a 3–slave system will have a remote configuration table containing 4 reserved words,12 words of data and 16 words of "0000".

A portion of the ladder program must configure this table (only once) at powerup. Use the LDA instruction as shown to the right, to load an address to place in the table. Use the regular LD constant to load the number of the slave's input or output points.

The following page gives a short program example for one slave.

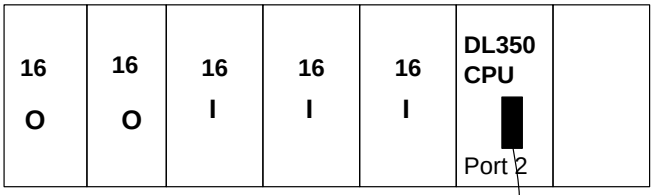




Consider the simple system featuring Remote I/O shown below. The DL350's built-in Remote I/O channel connects to one slave base, which we will assign a station address=1. The baud rates on the master and slave will be 38400 kB.

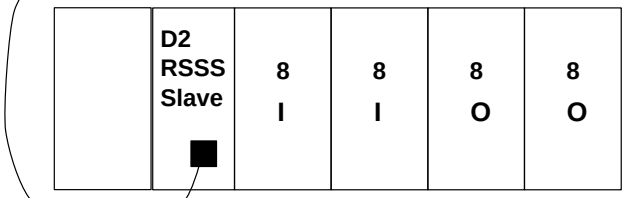
We can map the remote I/O points as any type of I/O point, simply by choosing the appropriate range of V-memory. Since we have plenty of standard I/O addresses available (X and Y), we will have the remote I/O points start at the next X and Y addresses after the main base points (X60 and Y40, respectively).

Main Base with CPU as Master



Y20-Y37 V40501 | Y0-Y17 V40500 | X40-X57 V40402 | X20-X37 V40401 | X0-X17 V40400

Remote Slave



X60-X67 V40403 | X70-X77 V40404 | Y40-Y47 V40502 | Y50-Y57 V40503

Remote Slave Worksheet

Remote Base Address 1 **(Choose 1–7)**

Slot Number	Module Name	INPUT		OUTPUT	
		Input Addr.	No. Inputs	Output Addr.	No.Outputs
0	08ND3S	X060	8		
1	08ND3S	X070	8		
2	08TD1			Y040	8
3	08TD1			Y050	8
4					
5					
6					
7					

Input Bit Start Address: X060 **V-Memory Address:V** 40403

Total Input Points 16

Output Bit Start Address: Y040 **V-Memory Address:V** 40502

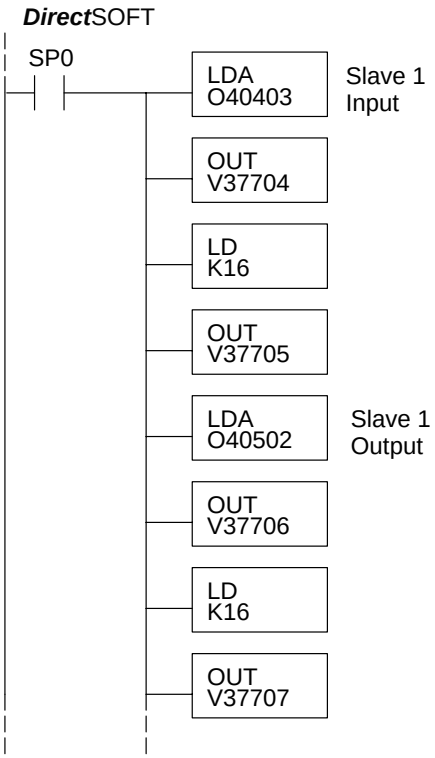
Total Output Points 16

Remote I/O Setup Program

Using the Remote Slave Worksheet shown above can help organize our system data in preparation for writing our ladder program (a blank full-page copy of this worksheet is in the Remote I/O Manual). The four key parameters we need to place in our Remote I/O configuration table is in the lower right corner of the worksheet. You can determine the address values by using the memory map given at the end of Chapter 3, CPU Specifications and Operation.

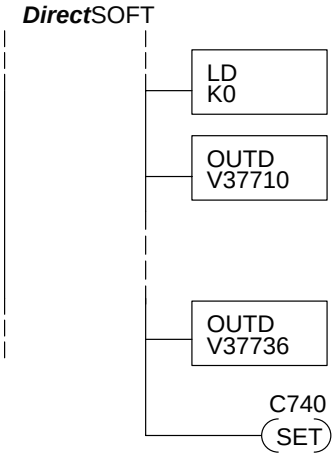
The program segment required to transfer our worksheet results to the Remote I/O configuration table is shown to the right. Remember to use the LDA or LD instructions appropriately.

The next page covers the remainder of the required program to get this remote I/O link up and running.



When configuring a Remote I/O channel for fewer than 7 slaves, we must fill the remainder of the table with zeros. This is necessary because the CPU will try to interpret any non-zero number as slave information.

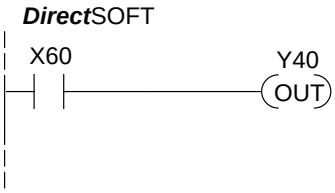
We continue our setup program from the previous page by adding a segment which fills the remainder of the table with zeros. The example to the right fills zeros for slave numbers 2–7, which do not exist in our example system.



On the last rung in the example program above, we set a special relay contact C740. This particular contact indicates to the CPU the ladder program has finished specifying a remote I/O system. At that moment the CPU begins remote I/O communications. Be sure to include this contact after any Remote I/O setup program.

Remote I/O Test Program

Now we can verify the remote I/O link and setup program operation. A simple quick check can be done with one rung of ladder, shown to the right. It connects the first input of the remote base with the first output. After placing the PLC in RUN mode, we can go to the remote base and activate its first input. Then its first output should turn on.

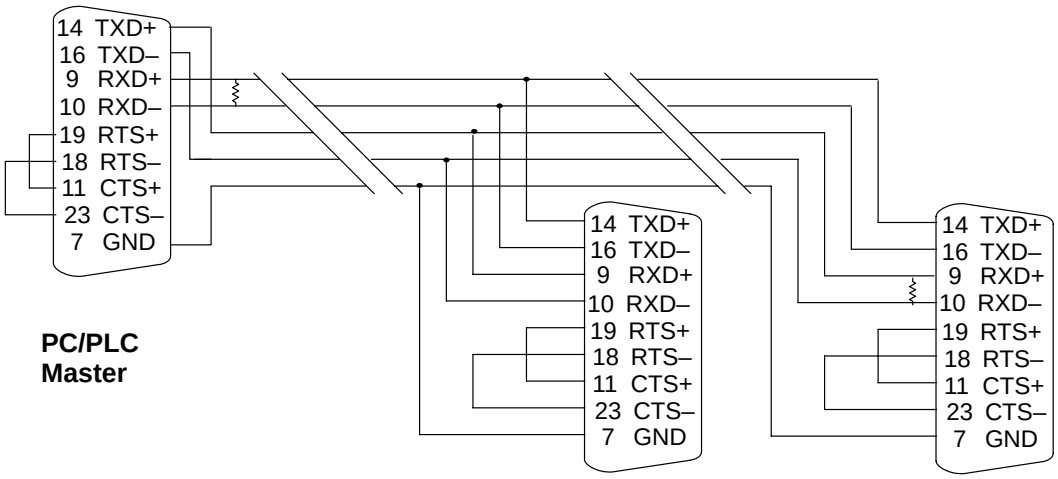


Network Connections to MODBUS® and *Direct*NET

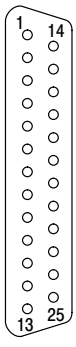
Configuring the CPU's Comm Port

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 DL305 PLC system directly to MODBUS networks using the RTU protocol, or to other devices on a ***Direct***NET network. MODBUS hosts system 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.

You will need to determine whether the network connection is a 3-wire RS–232 type, or a 5-wire RS–422 type. Normally, the RS–232 signals are used for shorter distances (15 meters max), for communications between two devices. RS–422 signals are for longer distances (1000 meters max.), and for multi-drop networks (from 2 to 247 devices). Use termination resistors at both ends of RS–422 network wiring, matching the impedance rating of the cable (between 100 and 500 ohms). Resistors should be insatlled close to the end of the cable at the master and last slave connections.



The recommended cable for RS422 is Beldon 9855 or equivalent.



25-pin Female D Connector

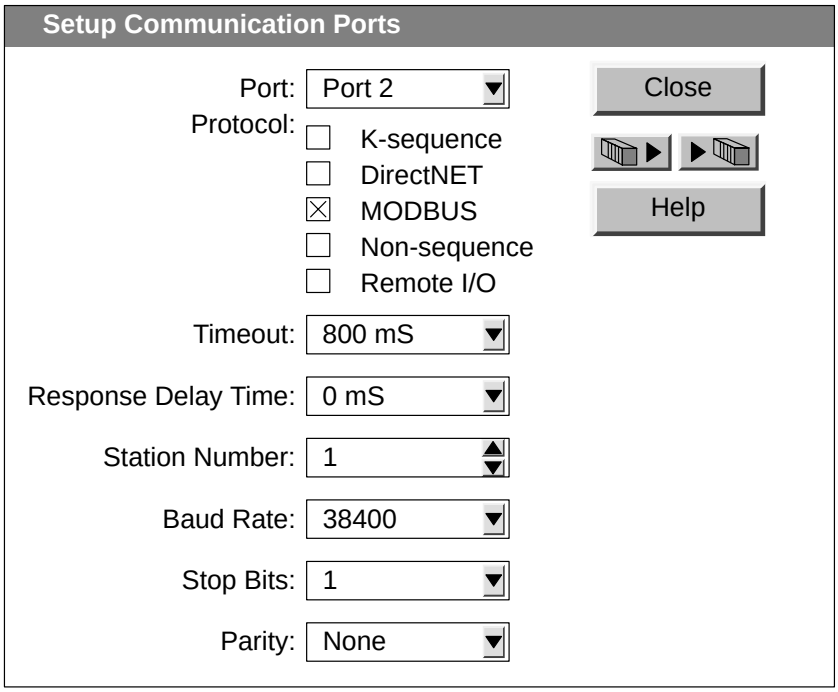
Port 2 Pin Descriptions (DL350 CPU)		
1	not used	
2	TXD	Transmit Data (RS232C)
3	RXD	Receive Data (RS232C)
4	RTS	Ready to Send (RS–232C)
5	CTS	Clear to Send (RS–232C)
6	not used	
7	0V	Power (–) connection (GND)
8	0V	Power (–) connection (GND)
9	RXD +	Receive Data + (RS–422)
10	RXD –	Receive Data (RS–422)
11	CTS +	Clear to Send + (RS422)
12	TXD +	Transmit Data + (REMIO)
13	TXD –	Transmit Data – (REMIO)

Port 2 Pin Descriptions (Cont'd)		
14	TXD +	Transmit Data + (RS–422
15	not used	
16	TXD –	Transmit Data – (RS–422)
17	not used	
18	RTS –	Request to Send – (RS–422)
19	RTS +	Request to Send – (RS–422)
20	not used	
21	not used	
22	not used	
23	CTS –	Clear to Send – (RS–422)
24	RXD +	Receive Data + (REMIO)
25	RXD –	Receive Data – (REMIO)

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:** Click the check box to the left of "MODBUS" (use AUX 56 on the HPP, and select "MBUS"), and then you'll see the dialog box below.



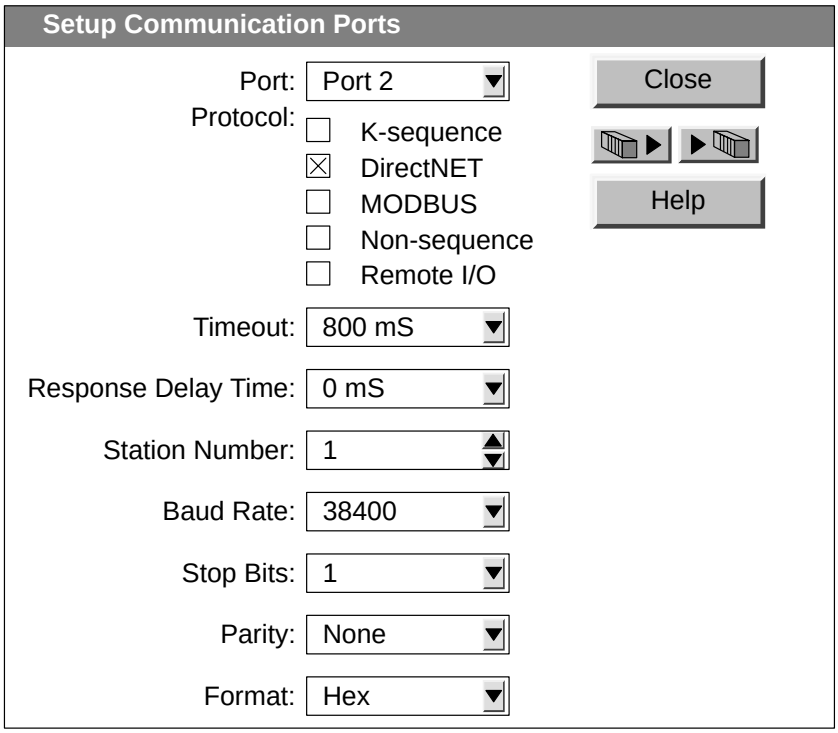
- **Timeout:** amount of time the port will wait after it sends a message to get a response before logging an error.
- **Response Delay Time:** The amount of time between raising the RTS line and sending the data. This is for devices that do not use RTS/CTS handshaking. The RTS and CTS lines must be bridged together for the CPU to send any data.
- **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 DL350 network instructions used in Master mode will access only slaves 1 to 90. Each slave must have a unique number. At powerup, the port is automatically a slave, unless and until the DL350 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, 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 for use in the protocol.
- **Parity:** Choose none, even, or odd parity for error checking.

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:** Click the check box to the left of “DirectNET” (use AUX 56 on the HPP, then select “DNET”), and then you’ll see the dialog box below.



- **Timeout:** amount of time the port will wait after it sends a message to get a response before logging an error.
- **Response Delay Time:** The amount of time between raising the RTS line and sending the data. This is for devices that do not use RTS/CTS handshaking. The RTS and CTS lines must be bridged together for the CPU to send any data.
- **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 DL350 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, 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.
- **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.

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 (DL350). A MODBUS host must use the MODBUS RTU protocol to communicate with the DL350 as a slave. The host software must send a MODBUS function code and MODBUS address to specify a PLC memory location the DL350 comprehends. The ***Direct***NET host uses normal I/O addresses to access the applicable DL305 CPU and system. No CPU ladder logic is required to support either MODBUS slave or ***Direct***NET slave 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 DL350 supports the MODBUS function codes described below.

MODBUS Function Code	Function	DL305 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

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.

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.

DL350 Memory Type	QTY (Dec.)	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 – 2560	Input
Special Relays (SP)	512	SP0 – SP777	3072 – 3584	Input
Outputs (Y)	512	Y0 – Y777	2048 – 2560	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 – 6271	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)	3072 4096	V1400 – V7377 V10000 – V17777	768 – 3839 4096 – 8191	Holding Register
V Memory, system (V)	256	V7400 – V7777	3480 – 3735	Holding Register

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

Example 1: V2100

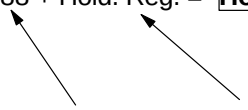
Find the MODBUS address for User V location V2100.

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

PLC Address (Dec.) + Data Type

V2100 = 1088 decimal

1088 + Hold. Reg. = **Holding Reg. 1088**



V Memory, user data (V)	3072 12288	V1400 – V7377 V10000–V37777	768 – 3839 4096 – 16383	Holding Register

Example 2: Y20

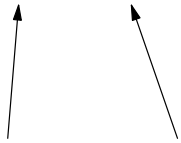
Find the MODBUS address for output Y20.

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.

PLC Addr. (Dec) + Start Addr. + Data Type

Y20 = 16 decimal

16 + 2048 + Coil = **Coil 2064**



Outputs (Y)	1024	Y0 – Y1777	2048 – 3071	Coil

Example 3: T10 Current Value

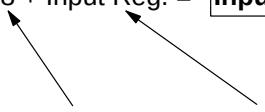
Find the MODBUS address to obtain the current value from Timer T10.

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

PLC Address (Dec.) + Data Type

T10 = 8 decimal

8 + Input Reg. = **Input Reg. 8**



Timer Current Values (V)	256	V0 – V377	0 – 255	Input Register

Example 4: C54

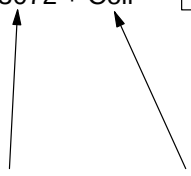
Find the MODBUS address for Control Relay C54.

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.

PLC Addr. (Dec) + Start Addr. +Data Type

C54 = 44 decimal

44 + 3072 + Coil = **Coil 3116**



Control Relays (CR)	2048	C0 – C3777	3072 – 5119	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, 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.

DL350 Memory Type	QTY (Dec.)	PLC Range (Octal)	MODBUS Address Range (Decimal)	484 Mode Address	584/984 Mode Address	MODBUS Data Type
For Discrete Data Types ... Convert PLC Addr. to Dec. +			**Start of Range**	**+ Appropriate Mode Address**		
Inputs (X)	512	X0 – X777	2048 – 2560	1001	10001	Input
Special Relays (SP)	512	SP0 – SP777	3072 – 3584	1001	10001	Input
Outputs (Y)	512	Y0 – Y777	2048 – 2560	1	1	Coil
Control Relays (CR)	1024	C0 – C3777	3072 – 4095	1	1	Coil
Timer Contacts (T)	256	T0 – T377	6144 – 6399	1	1	Coil
Counter Contacts (CT)	128	CT0 – CT177	6400 – 6527	1	1	Coil
Stage Status Bits (S)	1024	S0 – S1777	5120 – 6143	1	1	Coil
For Word Data Types .... Convert PLC Addr. to Dec.			**+**	**Appropriate Mode Address**		
Timer Current Values (V)	256	V0 – V377	0 – 255	3001	30001	Input Reg.
Counter Current Values (V)	128	V1000 – V1177	512 – 639	3001	30001	Input Reg
V Memory, user data (V)	3072 4096	V1400 – V7377 V10000 – V17777	768 – 3839 4096 – 8192	4001	40001	Hold Reg.
V Memory, system (V)	256	V7400 – V7777	3840 – 3735	4001	40001	Hold Reg.

The following examples show how to generate the MODBUS addresses for hosts which require this format.

Example 1: V2100 584/984 Mode

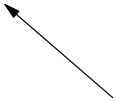
Find the MODBUS address for User V location V2100.

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**



V Memory, system (V)	320	V700 – V777 V7400 – V7777	448 – 768 3840 – 3735	4001	40001	Hold Reg.

Example 2: Y20 584/984 Mode

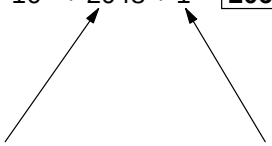
Find the MODBUS address for output Y20.

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 Addr. + Mode

Y20 = 16 decimal

16 + 2048 + 1 = **2065**



Outputs (Y)	1024	Y0 – Y1777	2048 – 3071	1	1	Coil

Example 3: T10 Current Value 484 Mode

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

PLC Address (Dec.) + Mode Address

T10 = 8 decimal

8 + 3001 = **3009**

Timer Current Values (V)	256	V0 – V377	0 – 255	3001	30001	Input Reg.

Example 4: C54 584/984 Mode

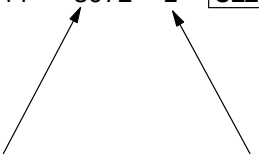
Find the MODBUS address for Control Relay C54.

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

PLC Addr. (Dec) + Start Address + Mode

C54 = 44 decimal

44 + 3072 + 1 = **3117**



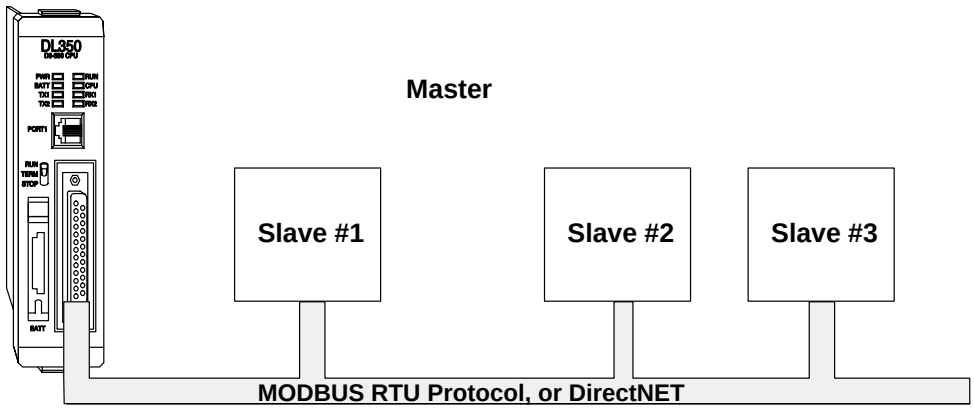
Control Relays (CR)	2048	C0 – C3777	3072 – 5119	1	1	Coil

Determining the *Direct*NET Address

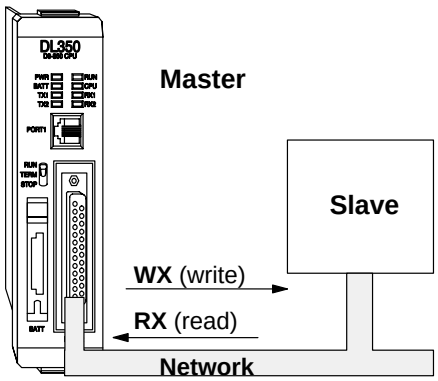
Addressing the memory types for ***Direct***NET slaves is very easy. Use the ordinary native address of the slave device itself. To access a slave PLC's memory address V2000 via ***Direct***NET, for example, the network master will request V2000 from the slave.

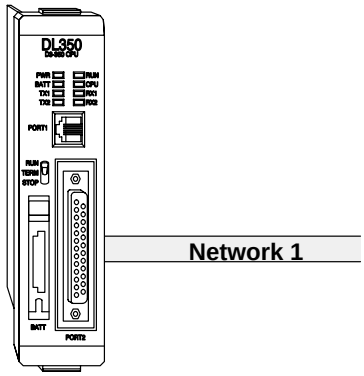
Network Master Operation

This section describes how the DL350 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 DL350 CPU as the master station, you use simple RLL instructions 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.

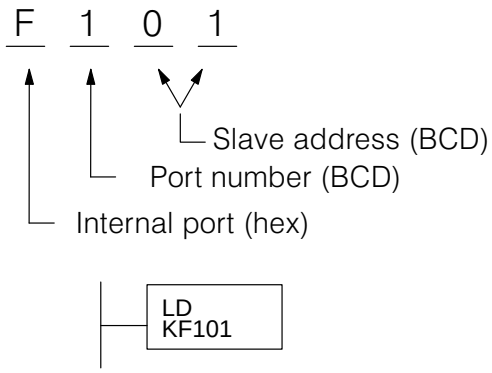




The following step-by-step procedure will provide you the information necessary to set up your ladder program to receive data from a network slave.

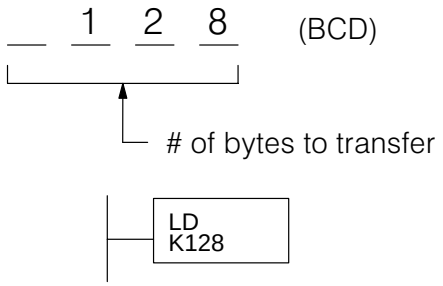
Step 1: Identify Master Port # and Slave #

The first Load (LD) instruction identifies the communications port number on the network master (DL350) and the address of the slave station. This instruction can address up to 90 MODBUS slaves, or 90 ***Direct***NET slaves. The format of the word is shown to the right. The "F" in the upper nibble tells the CPU the port is internal to the CPU (and not in a slot in the base). The second nibble indicates the port number, 1. This is the logical port number (0 for top port and 1 for the bottom). The lower byte contains the slave address number in BCD (01 to 90).



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 DL305 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.

DL 205 / 305 / 405 Memory	Bits per unit	Bytes
V memory	16	2
T / C current value	16	2
Inputs (X, SP)	8	1
Outputs (Y, C, Stage, T/C bits)	8	1
Scratch Pad Memory	8	1
Diagnostic Status	8	1

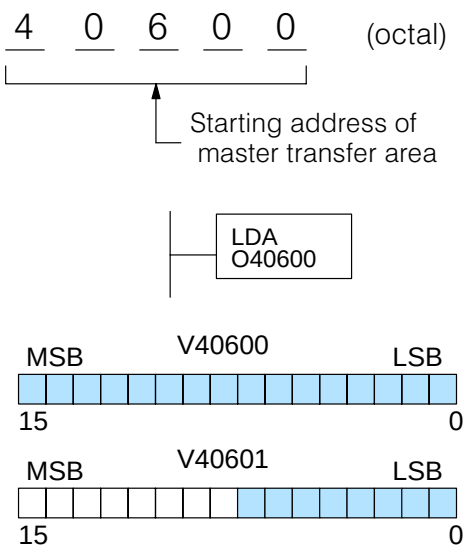
DL305C (DL330/340 CPUs) Memory	Bits per unit	Bytes
Data registers	8	1
T / C accumulator	16	2
I/O, internal relays, shift register bits, T/C bits, stage bits	8	1
Scratch Pad Memory	8	2
Diagnostic Status(5 word R/W)	16	10

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 DL350 CPU sends the number of bytes previously specified from its memory area beginning at the LDA address specified.

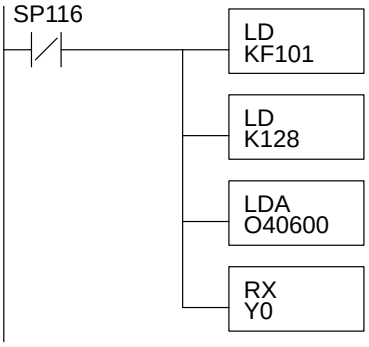
For an RX instruction, the DL350 CPU reads the number of bytes previously specified from the slave, placing the received data into its memory area beginning at the LDA address specified.



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.

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.



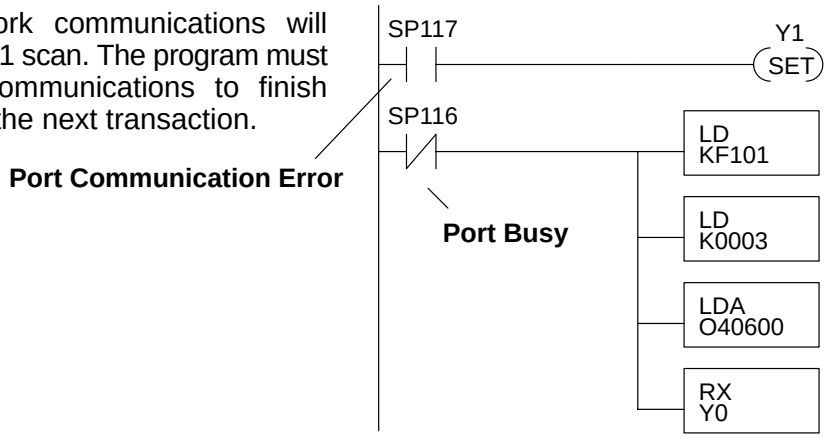
- ***Direct***NET slaves – specify the same address in the WX and RX instruction as the slave's native I/O address
- MODBUS DL405, DL305 (DL350 CPU), or DL205 slaves – specify the same address in the WX and RX instruction as the slave's native I/O address
- MODBUS 305C (DL330/340 CPUs) slaves – use the following table to convert DL305 addresses to MODBUS addresses

DL305C (DL330/340 CPUs) Series CPU Memory Type–to–MODBUS Cross Reference

PLC Memory type	PLC base address	MODBUS base addr.	PLC Memory Type	PLC base address	MODBUS base addr.
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.



The port 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.

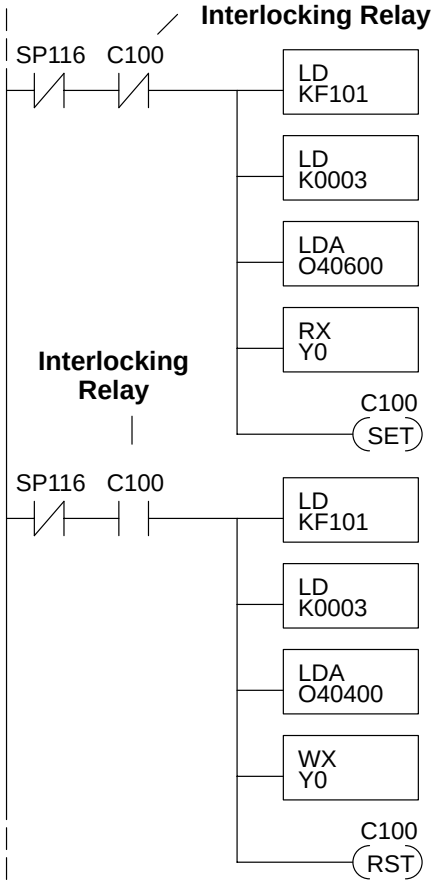
The “Port Communication Error” bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an RX or WX instruction is executed.

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, C0 is set. When the port has finished the communication task, the second routine is executed and C0 is reset.

If you’re using RLL*PLUS* Stage Programing, 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.



5

Standard RLL Instructions

In This Chapter. . . .

Introduction

The DL350 CPU offers a wide variety of instructions to perform many different types of operations. This chapter shows you how to use these individual instructions. There are two ways to quickly find the instruction you need.

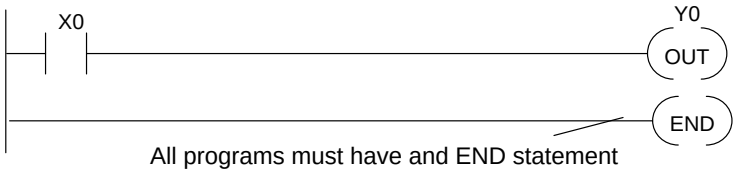
- If you know the instruction category (Boolean, Comparative Boolean, etc.) use the header at the top of the page to find the pages that discuss the instructions in that category.
- If you know the individual instruction name, use the following table to find the page that discusses the instruction.

Using Boolean Instructions

Do you question why many PLC manufacturers quote the scan time for a 1K boolean program? It is because most all programs utilize many boolean instructions. These are typically very simple instructions designed to join input and output contacts in various series and parallel combinations. Since the ***Direct***SOFT™ package allows the use of graphic symbols to build the program, you don't absolutely *have* to know the mnemonics of the instructions. However, it may helpful at some point, especially if you ever have to troubleshoot the program with a Handheld Programmer.

END Statement

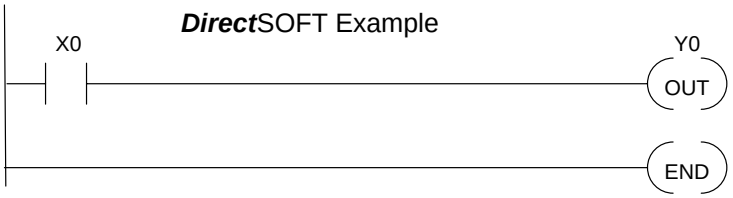
All programs require an END statement as the last instruction. This tells the CPU it is the end of the program. Normally, any instructions placed after the END statement will not be executed. There are exceptions to this such as interrupt routines, etc. The instruction set at the end of this chapter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



All programs must have and END statement

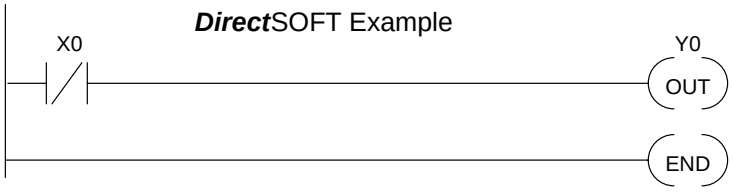
Simple Rungs

You will use a contact to start rungs that contain both contacts and coils. The boolean instruction, Store or, STR instruction performs this function. 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.



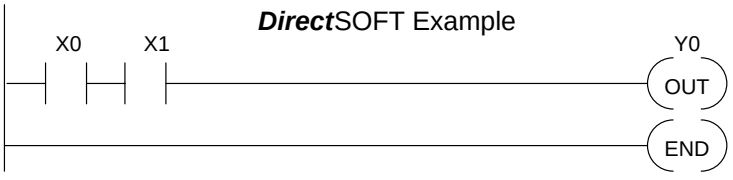
Normally Closed Contact

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.



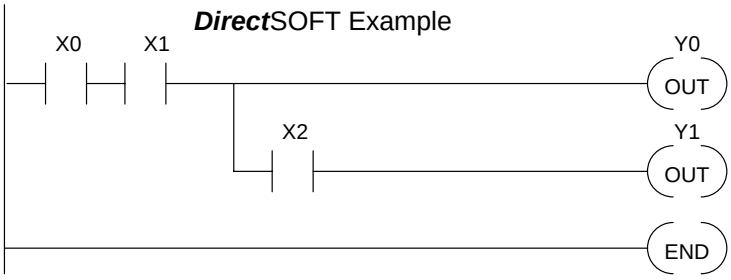
Contacts in Series

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 are STR X0, AND X1, followed by OUT Y0.



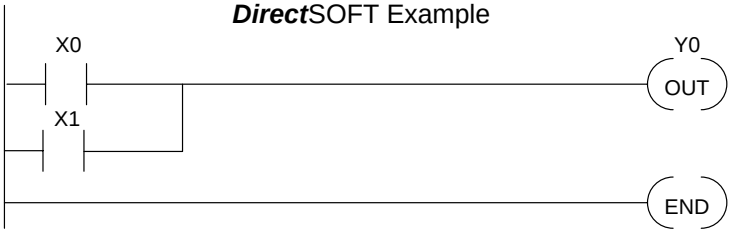
Midline Outputs

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.



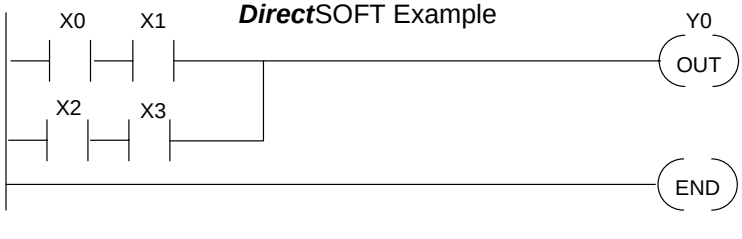
Parallel Elements

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.



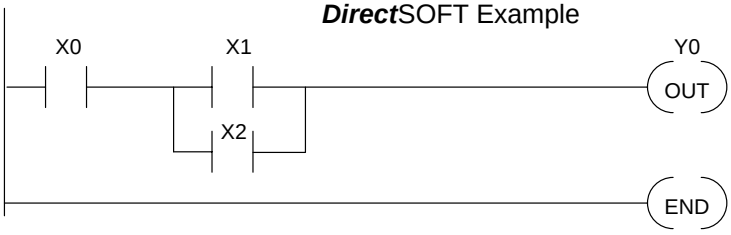
Joining Series Branches in Parallel

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.



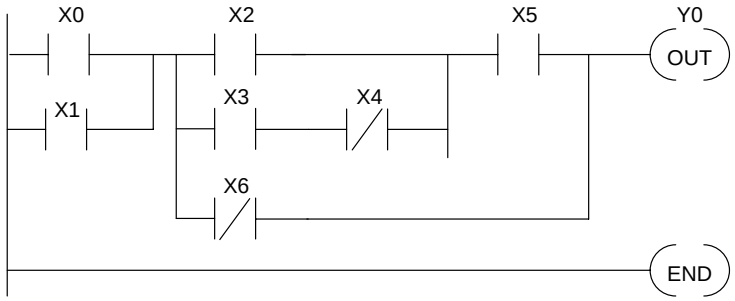
Joining Parallel Branches in Series

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.



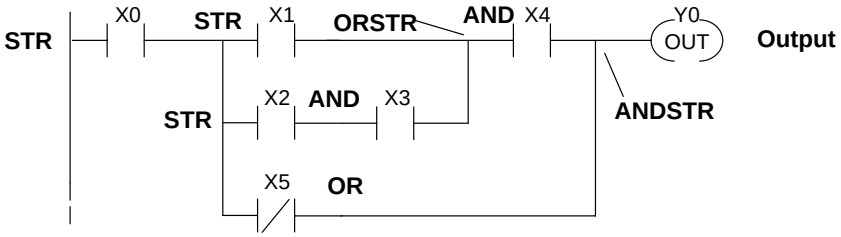
Combination Networks

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



Boolean Stack

There are limits to how many elements you can include in a rung. This is because the DL350 CPU uses 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 you enter a STR instruction, the instruction is placed on the top of the boolean stack. Any other STR instructions on the boolean stack are pushed down a level. The ANDSTR, and ORSTR instructions combine levels of the boolean stack when they are encountered. Since the boolean stack is only eight levels, an error will occur if the CPU encounters a rung that uses more than the eight levels of stack.



STR X0

1	STR X0
2	
3	
4	
5	
6	
7	
8	

STR X1

1	STR X1
2	STR X0
3	
4	
5	
6	
7	
8	

STR X2

1	STR X2
2	STR X1
3	STR X0
4	
5	
6	
7	
8	

AND X3

1	X2 AND X3
2	STR X1
3	STR X0
4	
5	
6	
7	
8	

ORSTR

1	X1 OR (X2 AND X3)
2	STR X0
3	
.	
8	

AND X4

1	X4 AND [X1 OR (X2 AND X3)]
2	STR X0
3	
.	
8	

ORNOT X5

1	NOT X5 OR X4 AND [X1 OR (X2 AND X3)]
2	STR X0
3	
.	
8	

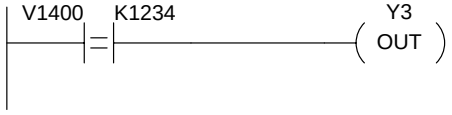
ANDSTR

1	X0 AND (NOT X5 OR X4) AND [X1 OR (X2 AND X3)]
2	
3	
:	
8	

Comparative Boolean

The DL350 CPU provides Comparative Boolean instructions that allow you to quickly and easily compare two numbers. The Comparative Boolean provides evaluation of two 4-digit values using boolean contacts. The valid evaluations are: equal to, not equal to, equal to or greater than, and less than.

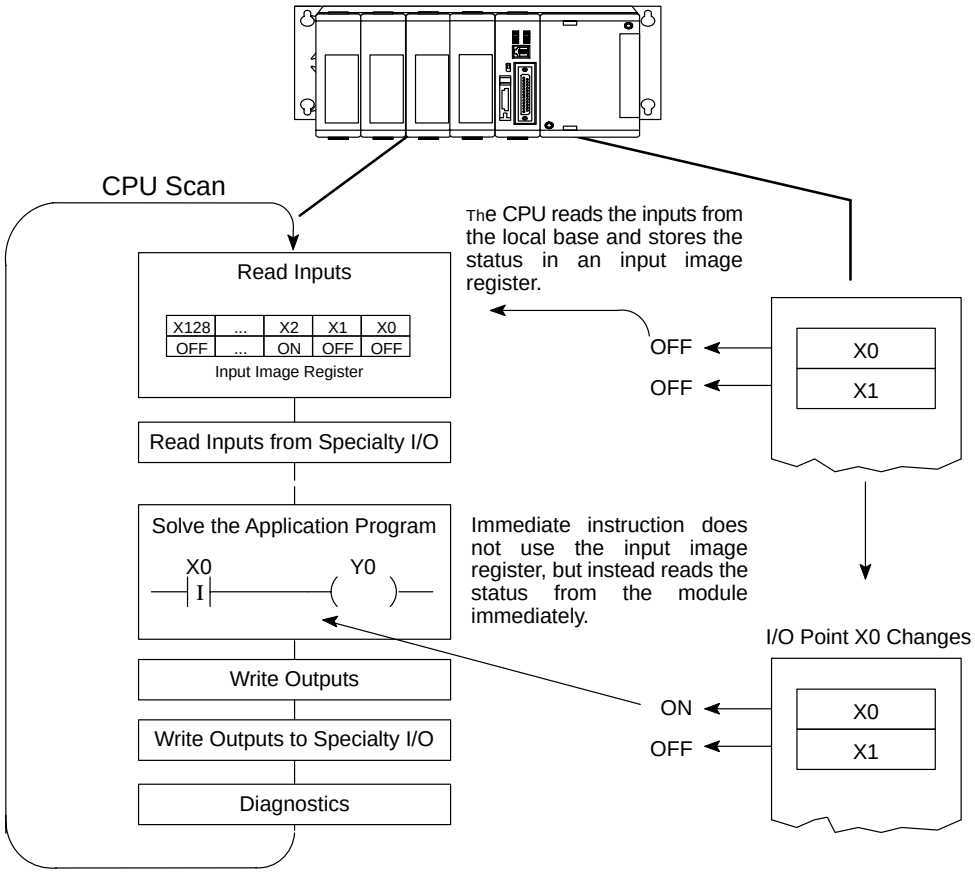
In the following example when the value in Vmemory location V1400 is equal to the constant 1234, Y3 will energize.



Immediate Boolean

The DL350 CPU usually can complete an operation cycle in milliseconds. However, in some applications you may not be able to wait a few milliseconds until the next I/O update occurs. The DL350 CPU offers 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. This is normally performed 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 module. This function is not normally performed until the read inputs or the write outputs portion of the CPU cycle.

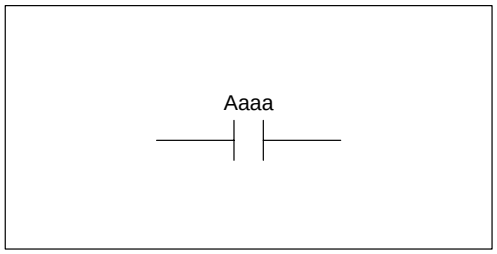
NOTE: Even though the immediate input instruction reads the most current status from the module, it only uses the results to solve that one instruction. It does not use the new status to update the image register. Therefore, any regular instructions that follow will still use the image register values.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



Boolean Instructions

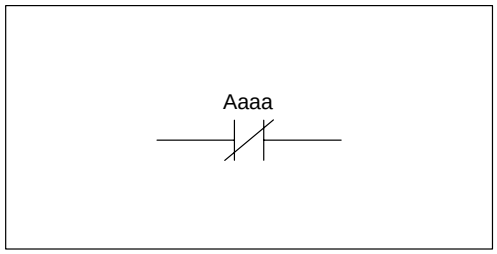
Store (STR)

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)

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		DL350 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	0–0777

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

DirectSOFT

Handheld Programmer Keystrokes

STR	→	1	ENT
OUT	→	2	ENT

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

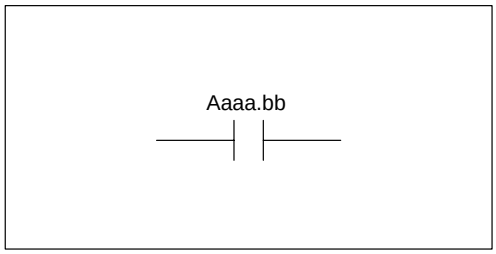
DirectSOFT

Handheld Programmer Keystrokes

STRN	→	1	ENT
OUT	→	2	ENT

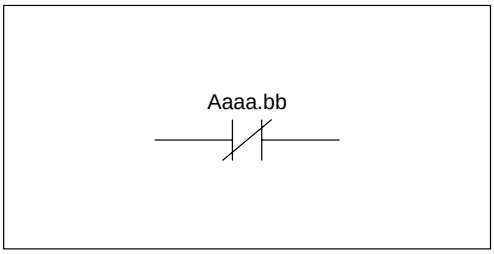
Store Bit-of-Word (STRB)

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)

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		DL350 Range	
	A	aaa	bb
Vmemory	B	All (See p.3–29)	BCD, 0 to 15
Pointer	PB	All (See p 3–29)	BCD, 0 to 15

In the following Store Bit-of-Word example, when bit 12 of V-memory location V1400 is on, output Y2 will energize.

DirectSOFT

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.

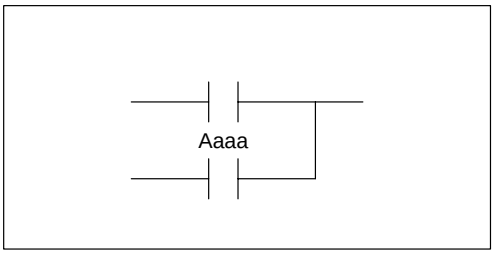
DirectSOFT

Handheld Programmer Keystrokes

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

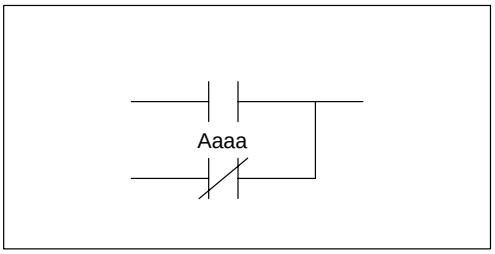
Or (OR)

The Or instruction logically ors 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)

The Or Not instruction logically ors 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		DL350 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	→	1	ENT
OR	→	2	ENT
OUT	→	5	ENT

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

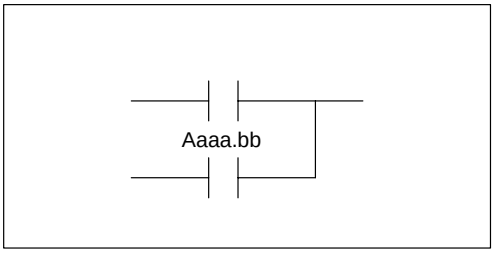
***Direct*SOFT**

Handheld Programmer Keystrokes

STR	→	1	ENT
ORN	→	2	ENT
OUT	→	5	ENT

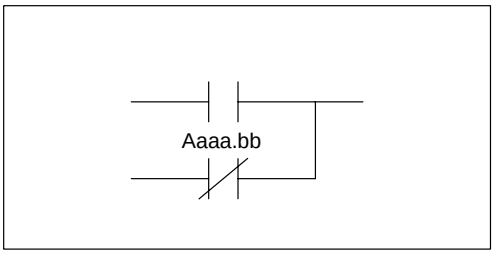
Or Bit-of-Word (ORB)

The Or Bit-of-Word instruction logically ors 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 (ORNB)

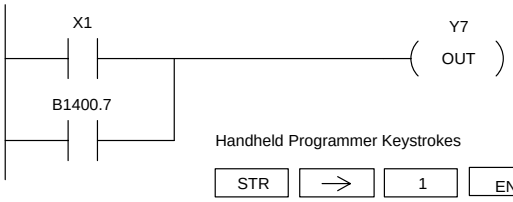
The Or Not Bit-of-Word instruction logically ors a normally closed contact in parallel with another contact in a rung. Status of the contact will be opposite the state of the bit referenced in the associated memory location.



Operand Data Type		DL350 Range	
	A	aaa	bb
Vmemory	B	All (See p. 3–29)	BCD, 0 to 15
Pointer	PB	All (See p.3–29)	BCD

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

DirectSOFT

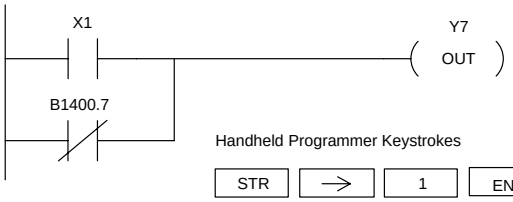


Handheld Programmer Keystrokes

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 or bit 7 of V1400 is off, output Y7 will energize.

DirectSOFT

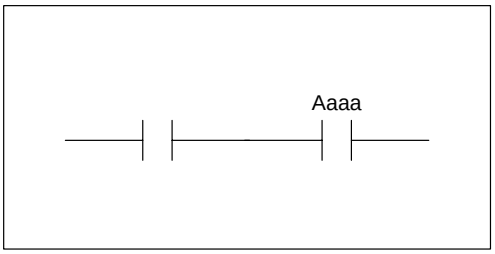


Handheld Programmer Keystrokes

STR	→	1	ENT					
ORN	SHFT	B	→	V	1	4	0	0
→	K	7	ENT					
OUT	→	5	ENT					

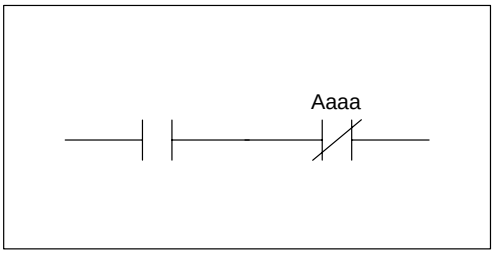
And (AND)

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)

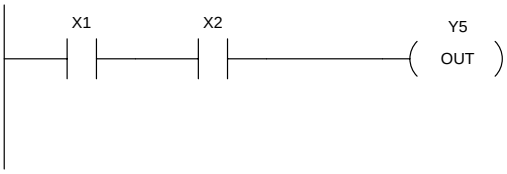
The And Not instruction logically ands a normally closed contact in series with another contact in a rung. The status of the contact will be opposite the state of the associated image register point or memory location.



Operand Data Type		DL350 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.

DirectSOFT

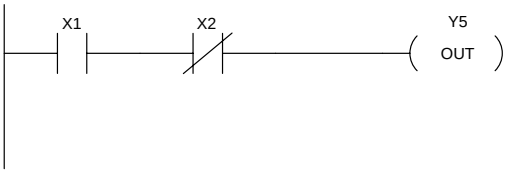


Handheld Programmer Keystrokes

STR	→	1	ENT
AND	→	2	ENT
OUT	→	5	ENT

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

DirectSOFT

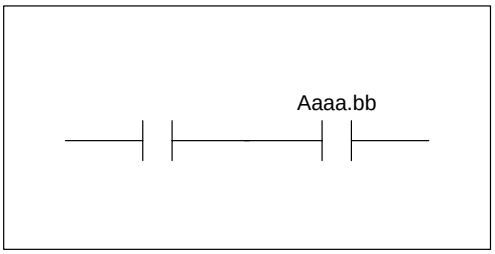


Handheld Programmer Keystrokes

STR	→	1	ENT
ANDN	→	2	ENT
OUT	→	5	ENT

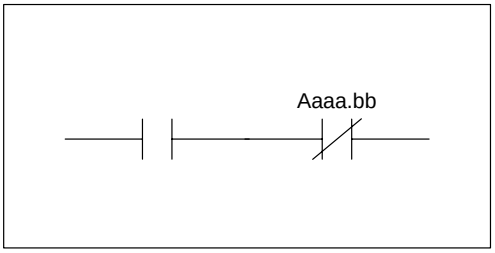
And Bit-of-Word (ANDB)

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 (ANDNB)

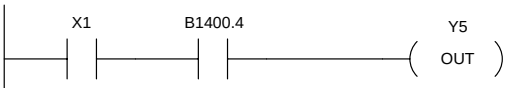
The And Not Bit-of-Word instruction logically ands a normally closed contact in series with another contact in a rung. The status of the contact will be opposite the state of the bit referenced in the associated memory location.



Operand Data Type		DL350 Range	
	A	aaa	bb
Vmemory	B	All (See p. 3–29)	BCD, 0 to 15
Pointer	PB	All (See p. 3–29)	BCD

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

DirectSOFT

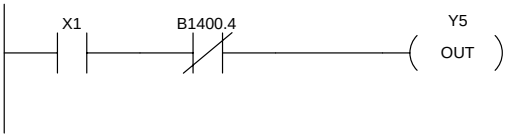


Handheld Programmer Keystrokes

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.

DirectSOFT

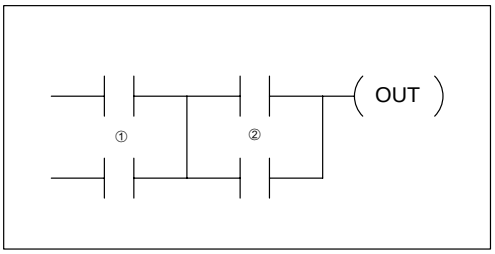


Handheld Programmer Keystrokes

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

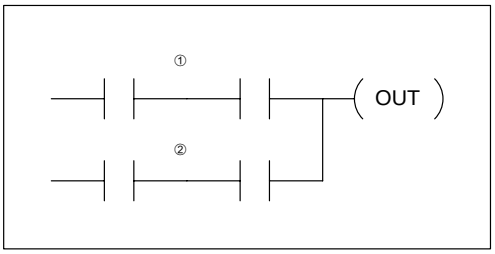
And Store (AND STR)

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



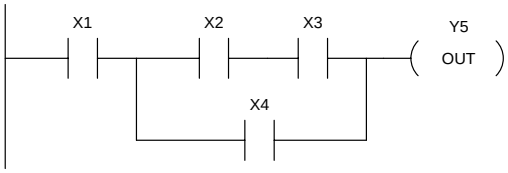
Or Store (OR STR)

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.

DirectSOFT

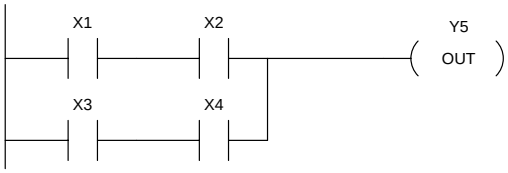


Handheld Programmer Keystrokes

STR	→	1	ENT
STR	→	2	ENT
AND	→	3	ENT
OR	→	4	ENT
ANDST	ENT		
OUT	→	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.

DirectSOFT

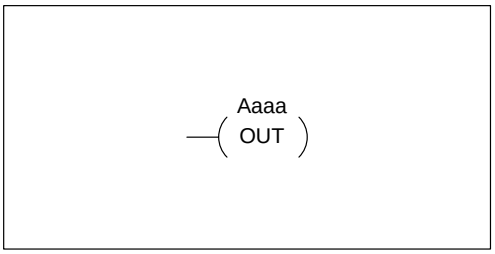


Handheld Programmer Keystrokes

STR	→	1	ENT
AND	→	2	ENT
STR	→	3	ENT
AND	→	4	ENT
ORST	ENT		
OUT	→	5	ENT

Out (OUT)

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.



Operand Data Type		DL350 Range
	A	**aaa**
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777

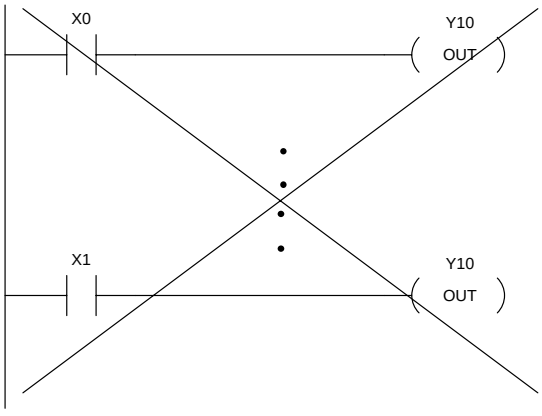
In the following Out example, when input X1 is on, output Y2 and Y5 will energize.

DirectSOFT

Handheld Programmer Keystrokes

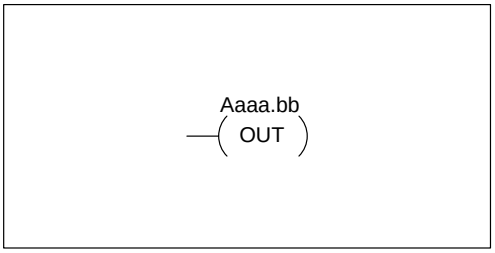
STR	→	1	ENT
OUT	→	2	ENT
OUT	→	5	ENT

In the following Out example the program contains two Out instructions using the same location (Y10). The physical output of Y10 is ultimately controlled by the last rung of logic referencing Y10. X1 will override the Y10 output being controlled by X0. To avoid this situation, multiple outputs using the same location should not be used in programming. If you need to have an output controlled by multiple inputs see the OROUT instruction on page 5–17.



Out Bit-of-Word (OUTB)

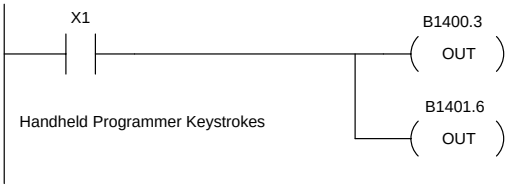
The Out Bit-of-Word instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) state to the specified bit in the referenced memory location. Multiple Out Bit-of-Word instructions referencing the same bit of the same word generally should not be used since only the last Out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



Operand Data Type		DL350 Range	
	A	aaa	bb
Vmemory	B	All (See p. 3–29)	BCD, 0 to 15
Pointer	PB	All (See p. 3–29)	BCD

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.

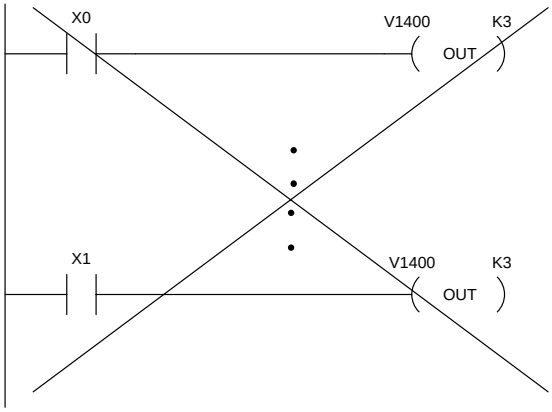
DirectSOFT



Handheld Programmer Keystrokes

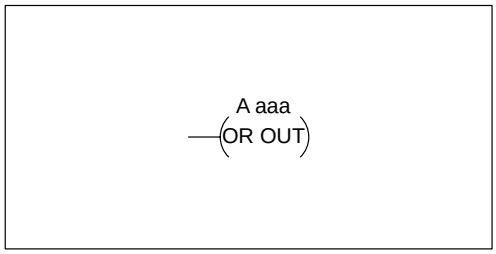
STR	→	1	ENT					
OUT	SHFT	B	→	V	1	4	0	0
→	K	3	ENT					
OUT	SHFT	B	→	V	1	4	0	0
→	K	4	ENT					

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.



Or Out (OR OUT)

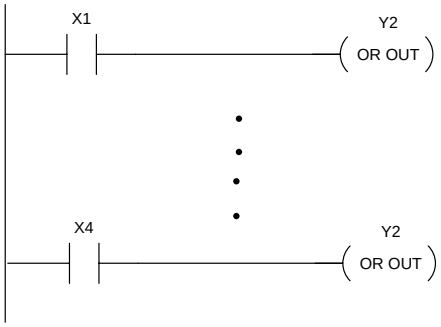
The Or Out instruction has been designed to used more than 1 rung of discrete logic to control a single output. Multiple Or Out instructions referencing the same output coil may be used, since *all* contacts controlling the output are ored together. If the status of *any* rung is on, the output will also be on.



Operand Data Type		DL350 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.

DirectSOFT

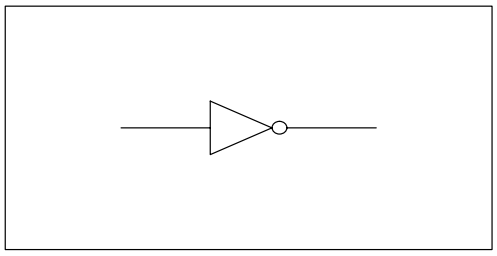


Handheld Programmer Keystrokes

STR	→	1	ENT				
INST#	3	5	ENT	ENT	→	2	ENT
STR	→	4	ENT				
INST#	3	5	ENT	ENT	→	2	ENT

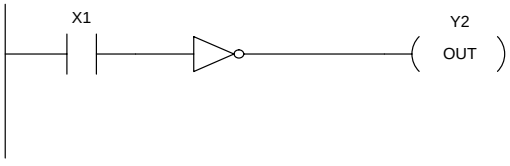
Not (NOT)

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.

DirectSOFT



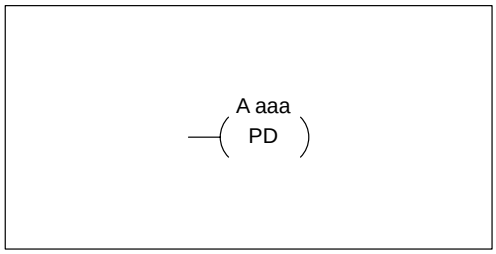
Handheld Programmer Keystrokes

STR	→	1	ENT	
SHFT	N	O	T	ENT
OUT	→	2	ENT	

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 rung cannot be created or displayed in ***Direct***SOFT versions earlier than 1.1i.

Positive Differential (PD)

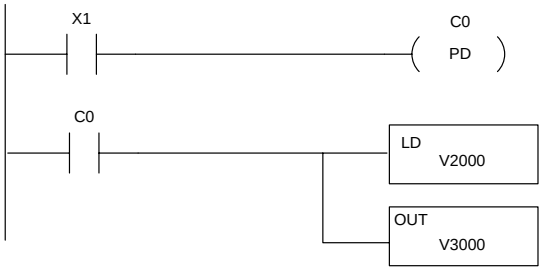
The Positive Differential instruction is typically known as a one shot. When the input logic produces an off to on transition, the output will energize for one CPU scan.



Operand Data Type		DL350 Range
	A	**aaa**
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777

In the following example, every time X1 is makes an off to on transition, C0 will energize for one scan.

DirectSOFT

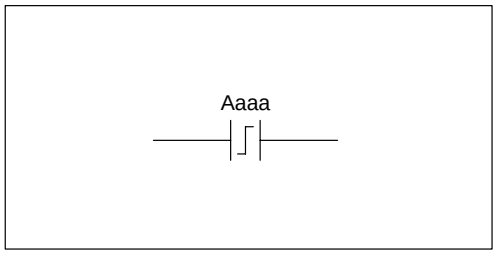


Handheld Programmer Keystrokes

STR	→	1	ENT			
SHFT	P	SHFT	D	→	0	ENT

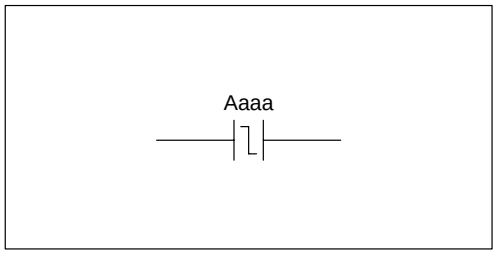
Store Positive Differential (STRPD)

The Store Positive Differential instruction begins a new rung or an additional branch in a rung with a normally open 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”.



Store Negative Differential (STRND)

The Store Negative Differential instruction begins a new rung or an additional branch in a rung with a normally closed 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).



Operand Data Type		DL350 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 is makes an Off-to-On transition, Y4 will energize for one scan.

DirectSOFT

Handheld Programmer Keystrokes

STR	SHFT	P	D	→
X	1	ENT		
OUT	Y	4	ENT	

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

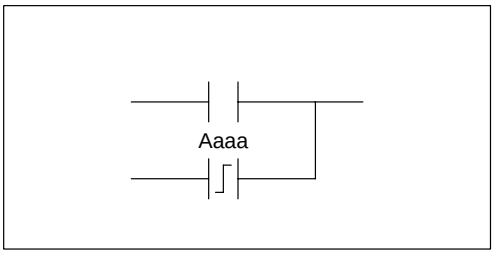
DirectSOFT

Handheld Programmer Keystrokes

STR	SHFT	N	D	→
X	1	ENT		
OUT	Y	4	ENT	

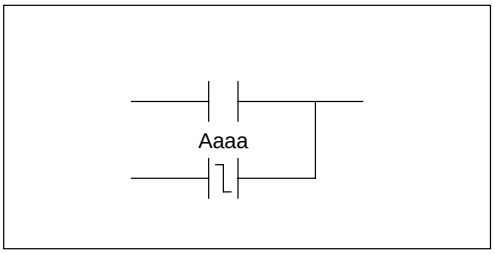
Or Positive Differential (ORPD)

The Or Positive Differential instruction logically ors a normally open 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)

The Or Negative Differential instruction logically ors a normally open 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		DL350 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.

DirectSOFT

Handheld Programmer Keystrokes

STR	X	1	ENT	
OR	SHFT	P	D	→
X	2	ENT		
OUT	Y	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.

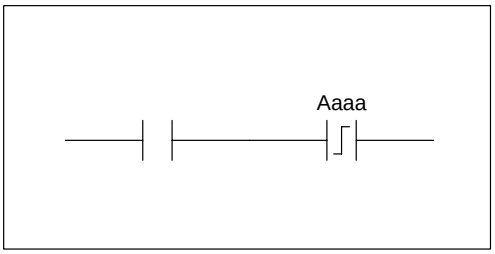
DirectSOFT

Handheld Programmer Keystrokes

STR	X	1	ENT	
OR	SHFT	N	D	→
X	2	ENT		
OUT	Y	5	ENT	

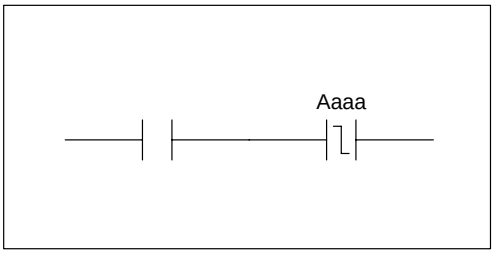
And Positive Differential (ANDPD)

The And Positive Differential instruction logically ands a normally open 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.



And Negative Differential (ANDND)

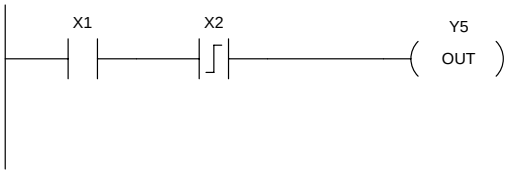
The And Negative Differential instruction logically ands a normally open contact in parallel with another contact in a rung. The status of the contact will be open until the associated image register point makes an On-to-Off transition, closing it for one CPU scan. Thereafter, it remains open until another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



Operand Data Type		DL350 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.

DirectSOFT

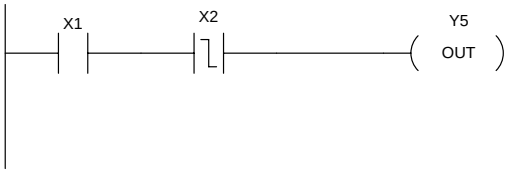


Handheld Programmer Keystrokes

STR	X	1	ENT	
AND	SHFT	P	D	→
X	2	ENT		
OUT	Y	5	ENT	

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

DirectSOFT

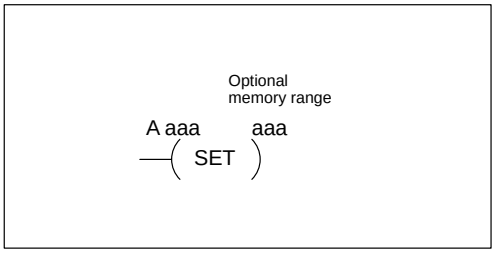


Handheld Programmer Keystrokes

STR	X(IN)	1	ENT	
AND	SHFT	N	D	→
X(IN)	2	ENT		
OUT	Y(OUT)	5	ENT	

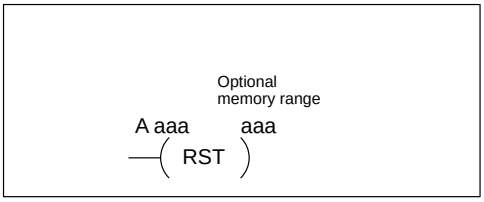
Set (SET)

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)

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		DL350 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

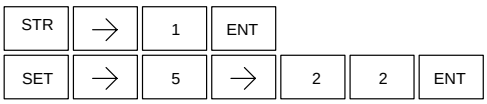
* Timer and counter operand data types are not valid using the Set instruction.

NOTE: You cannot set inputs (X's) that are assigned to input modules

In the following example when X1 is on, Y5 through Y22 will energize.

***Direct*SOFT**

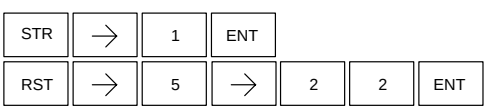
Handheld Programmer Keystrokes



In the following example when X1 is on, Y5 through Y22 will be reset or de–energized.

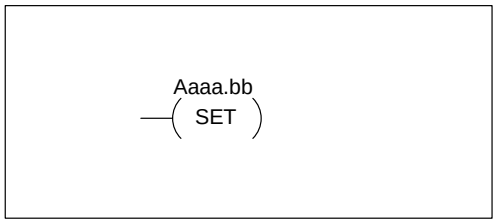
***Direct*SOFT**

Handheld Programmer Keystrokes



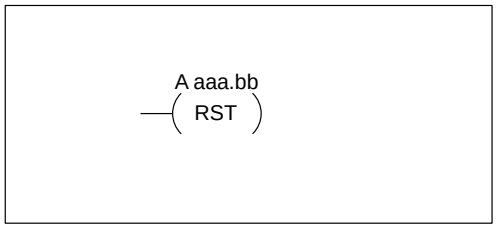
Set Bit-of-Word (SETB)

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 (RSTB)

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.

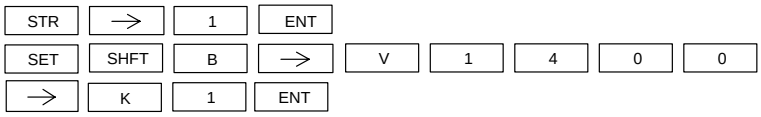


Operand Data Type		DL350 Range	
	A	aaa	bb
Vmemory	B	All (See p. 3–29)	0 to 15
Pointer	PB	All (See p. 3–29)	0 to 15

In the following example when X1 turns on, bit 0 in V1400 is set to the on state.

DirectSOFT

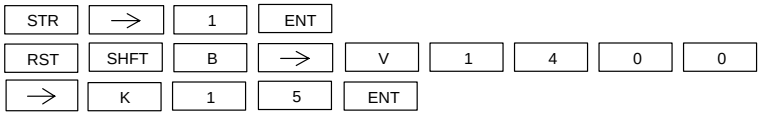
Handheld Programmer Keystrokes



In the following example when X1 turns on, bit 15 in V1400 is reset to the off state.

DirectSOFT

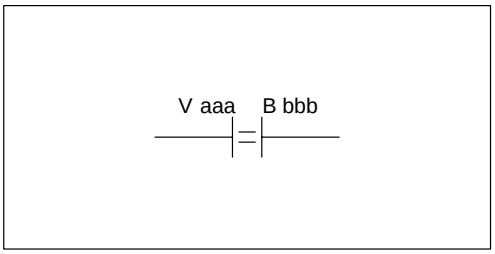
Handheld Programmer Keystrokes



Comparative Boolean

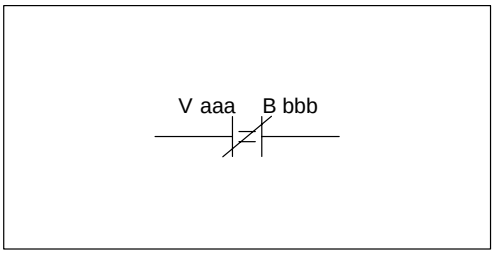
Store If Equal (STRE)

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 =Bbbb .



Store If Not Equal (STRNE)

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 ≠ Bbbb.



Operand Data Type		DL350 Range	
	B	**aaa**	**bbb**
V memory	V	All (See page 3–29)	All (See page 3–29)
Pointer	P	—	All V mem. (See page 3–29)
Constant	K	—	0–FFFF

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

***Direct*SOFT**

Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0
→	E 4	J 9	D 3	D 3	ENT		
GX OUT	→	D 3	ENT				

In the following example, when the value in V memory location V2000 ≠ 5060, Y3 will energize.

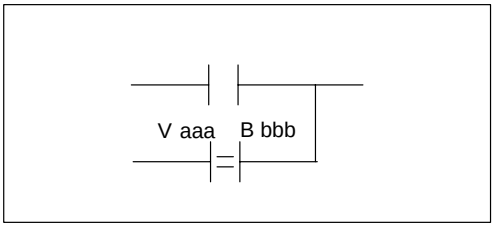
***Direct*SOFT**

Handheld Programmer Keystrokes

SP STRN	SHFT	E 4	→	C 2	A 0	A 0	A 0
→	F 5	A 0	G 6	A 0	ENT		
GX OUT	→	D 3	ENT				

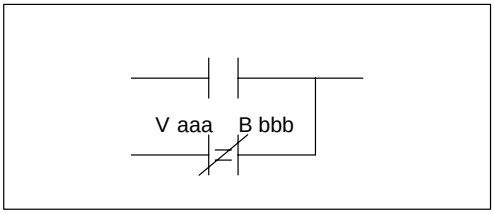
Or If Equal (ORE)

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)

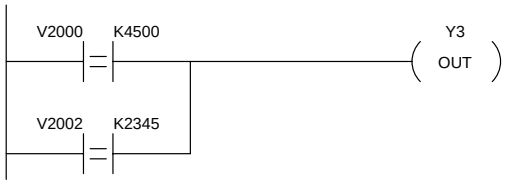
The Or If Not Equal instruction connects a normally closed comparative contact in parallel with another contact. The contact will be on when Vaaa ≠ Bbbb.



Operand Data Type		DL350 Range	
	B	**aaa**	**bbb**
V memory	V	All (See page 3–29)	All (See page 3–29)
Pointer	P	—	All V mem. (See page 3–29)
Constant	K	—	0–FFFF

In the following example, when the value in V memory location V2000 = 4500 or V2002 = 2345 , Y3 will energize.

DirectSOFT

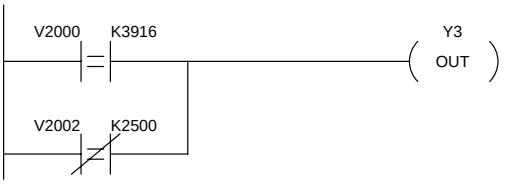


Handheld Programmer Keystrokes

$ 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 value in V memory location V2000 = 3916 or V2002 ≠ 2500, Y3 will energize.

DirectSOFT

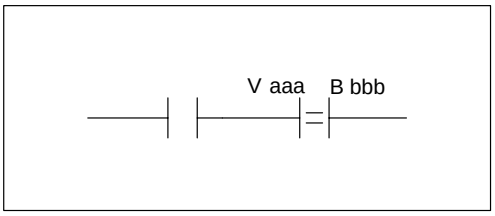


Handheld Programmer Keystrokes

$ 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					

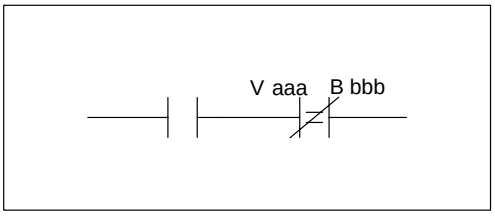
And If Equal (ANDE)

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)

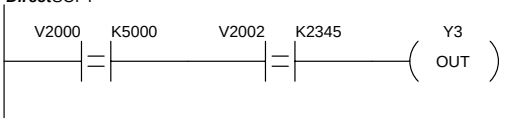
The And If Not Equal instruction connects a normally closed comparative contact in series with another contact. The contact will be on when Vaaa ≠ Bbbb



Operand Data Type		DL350 Range	
	A/B	aaa	bbb
V memory	V	All (See page 3–29)	All (See page 3–29)
Pointer	P	—	All V mem. (See page 3–29)
Constant	K	—	0–FFFF

In the following example, when the value in V memory location V2000 = 5000 and V2002 = 2345, Y3 will energize.

DirectSOFT

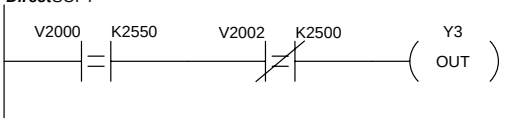


Handheld Programmer Keystrokes

$ 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 value in V memory location V2000 = 2550 and V2002 ≠ 2500, Y3 will energize.

DirectSOFT

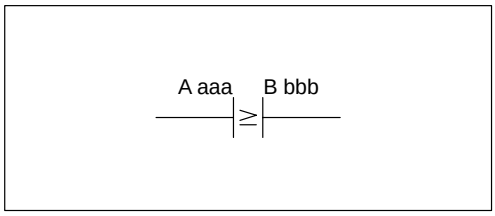


Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
C 2	F 5	F 5	A 0	ENT				
W ANDN	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					

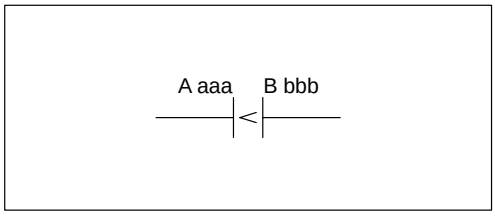
Store (STR)

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 ≥ Bbbb.



Store Not (STRN)

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.

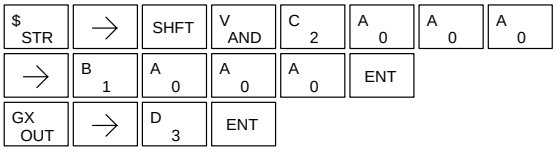


Operand Data Type		DL350 Range	
	A/B	aaa	bbb
V memory	V	All (See page 3–29)	All (See page 3–29)
Pointer	P	—	All V mem. (See page 3–29)
Constant	K	—	0–FFFF
Timer	T	0–377	
Counter	CT	0–177	

In the following example, when the value in V memory location V2000 ≥ 1000, Y3 will energize.

DirectSOFT

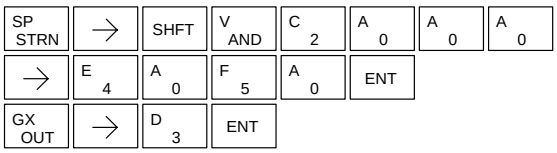
Handheld Programmer Keystrokes



In the following example, when the value in V memory location V2000 < 4050, Y3 will energize.

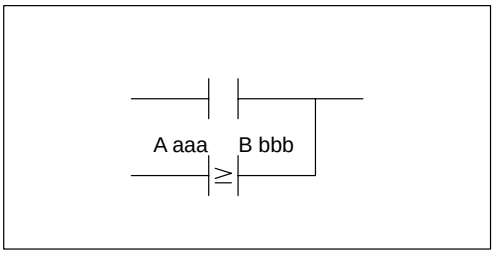
DirectSOFT

Handheld Programmer Keystrokes



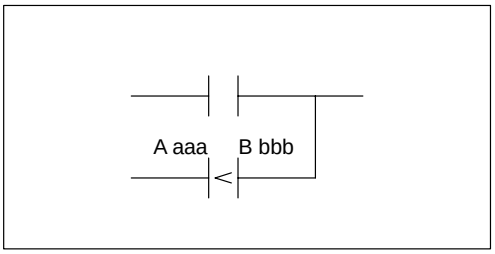
Or (OR)

The Comparative Or instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when Aaaa ≥ Bbbb.



Or Not (ORN)

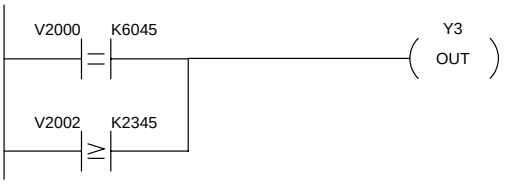
The Comparative Or Not instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when Aaaa < Bbbb.



Operand Data Type		DL350 Range	
	A/B	aaa	bbb
V memory	V	All (See page 3–29)	All (See page 3–29)
Pointer	P	—	All V mem. (See page 3–29)
Constant	K	—	0–FFFF
Timer	T	0–377	
Counter	CT	0–177	

In the following example, when the value in V memory location V2000 = 6045 or V2002 ≥ 2345, Y3 will energize.

***Direct*SOFT**

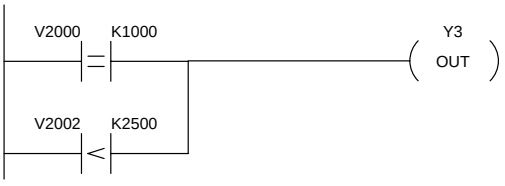


Handheld Programmer Keystrokes

$ 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 value in V memory location V2000 = 1000 or V2002 < 2500, Y3 will energize.

***Direct*SOFT**

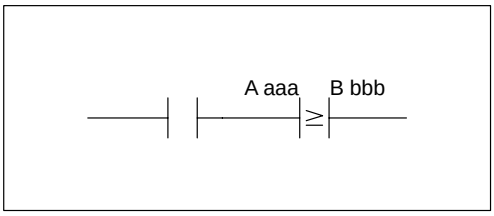


Handheld Programmer Keystrokes

$ 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					

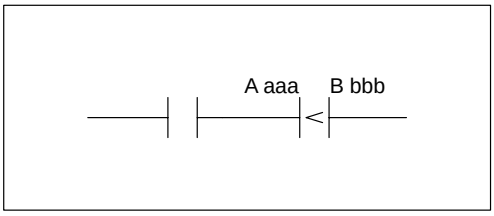
And (AND)

The Comparative And instruction connects a normally open comparative contact in series with another contact. The contact will be on when Aaaa ≥ Bbbb.



And Not (ANDN)

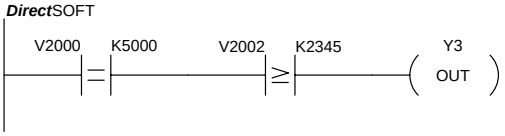
The Comparative And Not instruction connects a normally open comparative contact in series with another contact. The contact will be on when Aaaa < Bbbb.



Operand Data Type		DL350 Range	
	A/B	aaa	bbb
V memory	V	All (See page 3–29)	All (See page 3–29)
Pointer	P	—	All V mem. (See page 3–29)
Constant	K	—	0–FFFF
Timer	T	0–377	
Counter	CT	0–177	

In the following example, when the value in V memory location V2000 = 5000, and V2002 ≥ 2345, Y3 will energize.

DirectSOFT

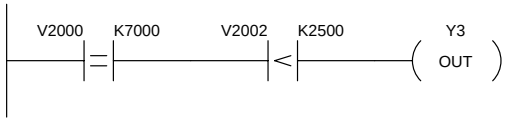


Handheld Programmer Keystrokes

$ 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.

DirectSOFT



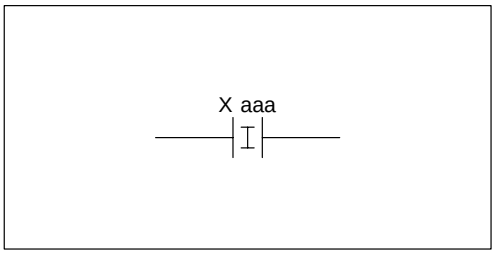
Handheld Programmer Keystrokes

$ 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

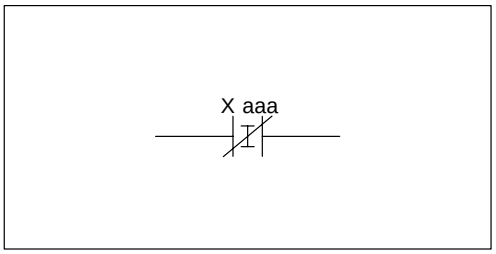
Store Immediate (STRI)

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 on the module *at the time the instruction is executed*. The image register is not updated.



Store Not Immediate (STRNI)

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 on the module *at the time the instruction is executed*. The image register is not updated.

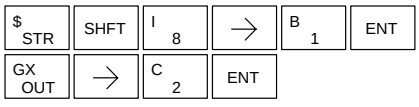


Operand Data Type		DL350 Range
		aaa
Inputs	X	0–777

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

DirectSOFT

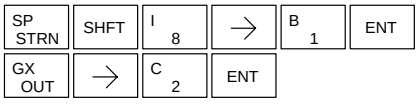
Handheld Programmer Keystrokes



In the following example when X1 is off, Y2 will energize.

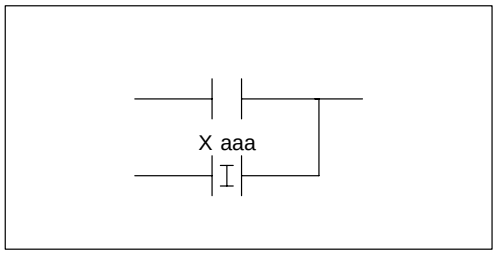
DirectSOFT

Handheld Programmer Keystrokes



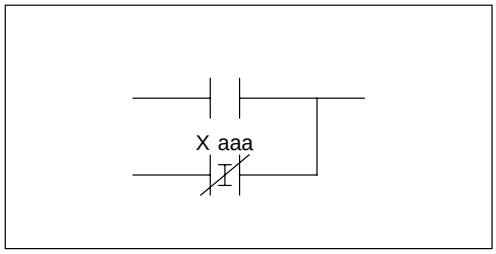
Or Immediate (ORI)

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 on the module *at the time the instruction is executed*. The image register is not updated.



Or Not Immediate (ORNI)

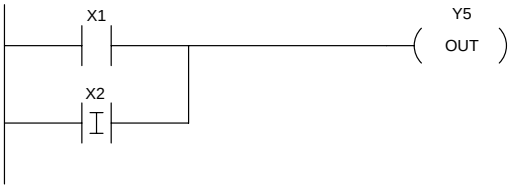
The Or Not Immediate connects two contacts in parallel. The status of the contact will be opposite the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.



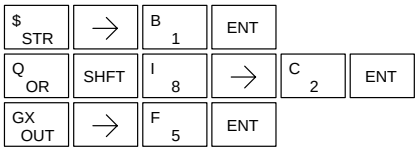
Operand Data Type		DL350 Range
		aaa
Inputs	X	0–777

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

DirectSOFT

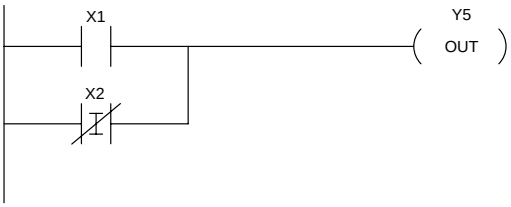


Handheld Programmer Keystrokes

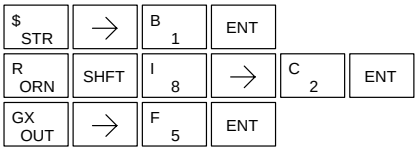


In the following example, when X1 is on or X2 is off, Y5 will energize.

DirectSOFT

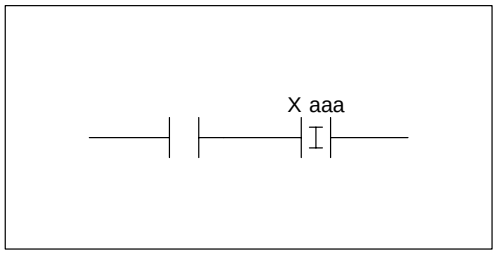


Handheld Programmer Keystrokes



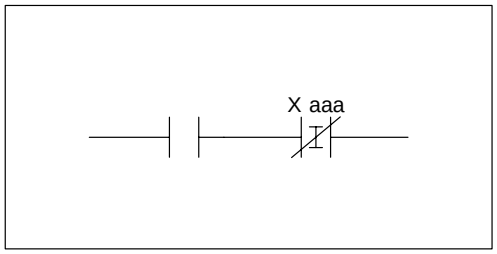
And Immediate (ANDI)

The And Immediate connects two contacts in series. The status of the contact will be the same as the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.



And Not Immediate (ANDNI)

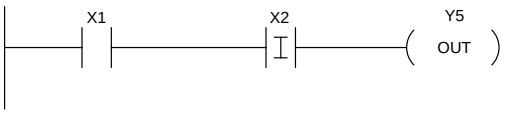
The And Not Immediate connects two contacts in series. The status of the contact will be opposite the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.



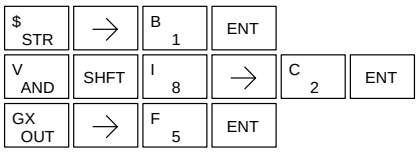
Operand Data Type		DL350 Range
		aaa
Inputs	X	0–777

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

DirectSOFT

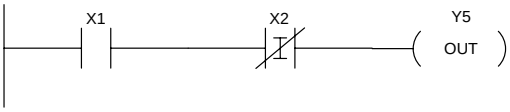


Handheld Programmer Keystrokes

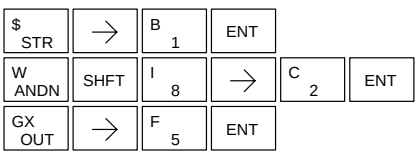


In the following example, when X1 is on and X2 is off, Y5 will energize.

DirectSOFT

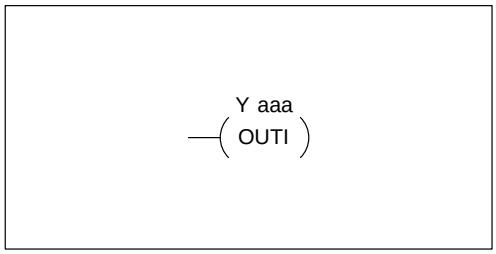


Handheld Programmer Keystrokes



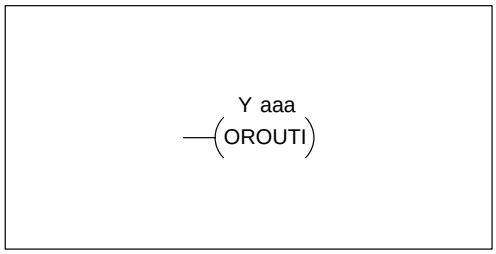
Out Immediate (OUTI)

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)

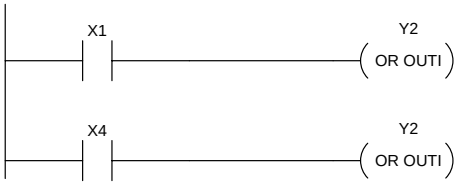
The Or Out Immediate instruction has been designed to use more than 1 rung of discrete logic to control a single output. Multiple Or Out Immediate instructions referencing the same output coil may be used, since all contacts controlling the output are ored together. If the status of *any* rung is on *at the time the instruction is executed*, the output will also be on.



Operand Data Type		DL350 Range
		aaa
Outputs	Y	0–777

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

DirectSOFT

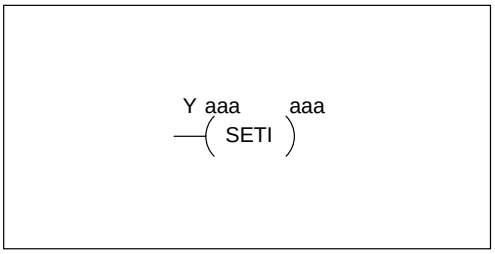


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
O INST#	D 3	F 5	A 0	ENT	ENT	
→	C 2	ENT				
$ STR	→	E 4	ENT			
O INST#	D 3	F 5	A 0	ENT	ENT	
→	C 2	ENT				

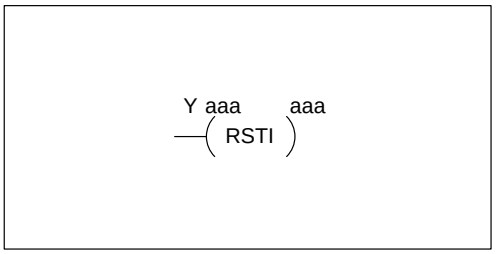
Set Immediate (SETI)

The Set Immediate instruction immediately sets, or turns on an output or a range of outputs in the image register and the corresponding output module(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)

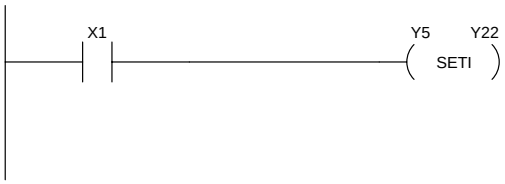
The Reset Immediate instruction immediately resets, or turns off an output or a range of outputs in the image register and the output module(s) *at the time the instruction is executed*. Once the outputs are reset it is not necessary for the input to remain on.



Operand Data Type		DL350 Range
		aaa
Outputs	Y	0–777

In the following example, when X1 is on, Y5 through Y22 will be set on in the image register and on the corresponding output module(s).

DirectSOFT



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
X SET	SHFT	I 8	→	F 5	→	C 2	C 2	ENT

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).

DirectSOFT

Handheld Programmer Keystrokes

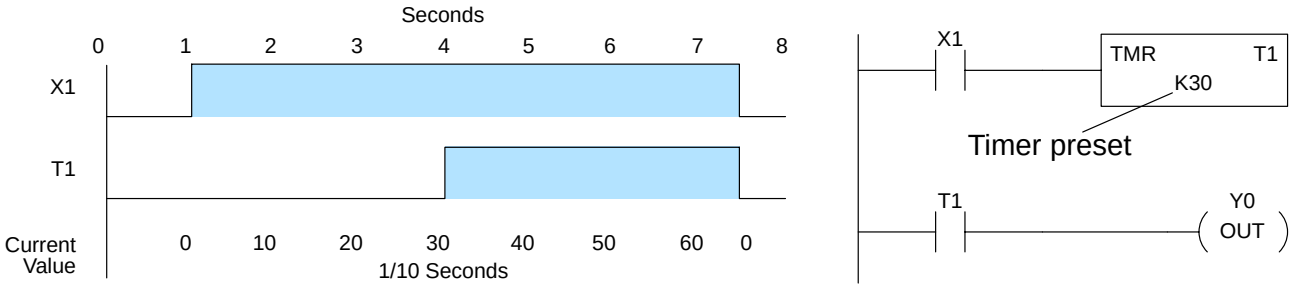
$ STR	→	B 1	ENT					
S RST	SHFT	I 8	→	F 5	→	C 2	C 2	ENT

Timer, Counter and Shift Register Instructions

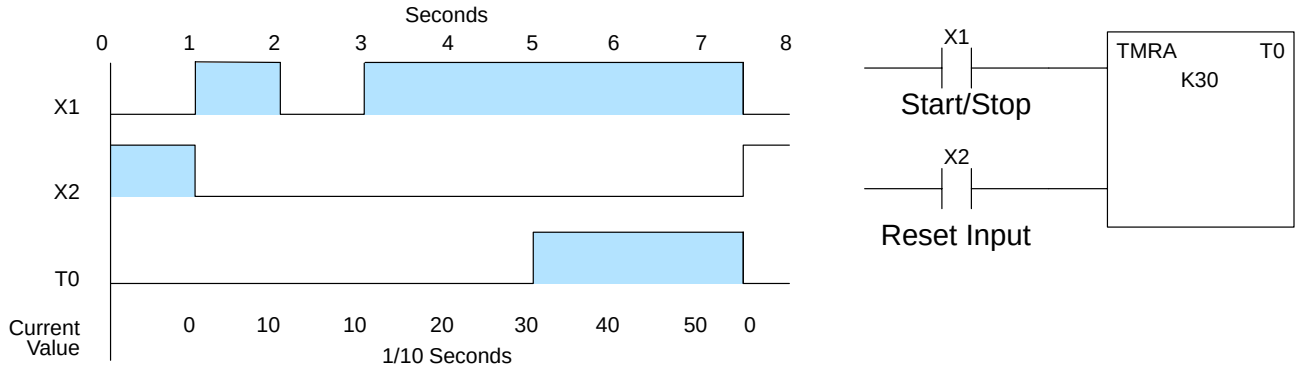
Using Timers

Timers are used to time an event for a desired length of time. 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 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 discrete bit associated with each timer to indicate 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.

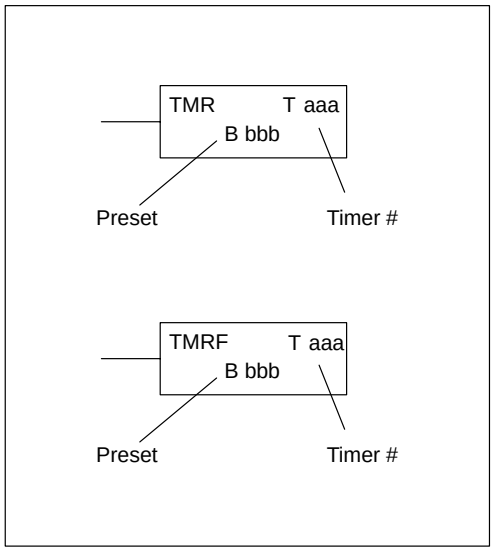


The accumulating timer works similarly to the regular timer, but two inputs are required. The start/stop input starts and stops the timer. When the timer stops, the elapsed time is maintained. When the timer starts again, the timing continues from the elapsed time. When the reset input is turned on, 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.



Timer (TMR) and Timer Fast (TMRF)

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).



The timer discrete status bit and the current value are not specified in the timer instruction.

Instruction Specifications

Timer Reference (Taaa): Specifies the timer number.

Preset Value (Bbbb): Constant value (K), V memory location, or Pointer (P).

Current Value: Timer current values 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 accessed by referencing the associated T memory location. 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.

Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Timers	T	0–377	—
V memory for preset values	V	—	All Data Words (See Page 3–29)
Pointers (preset only)	P	—	All Data Words (See Page 3–29)
Constants (preset only)	K	—	0–9999
Timer discrete status bits	T/V	0–377	
Timer current values	V /T*	0–377	

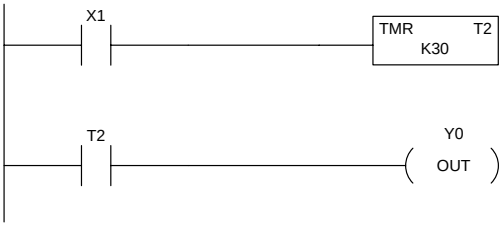
There are two methods of programming timers. You can perform functions when the timer reaches the specified preset using the the discrete status bit, or use the comparative contacts to perform functions at different time intervals based on one timer. The following examples show each method of using timers.

NOTE: The current value of a timer can be accessed by using the TA data type (i.e., TA2). Current values may also be accessed by the V-memory location.

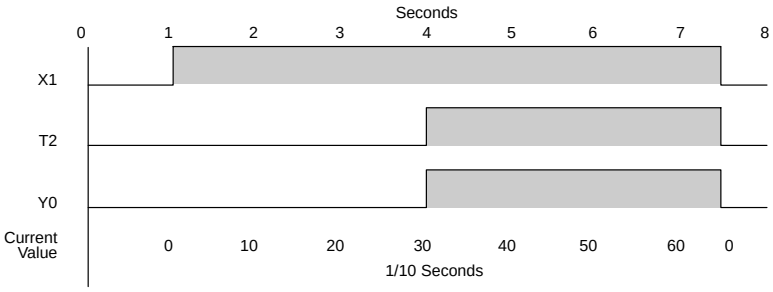
Timer Example Using Discrete Status Bits

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.

DirectSOFT



Timing Diagram



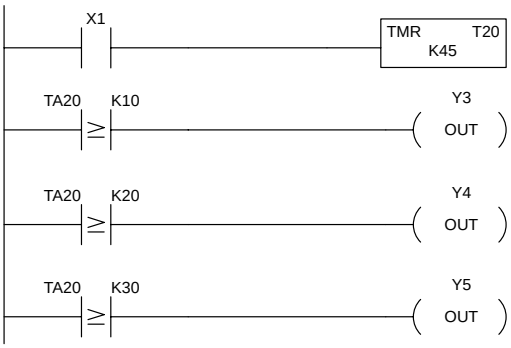
Handheld Programmer Keystrokes

$ 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					

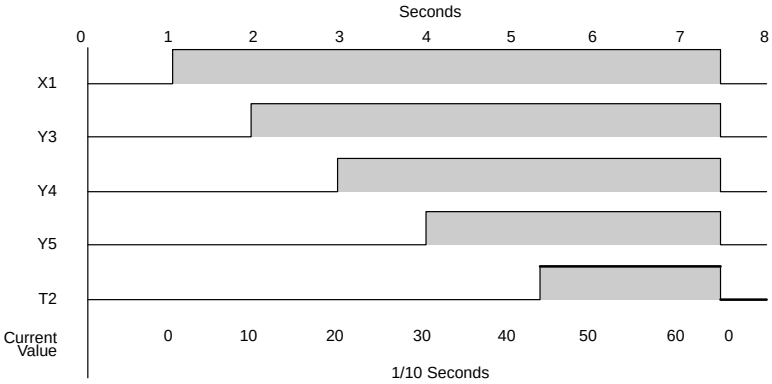
Timer Example Using Comparative Contacts

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.

DirectSOFT



Timing Diagram



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) Accumulating Fast Timer (TMRAF)

The Accumulating Timer is a 0.1 second two input timer that will time to a maximum of 9999999.9. The Accumulating Fast Timer is a 0.01 second two input timer that will time to a maximum of 99999.99. These timers have two inputs, an enable and a reset. The timer will start timing when the enable is on and stop timing when the enable is off without resetting the current value to 0. The reset will reset the timer when on and allow the timer to time when off.

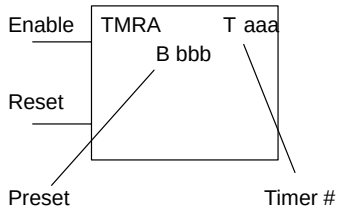
Instruction Specifications

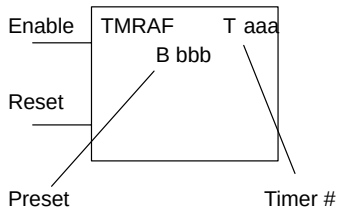
Timer Reference (Taaa): Specifies the timer number.

Preset Value (Bbbb): Constant value (K), V memory location,or Pointer (P).

Current Value: Timer current values are accessed by referencing the associated V or T memory location (See Note). For example, the timer current value for T3 resides in V-memory location V3.

Discrete Status Bit: The discrete status bit is accessed by referencing the associated T memory location. 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.





Caution: The TMRA uses two consecutive timer locations, since the preset can now be 8 digits, which requires two V-memory locations. For example, if TMRA T0 is used in the program, the next available timer would be T2. Or if T0 was a normal timer, and T1 was an accumulating timer, the next available timer would be T3.

The timer discrete status bit and the current value are not specified in the timer instruction.

Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Timers	T	0–377	—
V memory for preset values	V	—	All Data Words (See Page 3–29)
Pointers (preset only)	P	—	All Data Words (See Page 3–29)
Constants (preset only)	K	—	0–9999
Timer discrete status bits	T/V	0–377 or V41100–41117	
Timer current values	V /T*	0–377	

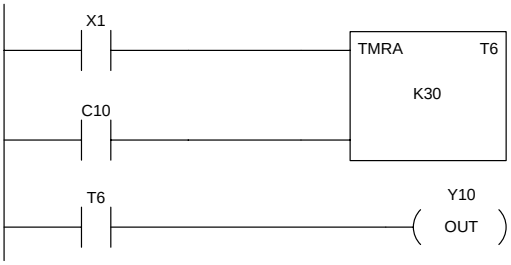
There are two methods of programming timers. You can perform functions when the timer reaches the specified preset using the the discrete status bit, or use the comparative contacts to perform functions at different time intervals based on one timer. The following examples show each method of using timers.

NOTE: The current value of a timer can be accessed by using the TA data type (i.e., TA2). Current values may also be accessed by the V-memory location.

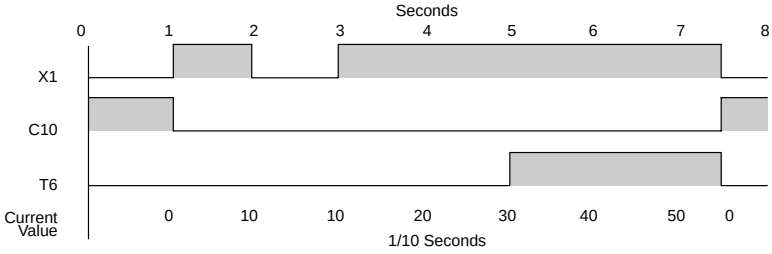
Accumulating Timer Example using Discrete Status Bits

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 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.

DirectSOFT



Timing Diagram



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
$ STR	→	SHFT	C 2	B 1	A 0	ENT		
N TMR	SHFT	A 0	→	G 6	→			

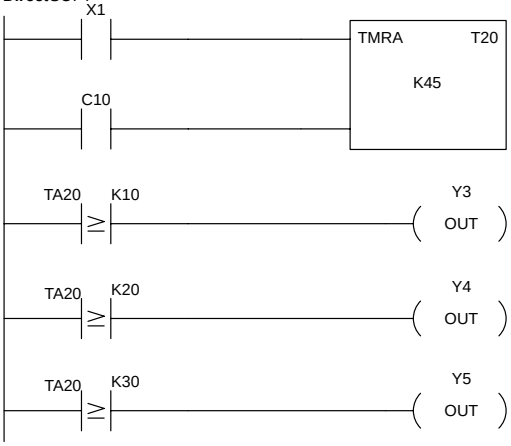
Handheld Programmer Keystrokes (cont)

D 3	A 0	ENT						
$ STR	→	SHFT	T MLR	G 6	ENT			
GX OUT	→	B 1	A 0	ENT				

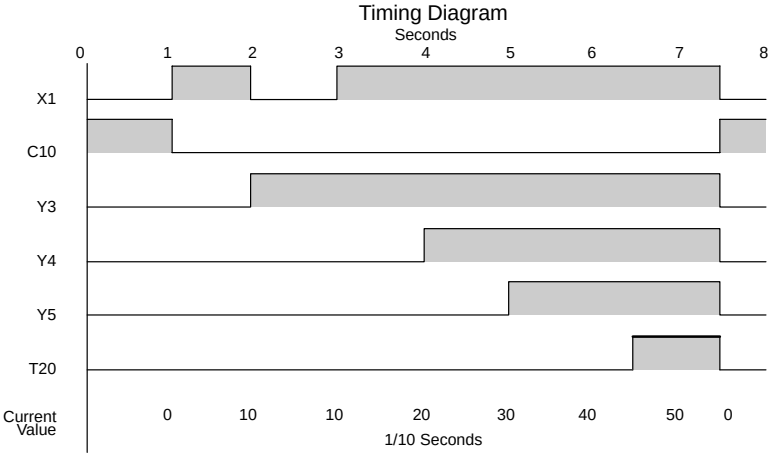
Accumulator Timer Example Using Comparative Contacts

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.

DirectSOFT



Timing Diagram



Handheld Programmer Keystrokes

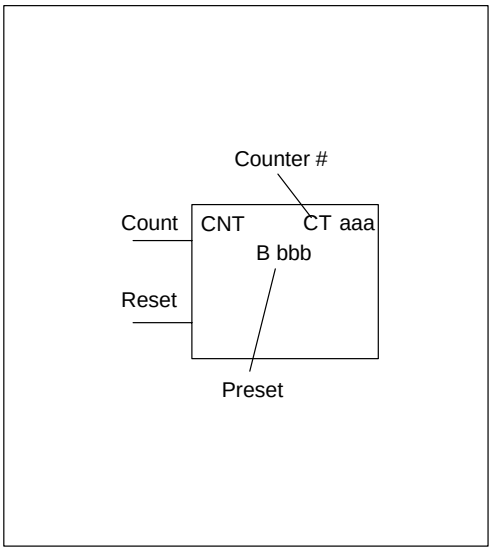
$ STR	→	B 1	ENT							
$ STR	→	SHFT	C 2	B 1	A 0	ENT				
N TMR	SHFT	A 0	→	C 2	A 0	→	E 4	F 5	ENT	
$ STR	→	SHFT	T MLR	C 2	A 0	→	B 1	A 0	ENT	
GX OUT	→	D 3	ENT							
$ STR	→	SHFT	T MLR	C 2	A 0	→	C 2	A 0	ENT	

Handheld Programmer Keystrokes (cont)

GX OUT	→	E 4	ENT					
$ STR	→	SHFT	T MLR	C 2	A 0	→		
D 3	A 0	ENT						
GX OUT	→	F 5	ENT					

Counter (CNT)

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.



The counter discrete status bit and the current value are not specified in the counter instruction.

Instruction Specifications

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K), V memory location, or Pointer (P).

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.

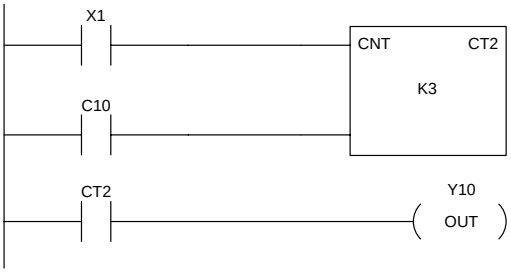
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Counters	CT	0–177	—
V memory (preset only)	V	—	All Data Words (See Page 3–29)
Pointers (preset only)	P	—	All Data Words (See Page 3–29)
Constants (preset only)	K	—	0–9999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V/CT*	1000–1177	

NOTE: The current value of a counter can be accessed by using the CTA data type (i.e., CTA2). Current values may also be accessed by the V-memory location.

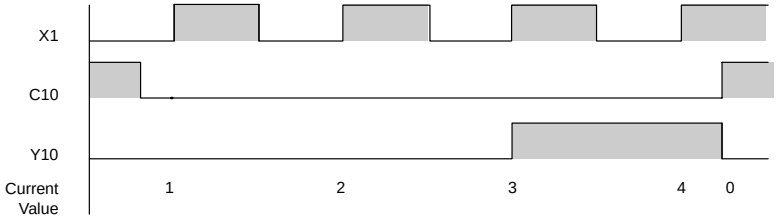
Counter Example Using Discrete Status Bits

In the following example, when X1 makes an off to on transition, counter CTA2 will increment by one. When the current value reaches the preset value of 3, the counter status bit CTA2 will turn on and energize Y10. 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 CTA2 will be held in V memory location V1002.

DirectSOFT



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			

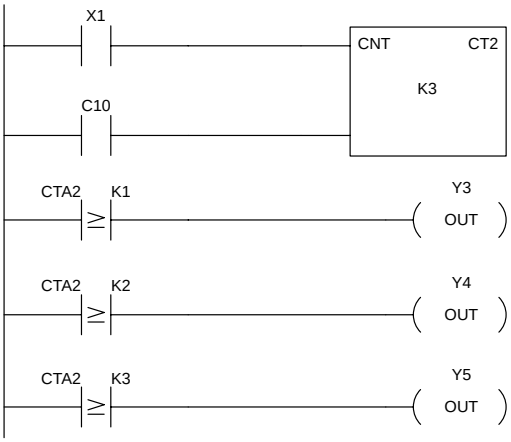
Handheld Programmer Keystrokes (cont)

$ STR	→	SHFT	C 2	SHFT	T MLR	C 2	ENT	
GX OUT	→	B 1	A 0	ENT				

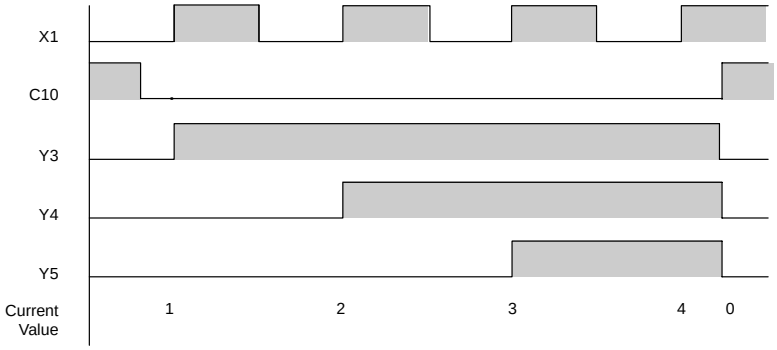
Counter Example Using Comparative Contacts

In the following example, when X1 makes an off to on transition, counter CTA2 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.

DirectSOFT



Counting diagram



Handheld Programmer Keystrokes

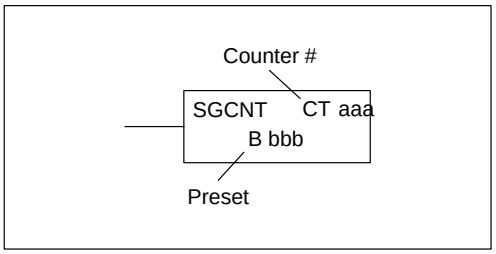
$ STR	→	B 1	ENT					
$ STR	→	SHFT	C 2	B 1	A 0	ENT		
GY CNT	→	C 2	→	D 3	ENT			
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2		
→	B 1	ENT						
GX OUT	→	D 3	ENT					

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)

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.



The counter discrete status bit and the current value are not specified in the counter instruction.

Instruction Specifications

Counter Reference (CTaaa): Specifies the counter number.
Preset Value (Bbbb): Constant value (K), V memory location or Pointer (P).
Current Values: Counter current values are accessed by referencing the associated V or CTA 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.

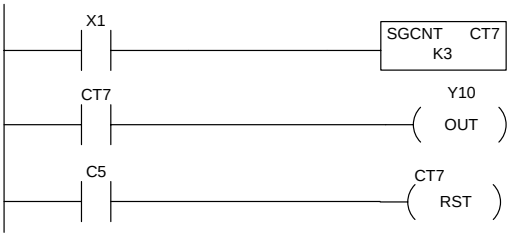
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Counters	CT	0–177	—
V memory (preset only)	V	—	All Data Words (See Page 3–29)
Pointers (preset only)	P	—	All Data Words (See Page 3–29)
Constants (preset only)	K	—	0–9999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V/CTA*	1000–1177	

NOTE: The current value of a counter can be accessed by using the CTA data type (i.e., CTA2). Current values may also be accessed by the V-memory location.

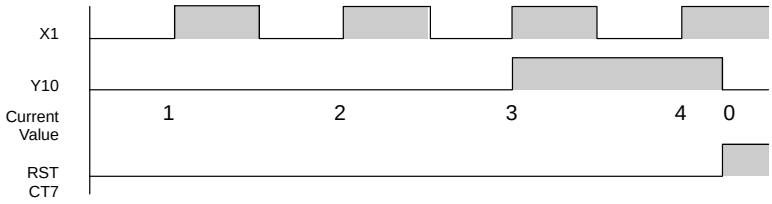
Stage Counter Example Using Discrete Status Bits

In the following example, when X1 makes an off to on transition, stage counter CTA7 will increment by one. When the current value reaches 3, the counter status bit CTA7 will turn on and energize Y10. The counter status bit CTA7 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 CTA7 will be held in V memory location V1007.

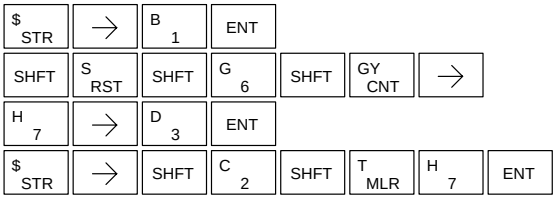
DirectSOFT



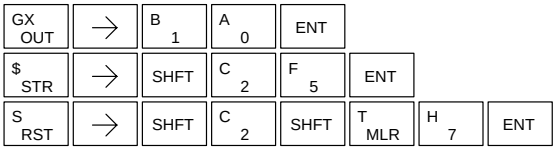
Counting diagram



Handheld Programmer Keystrokes



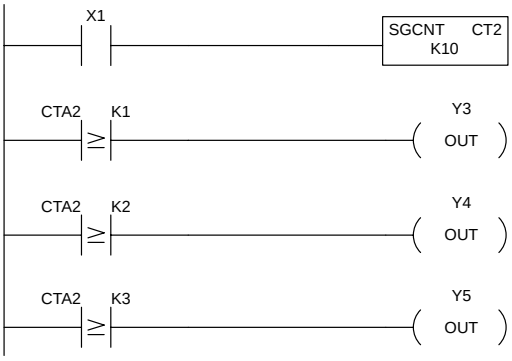
Handheld Programmer Keystrokes (cont)



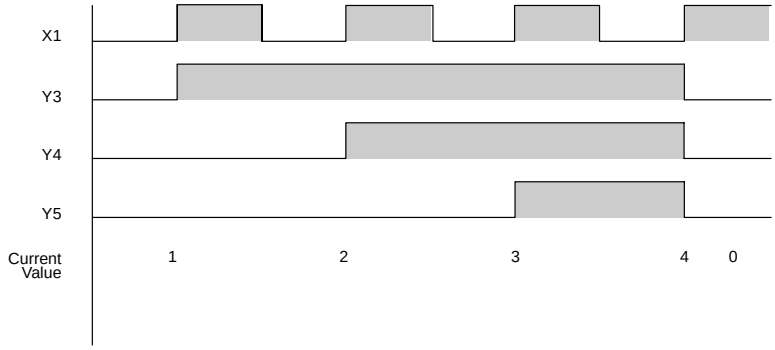
Stage Counter Example Using Comparative Contacts

In the following example, when X1 makes an off to on transition, counter CTA2 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 CTA2 will be held in V memory location V1007.

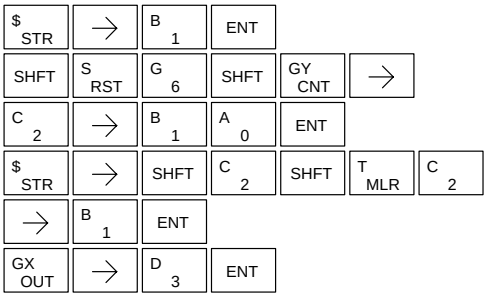
DirectSOFT



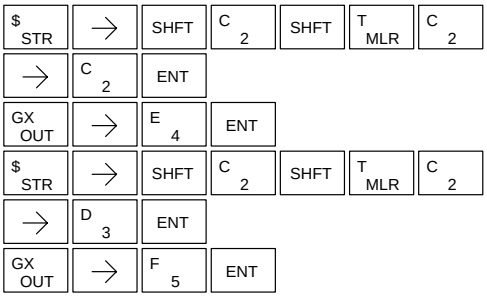
Counting diagram



Handheld Programmer Keystrokes

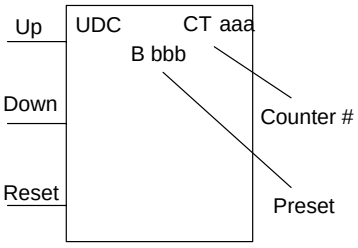


Handheld Programmer Keystrokes (cont)



Up Down Counter (UDC)

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 the UDC uses two consecutive counter locations. If UDC CT1 is used in the program, the next available counter is CT3.

The counter discrete status bit and the current value are not specified in the counter instruction.

Instruction Specification

Counter Reference (CTaaa): Specifies the counter number.
Preset Value (Bbbb): Constant value (K), V memory locations, or Pointer (P).
Current Values: Current count is a double word value 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 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. It will be on if value is equal to or greater than the preset value. For example the discrete status bit for counter 2 would be CT2.

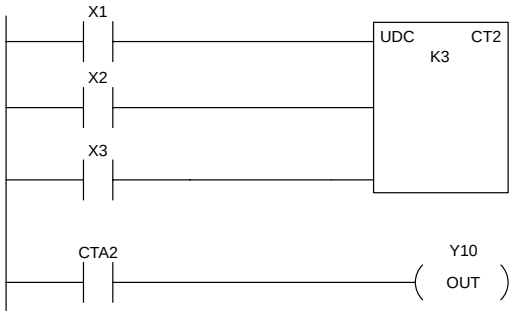
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Counters	CT	0–177	—
V memory (preset only)	V	—	All Data Words (See Page 3–29)
Pointers (preset only)	P	—	All Data Words (See Page 3–29)
Constants (preset only)	K	—	0–99999999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V/CTA*	1000–1177	

NOTE: The current value of a counter can be accessed by using the CTA data type (i.e., CTA2). Current values may also be accessed by the V-memory location.

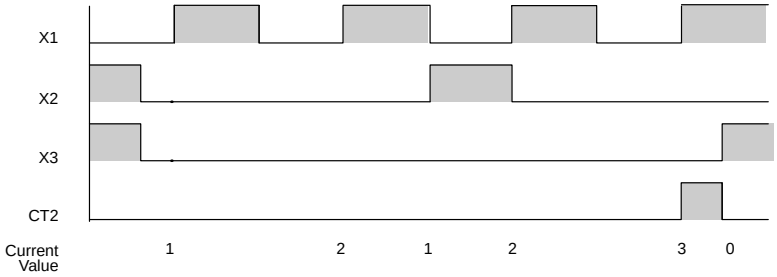
Up / Down Counter Example Using Discrete Status Bits

In the following example if X2 and X3 are off ,when X1 toggles from off to on the counter will increment by one. 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	

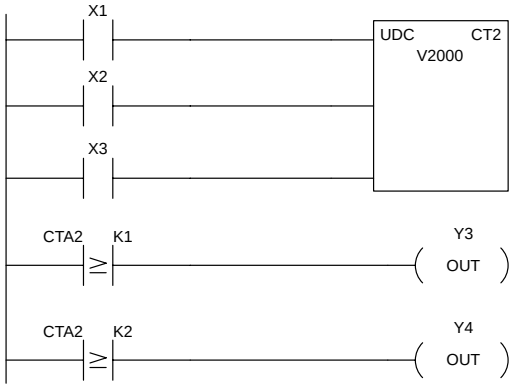
Handheld Programmer Keystrokes (cont)

→	D 3	ENT					
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2	ENT
GX OUT	→	B 1	A 0	ENT			

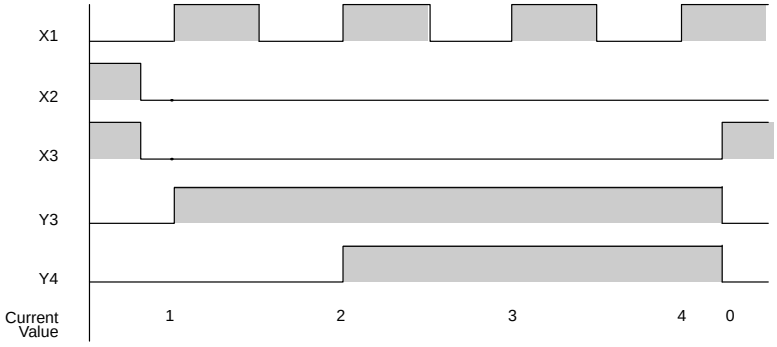
Up / Down Counter Example Using Comparative Contacts

In the following example, when X1 makes an off to on transition, counter CTA2 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)

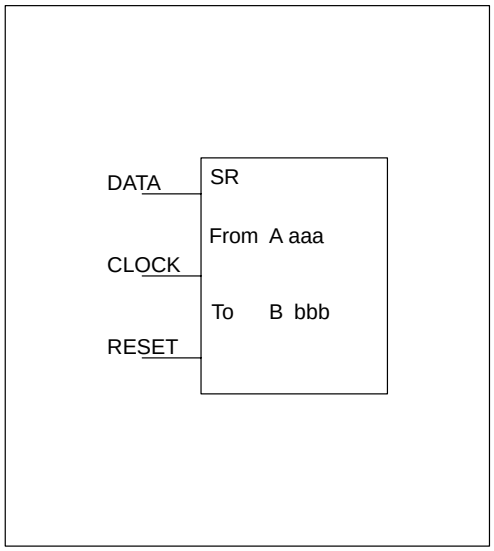
→	B 1	ENT				
GX OUT	→	D 3	ENT			
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	C 2	ENT				
GX OUT	→	E 4	ENT			

Shift Register (SR)

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 end at the end of an 8 bit boundary.

The Shift Register has three contacts.

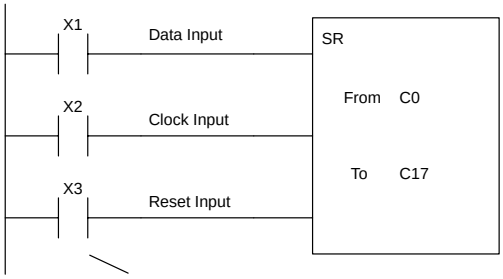
- Data — determines the value (1 or 0) that will enter the register
- Clock — shifts the bits one position on each low to high transition
- Reset —resets the Shift Register to all zeros.



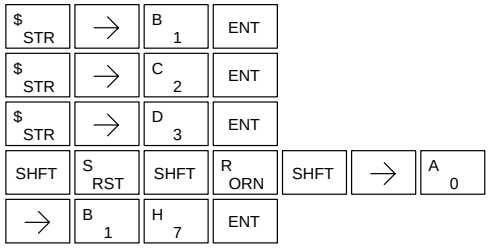
With each off to on transition of the clock input, the bits which make up the shift register block are shifted by one bit position and the status of the data input is placed into the starting bit position in the shift register. The direction of the shift depends on the entry in the From and To fields. 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		DL350 Range	
	A/B	aaa	bbb
Control Relay	C	0–1777	0–1777

DirectSOFT



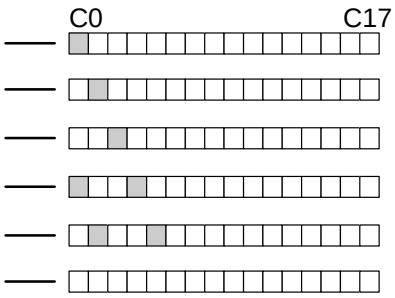
Handheld Programmer Keystrokes



Inputs on Successive Scans

Data	Clock	Reset
1	1	0
0	1	0
0	1	0
1	1	0
0	1	0
0	0	1

Shift Register Bits



■ – indicates on □ – indicates off

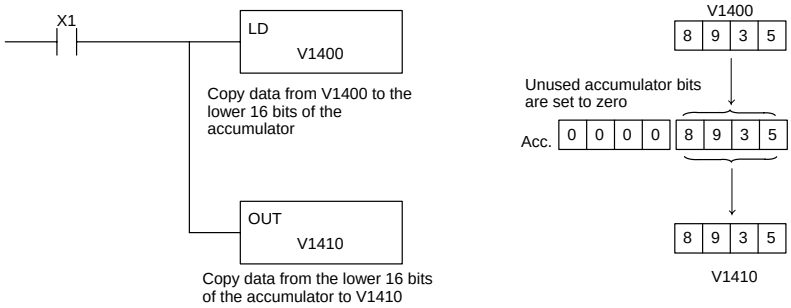
Accumulator / Stack Load and Output Data Instructions

Using the Accumulator

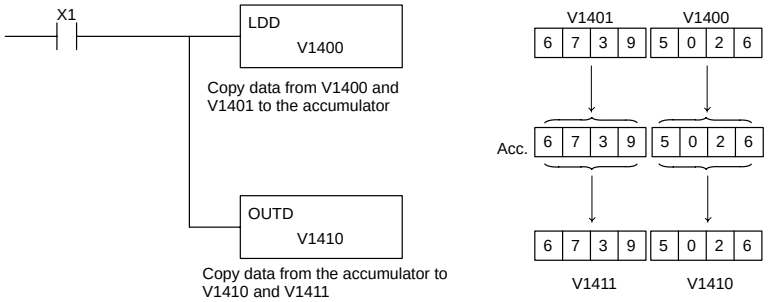
The accumulator in the DL350 CPU is a 32 bit register which is used as a temporary storage location for data that is being copied or manipulated in some manor. For example, you have to use the accumulator to perform math operations such as add, subtract, multiply, etc. Since there are 32 bits, you can use up to an 8-digit BCD number, or a 32-bit 2's complement 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 V1400 to Vmemory location V1410.

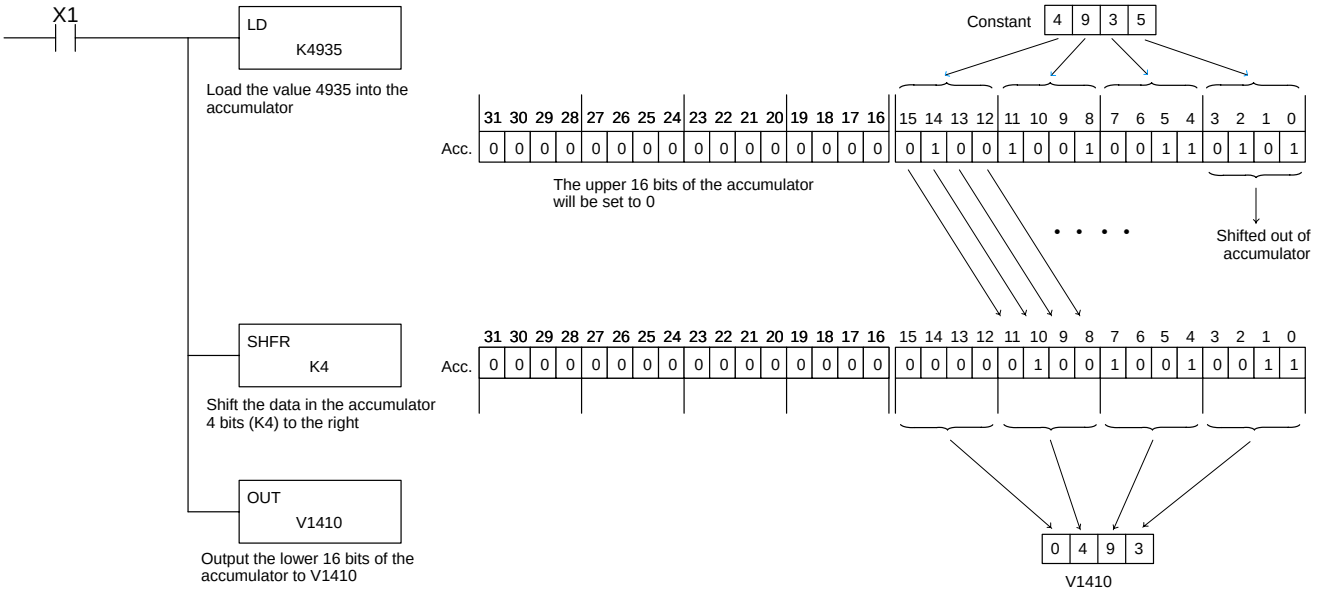


Since the accumulator is 32 bits and V memory locations are 16 bits the Load Double and Out Double (or variations thereof) use two consecutive V memory locations or 8 digit BCD constants to copy data either to the accumulator from a Vmemory address or from a Vmemory address to the accumulator. For example if you wanted to copy data from Vmemory location V1400 and V1401 to Vmemory location V1410 and V1411 the most efficient way to perform this function would be as follows:

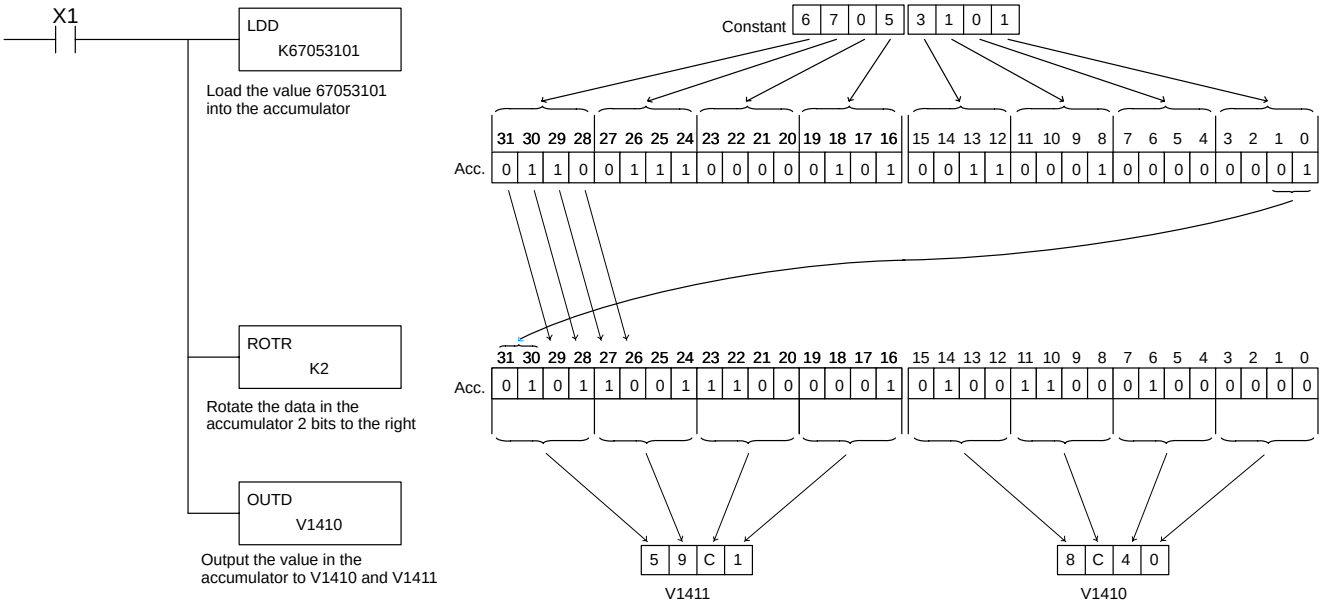


Changing the Accumulator Data

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 BCD value 4935 into the accumulator, shifts the data right 4 bits, and outputs the result to V1410.

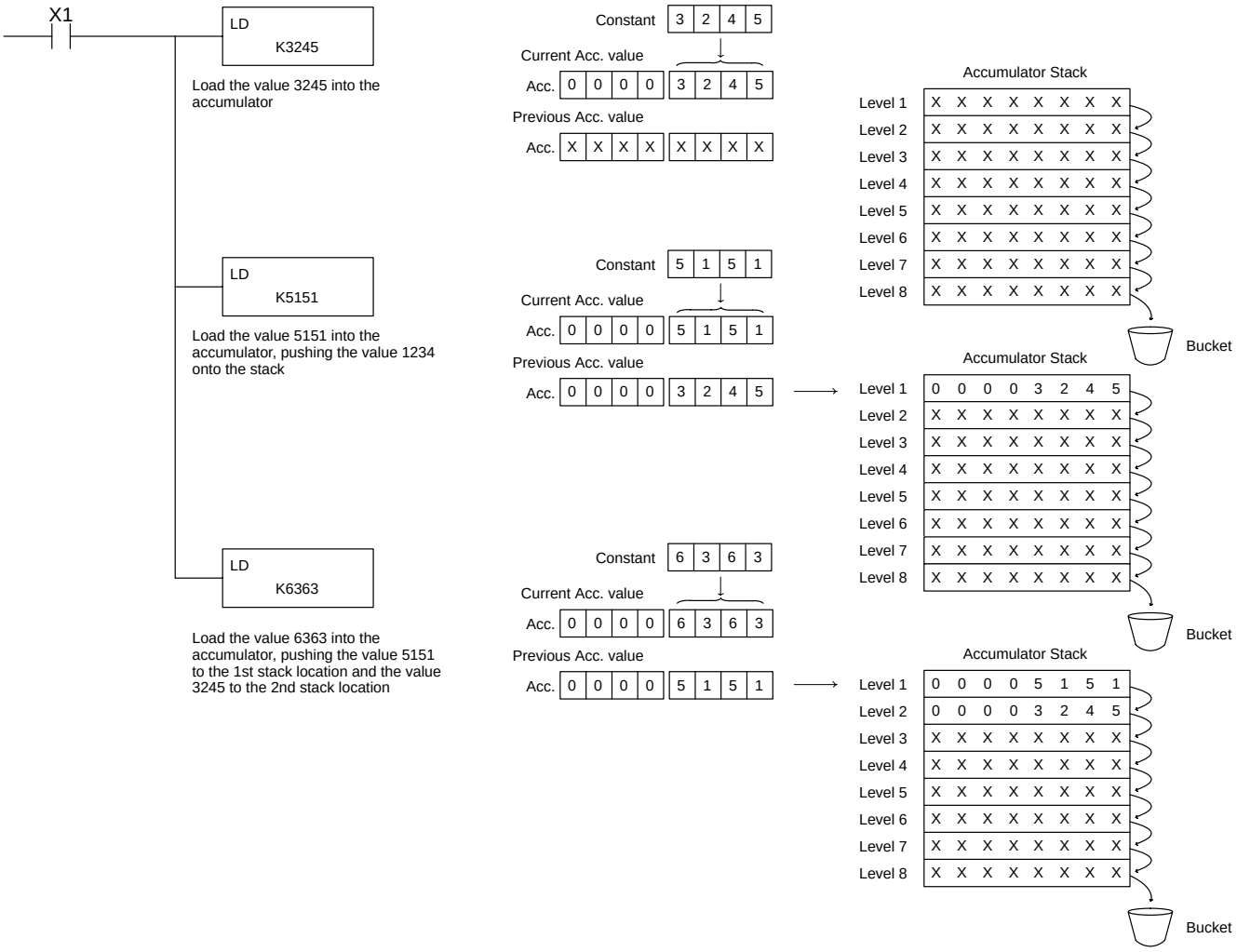


Some of the data manipulation instructions use 32 bits. They use two consecutive V memory locations or 8 digit BCD constants to manipulate data in the accumulator. The following example rotates the value 67053101 two bits to the right and outputs the value to V1410 and V1411.

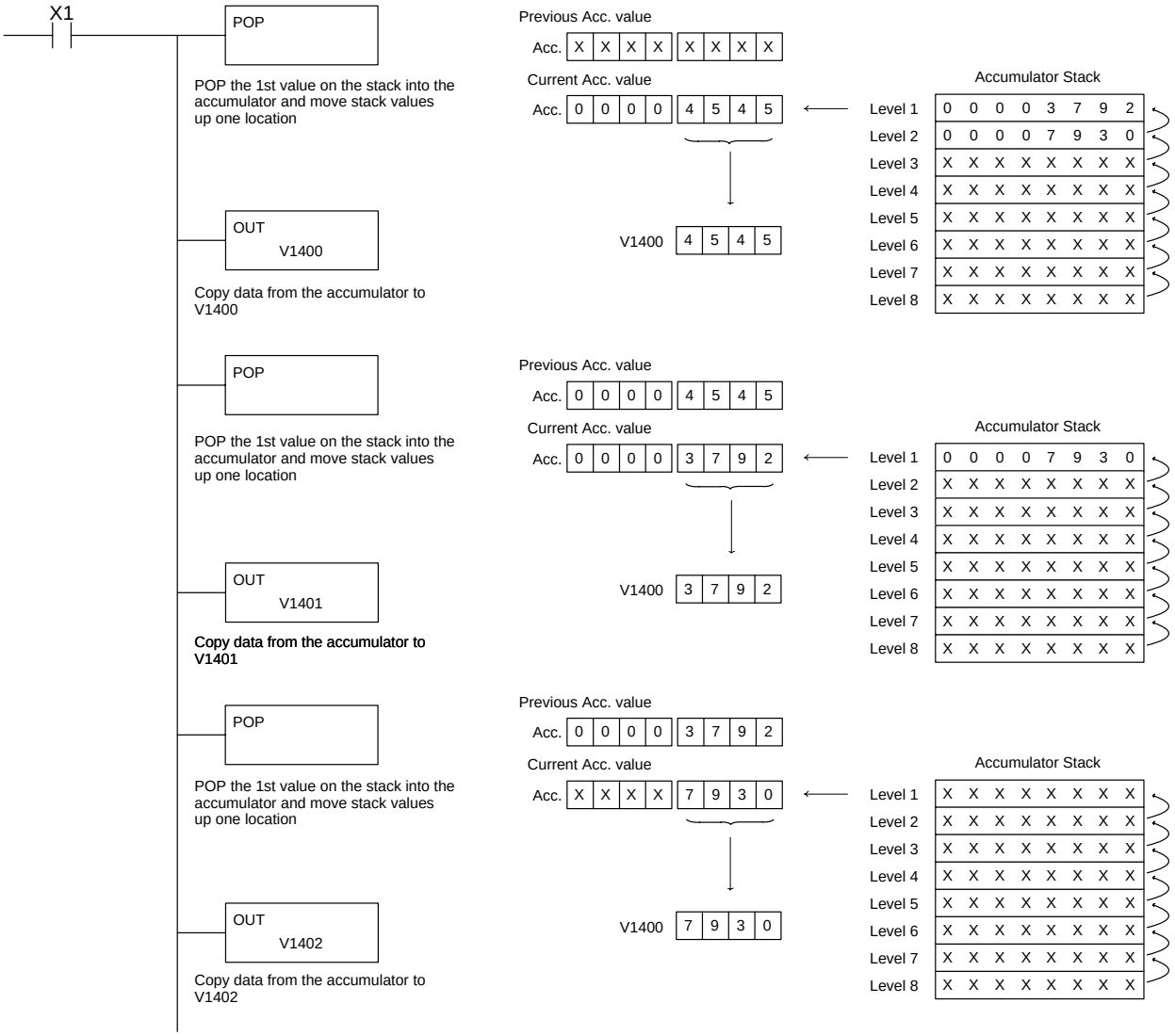


Using the Accumulator Stack

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 type instruction is executed without the use of the Out type 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.



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.

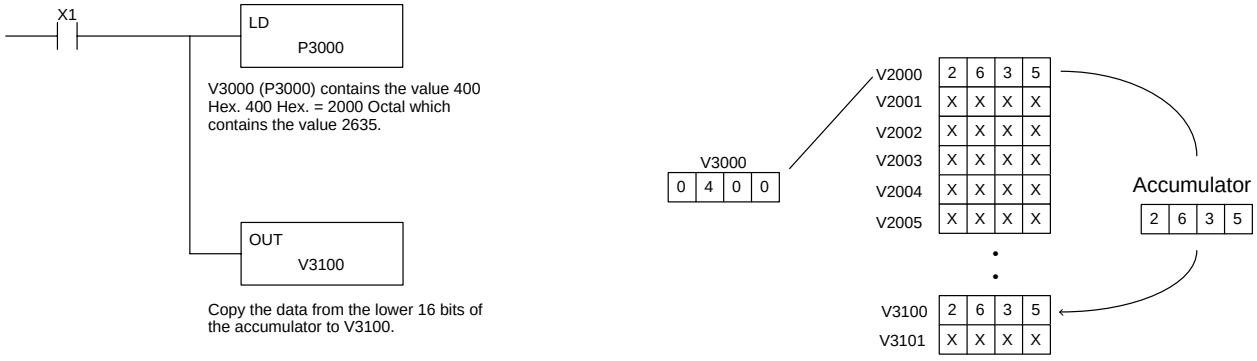

X1
POP
POP the 1st value on the stack into the accumulator and move stack values up one location
OUT
V1400
Copy data from the accumulator to V1400
POP
POP the 1st value on the stack into the accumulator and move stack values up one location
OUT
V1401
Copy data from the accumulator to V1401
POP
POP the 1st value on the stack into the accumulator and move stack values up one location
OUT
V1402
Copy data from the accumulator to V1402
Previous Acc. value
Acc. X X X X X X X X
Current Acc. value
Acc. 0 0 0 0 4 5 4 5
V1400 4 5 4 5
Accumulator Stack
Level 1 0 0 0 0 3 7 9 2
Level 2 0 0 0 0 7 9 3 0
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
Previous Acc. value
Acc. 0 0 0 0 4 5 4 5
Current Acc. value
Acc. 0 0 0 0 3 7 9 2
V1400 3 7 9 2
Accumulator Stack
Level 1 0 0 0 0 7 9 3 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
Previous Acc. value
Acc. 0 0 0 0 3 7 9 2
Current Acc. value
Acc. X X X X 7 9 3 0
V1400 7 9 3 0
Accumulator Stack
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

Using Pointers

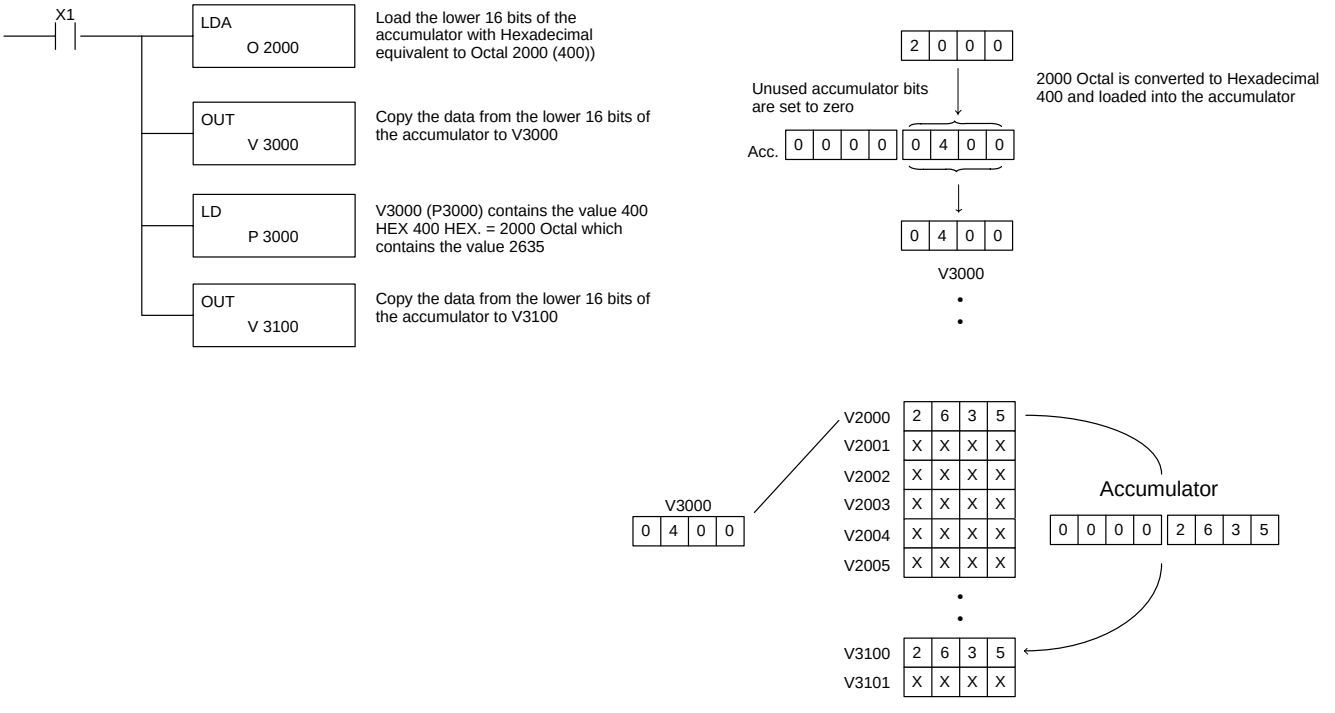
Many of the instructions will allow Vmemory pointers as a operand. Pointers can be useful in ladder logic programming, but can be difficult to understand or implement in your application if you do not have prior experience with pointers (commonly known as indirect addressing). Pointers allow instructions to obtain data from Vmemory locations referenced by the pointer value.

NOTE: V-memory addressing is in octal. However the value in the pointer location which will reference a V-memory location is viewed as HEX. Use the Load Address instruction to move a address into the pointer location. This instruction performs the Octal to Hexadecimal conversion for you.

The following example uses a pointer operand in a Load instruction. V-memory location 3000 is the pointer location. V3000 contains the value 400 which is the HEX equivalent of the Octal address V-memory location V2000. The CPU copies the data from V2000 into the lower word of the accumulator.

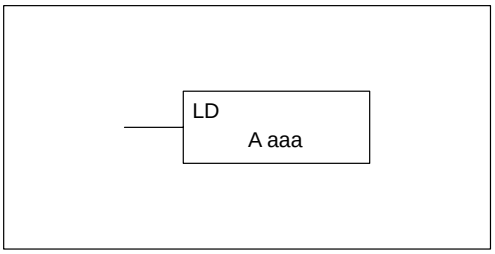


The following example is similar to the one above, except for the LDA (load address) instruction which automatically converts the Octal address to the Hex equivalent.



Load (LD)

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.



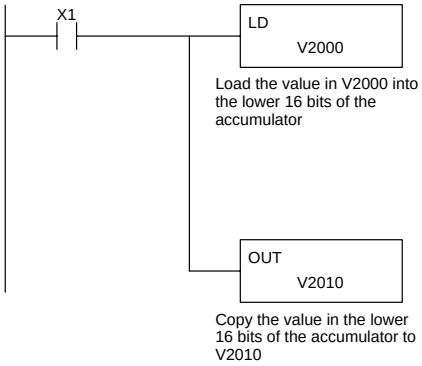
Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	0–FFFF

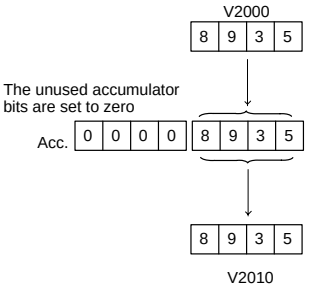
Discrete Bit Flags	Description
SP76	on when the value loaded into the accumulator by any instruction is zero.

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.

DirectSOFT



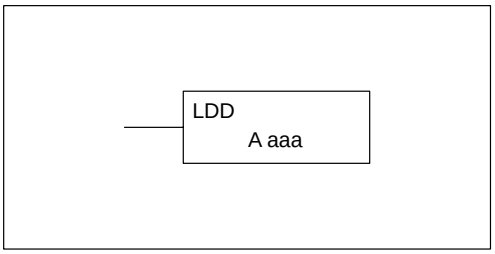


Handheld Programmer Keystrokes

$ 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

Load Double (LDD)

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.



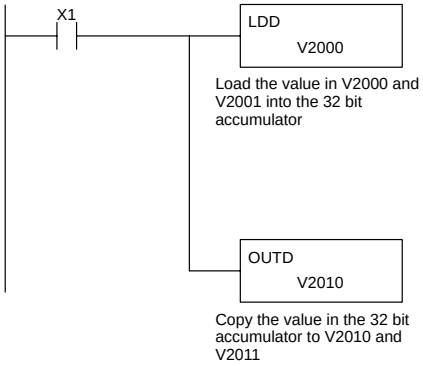
Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	0–FFFF

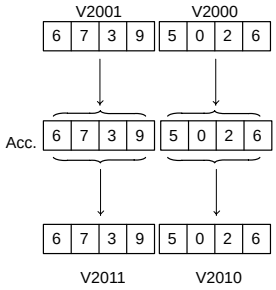
Discrete Bit Flags	Description
SP76	on when the value loaded into the accumulator by any instruction is zero.

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.

DirectSOFT



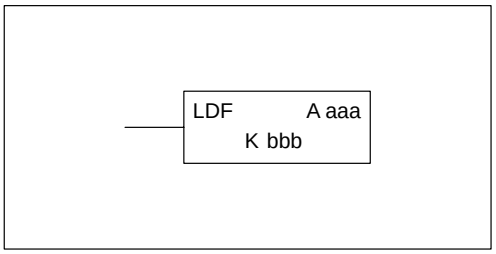


Handheld Programmer Keystrokes

$ 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)

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.



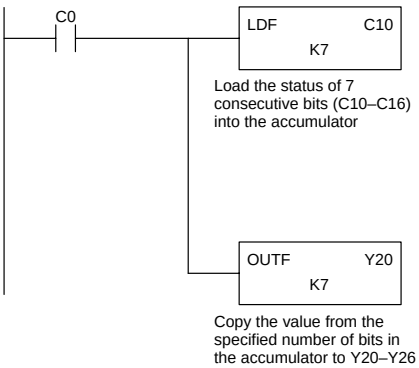
Operand Data Type		DL350 Range	
	A	**aaa**	**bbb**
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0–777	—
Constant	K	—	1–32

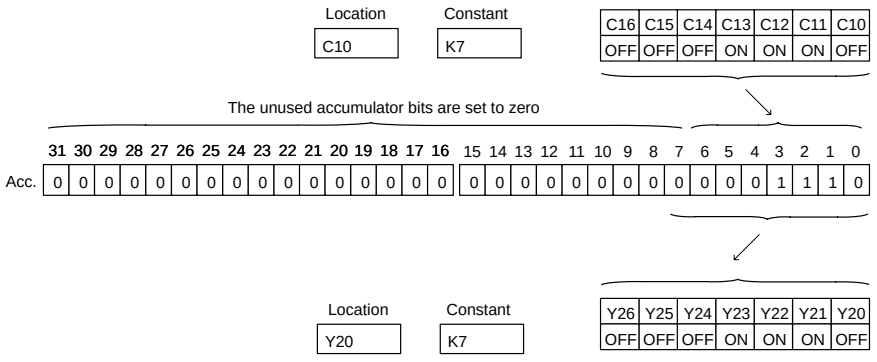
Discrete Bit Flags	Description
SP76	on when the value loaded into the accumulator by any instruction is zero.

NOTE: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.

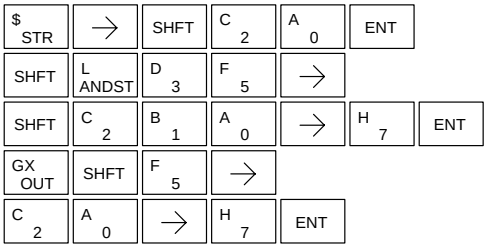
In the following example, when C0 is on, the binary pattern of C10–C16 (7 bits) will be loaded into the accumulator using the Load Formatted instruction. The lower 6 bits of the accumulator are output to Y20–Y26 using the Out Formatted instruction.

DirectSOFT



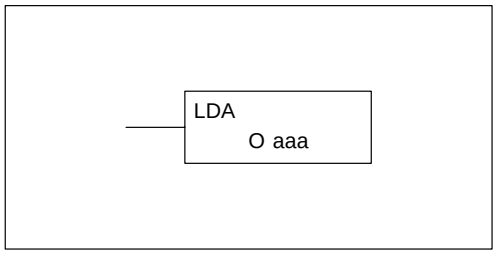


Handheld Programmer Keystrokes



Load Address (LDA)

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 system are in octal.



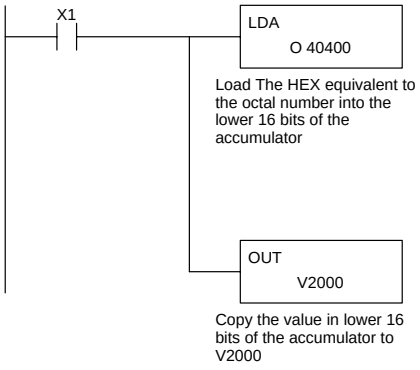
Operand Data Type		DL350 Range
		aaa
Octal Address	O	All V mem. (See page 3–29)

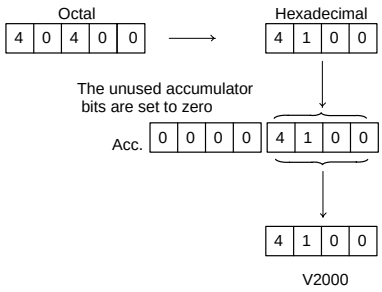
Discrete Bit Flags	Description
SP76	on when the value loaded into the accumulator by any instruction is zero.

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.

DirectSOFT



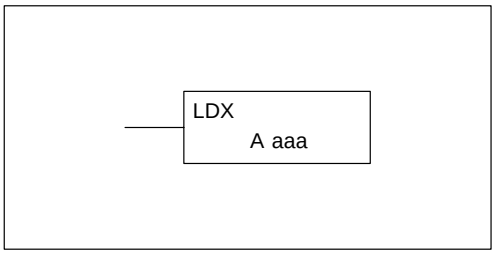


Handheld Programmer Keystrokes

$ 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)

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.

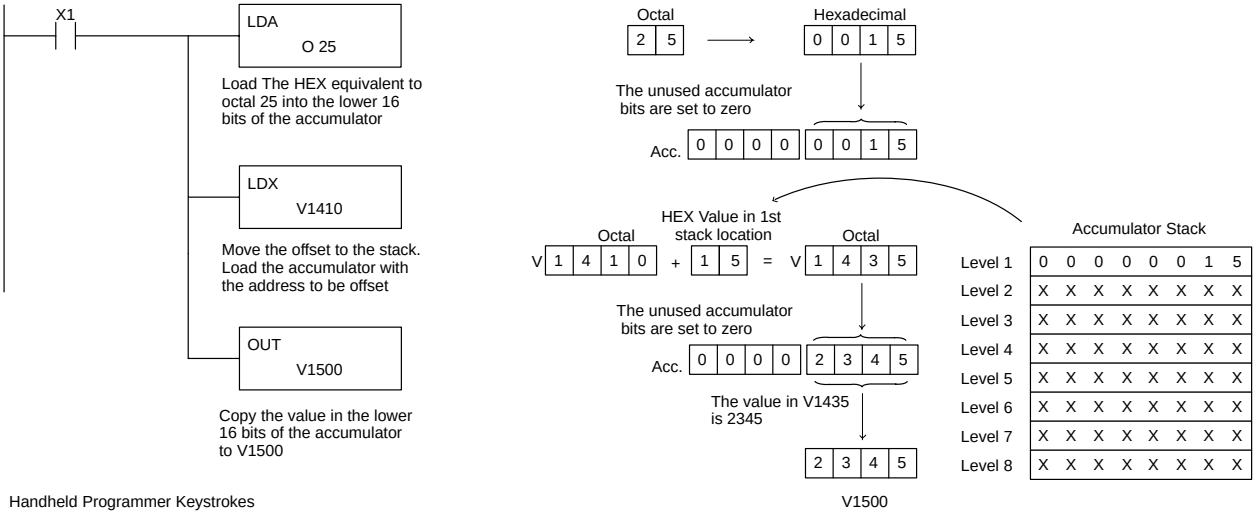


Helpful Hint: — The Load Address instruction can be used to convert an octal address to a HEX address and load the value into the accumulator.

Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)

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 1st. 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.

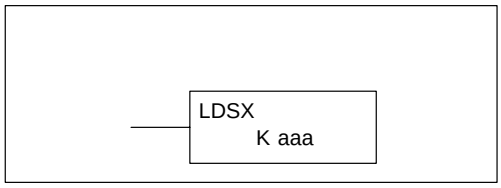


Handheld Programmer Keystrokes

STR	→	X	1	ENT								
SHFT	L	D	A	→	O	2	5	ENT				
SHFT	L	D	X	→	SHFT	V	1	4	1	0	ENT	
OUT	→	V	1	5	0	0	ENT					

Load Accumulator Indexed from Data Constants (LDSX)

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.



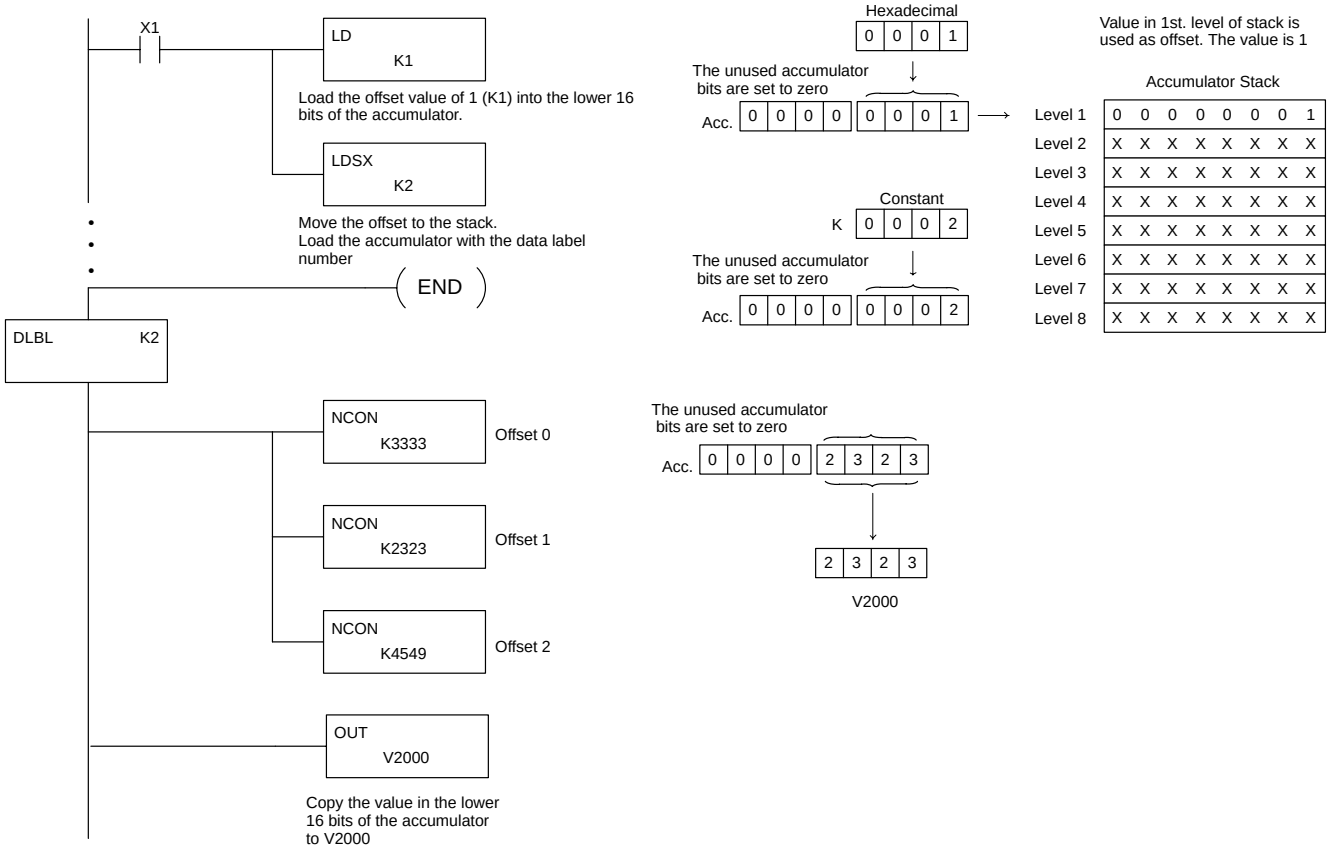
The LDSX instruction uses the value in the first level of the accumulator stack as an offset to determine which numerical or ASCII constant within the Data Label Area will be loaded into the accumulator. The LDSX instruction interprets the value in the first level of the accumulator stack as a HEX value.

Helpful Hint: — The Load Address instruction can be used to convert octal to HEX and load the value into the accumulator.

Operand Data Type		DL350 Range
		aaa
Constant	K	1–FFFF

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.

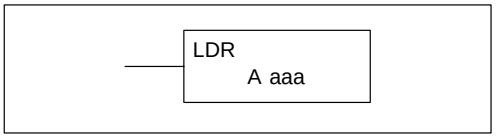


Handheld Programmer Keystrokes

$ 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		

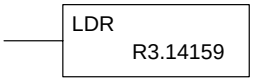
Load Real Number (LDR)

The Load Real Number instruction loads a real number contained in two consecutive V-memory locations, or an 8-digit constant into the accumulator.

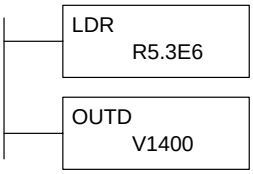


Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All V mem (See p. 3–29)
Pointer	P	All V mem (See p. 3–29)
Real Constant	R	Full IEEE 32-bit range

DirectSOFT 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”.

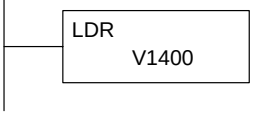


For very large numbers or very small numbers, you can use exponential notation. The number to the right is 5.3 million. The OUTD instruction stores it in V1400 and V1401.



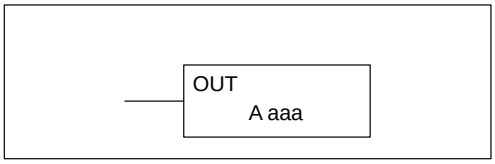
These real numbersare in the IEEE 32-bit floating point format. They occupy two V-memory locations, *regardless of how big or small the number may be!* If you view a stored real number in hex, binary, or even BCD, the number shown will be very difficult to decipher. Like all other number types, you must 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 retreive 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)

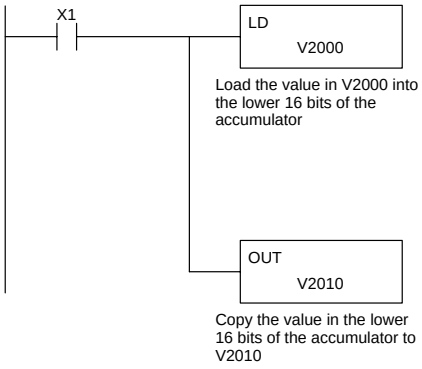
The Out instruction is a 16 bit instruction that copies the value in the lower 16 bits of the accumulator to a specified V memory location (Aaaa).

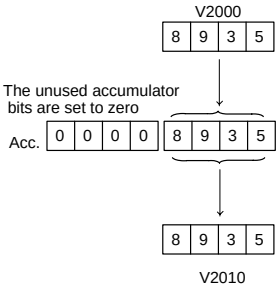


Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

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



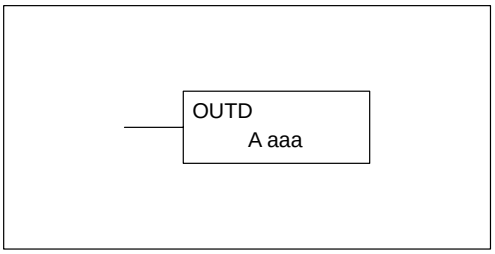


Handheld Programmer Keystrokes

$ 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)

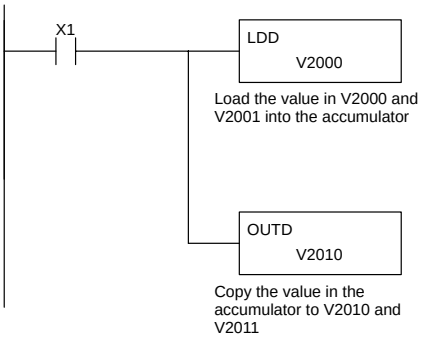
The Out Double instruction is a 32 bit instruction that copies the value in the accumulator to two consecutive V memory locations at a specified starting location (Aaaa).

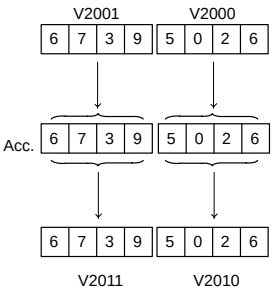


Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

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 Out Double instruction.

DirectSOFT



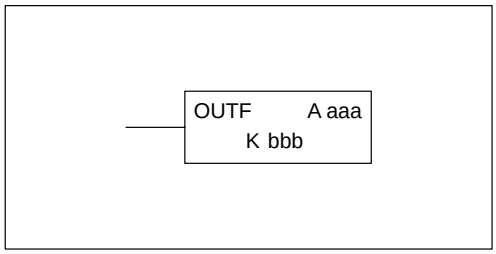


Handheld Programmer Keystrokes

$ 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

Out Formatted (OUTF)

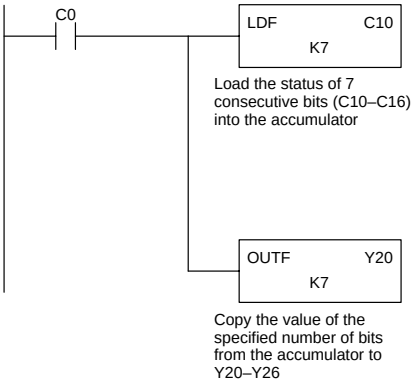
The Out Formatted instruction outputs 1–32 bits from the accumulator to the specified discrete memory locations. The instruction requires a starting location (Aaaa) for the destination and the number of bits (Kbbb) to be output.

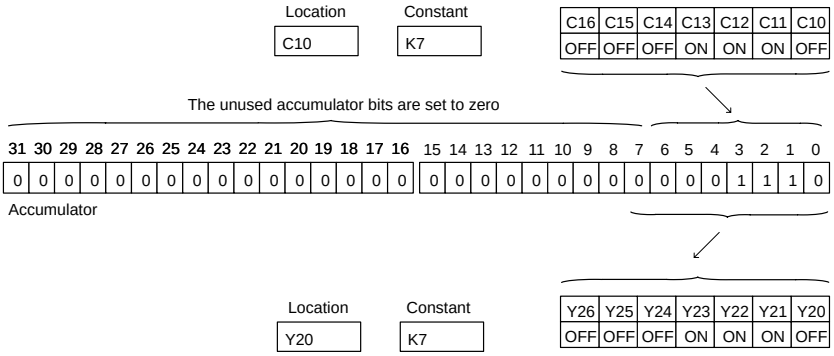


Operand Data Type		DL350 Range	
	A	**aaa**	**bbb**
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Constant	K	—	1–32

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 Y20–Y26 using the Out Formatted instruction.

DirectSOFT



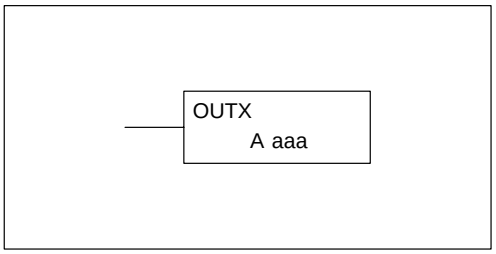


Handheld Programmer Keystrokes

$ 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	→		
C 2	A 0	→	H 7	ENT	

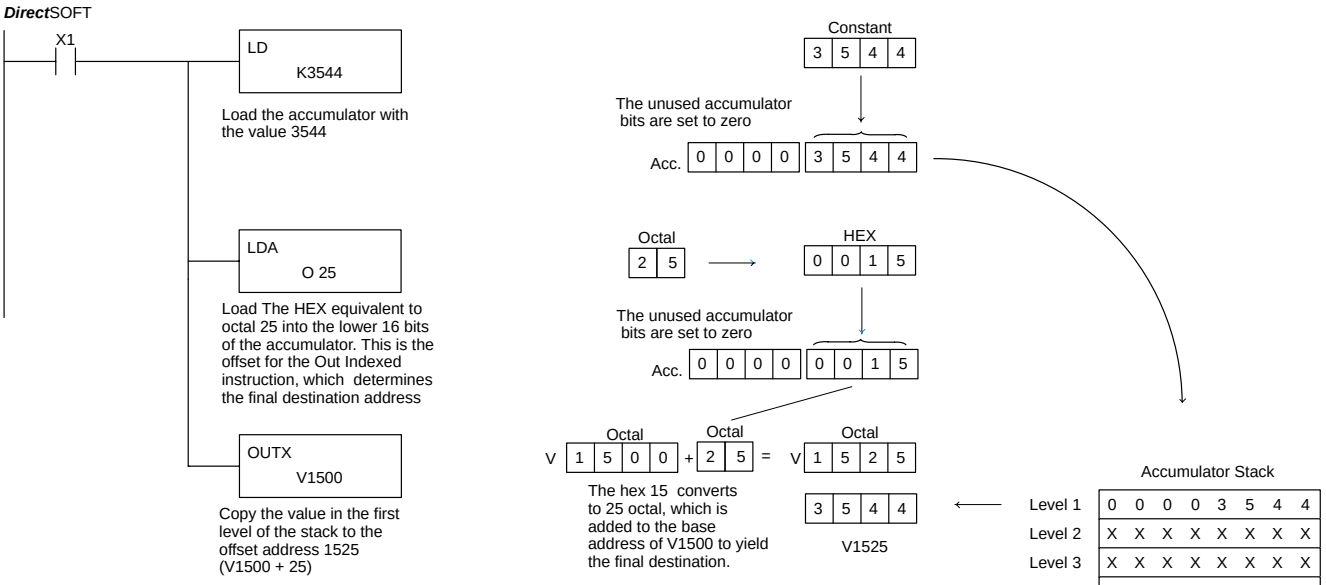
Out Indexed (OUTX)

The Out Indexed 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		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)

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 Load Address instruction is executed. Remember, two consecutive Load instructions places the value of the first load instruction onto the stack. The Load Address instruction converts octal 25 to HEX 15 and places the value in the accumulator. The Out Indexed instruction outputs the value 3544 which resides in the first level of the accumulator stack to V1525.

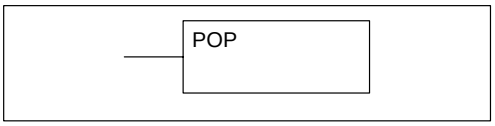


Handheld Programmer Keystrokes

STR	→	1	ENT					
SHFT	L	D	3	5	4	4	ENT	
SHFT	L	D	A	→	O	2	5	ENT
OUT	SHFT	X	→	V	1	5	0	0	ENT

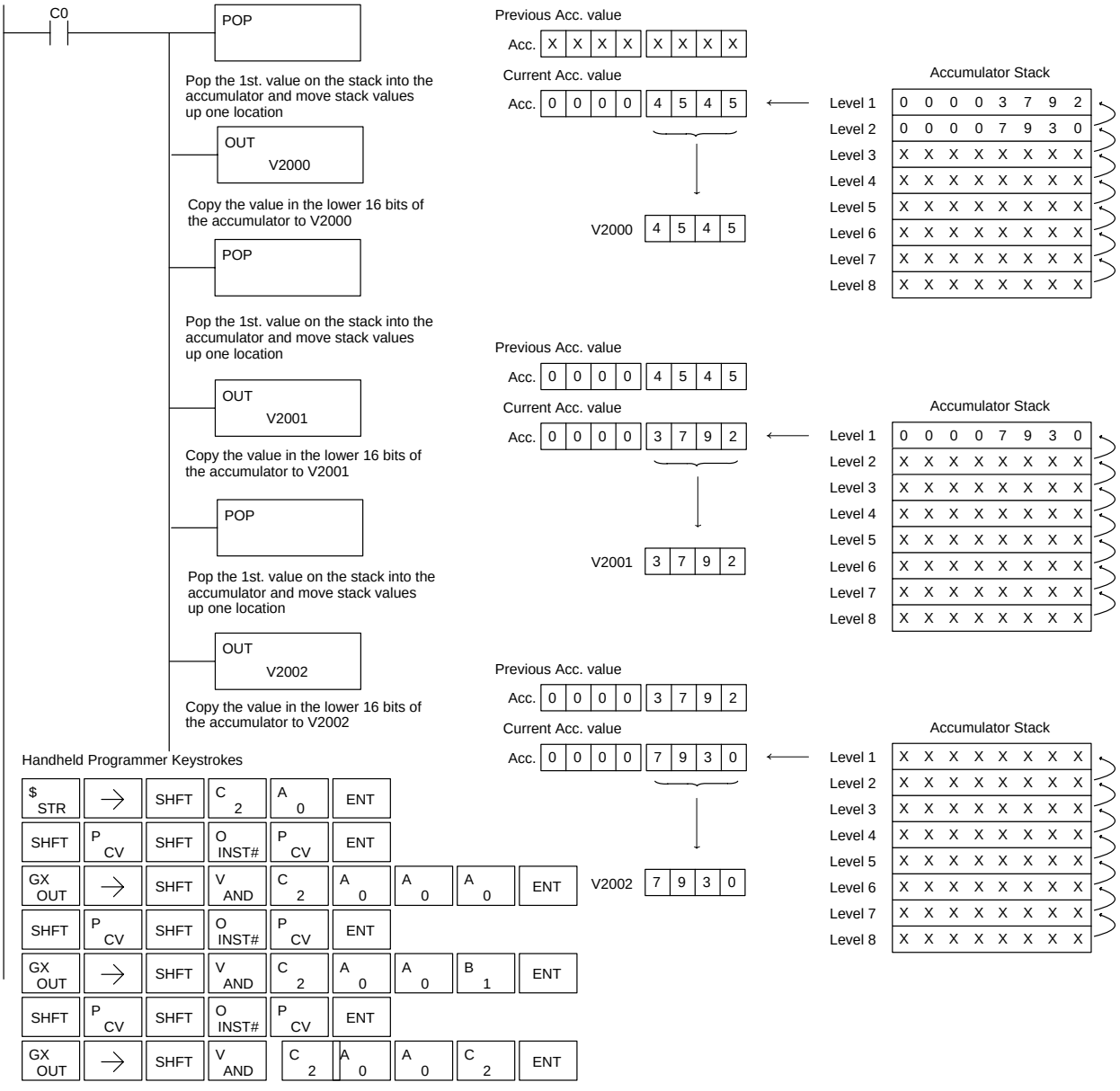
Pop (POP)

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.



In the example, 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 Out Double instruction would be used and 2 V memory locations for each Out Double need to be allocated.

Discrete Bit Flags	Description
SP63	on when the result of the instruction causes the value in the accumulator to be zero.



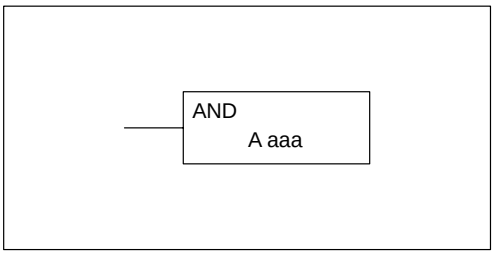
Handheld Programmer Keystrokes

$ 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	

Accumulator Logical Instructions

And (AND)

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.



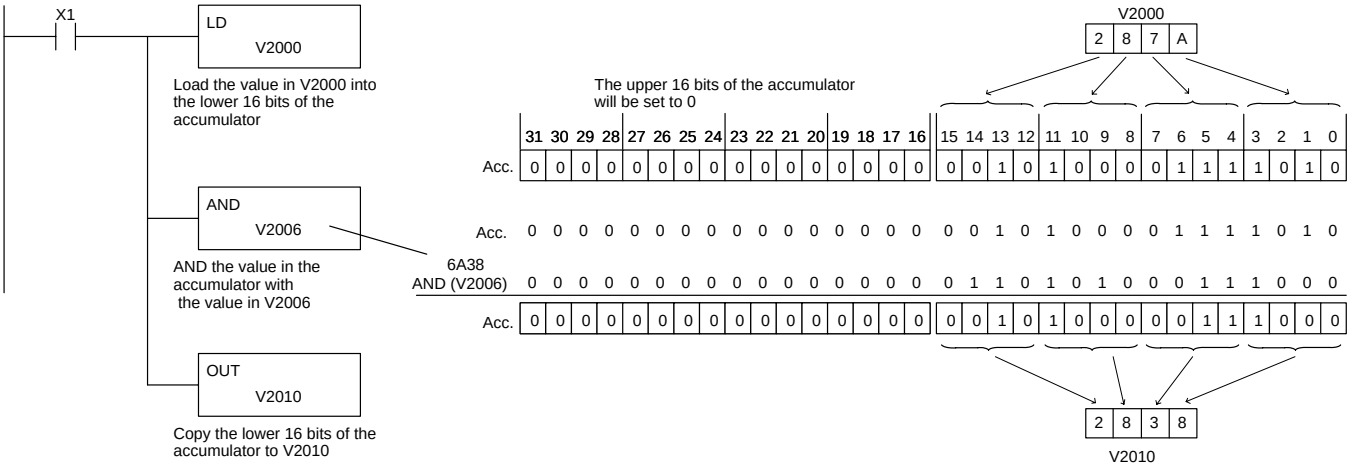
Operand Data Type		DL3540 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero

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 will be loaded into the accumulator using the Load 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.

DirectSOFT

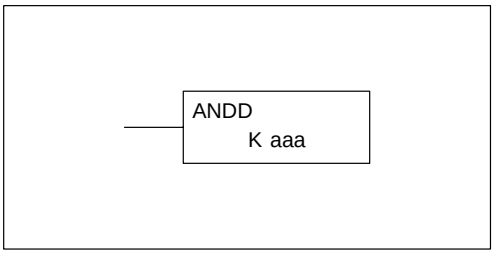


Handheld Programmer Keystrokes

$ 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)

The And Double is a 32 bit instruction that logically ands the value in the accumulator with an 8 digit (max.) constant value (Aaaa). The result resides in the accumulator. Discrete status flags indicate if the result of the And Double is zero or a negative number (the most significant bit is on).



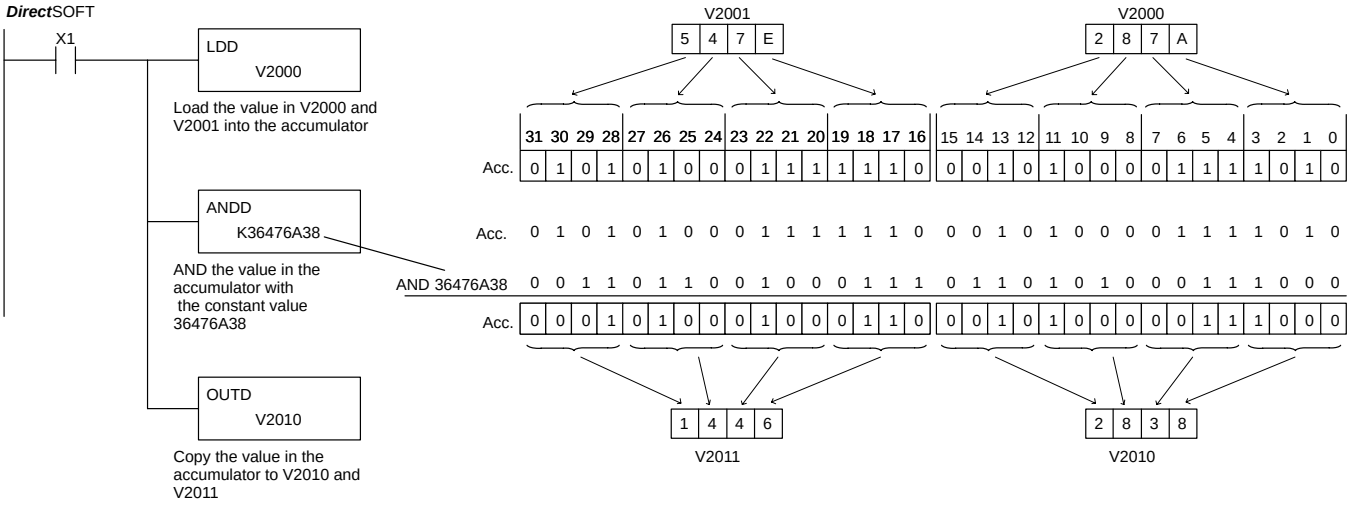
Operand Data Type		DL350 Range
		aaa
Constant	K	0–FFFF

Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	Will be on is the result in the accumulator is negative

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 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is anded with 36476A38 using the And double instruction. The value in the accumulator is output to V2010 and V2011 using the Out Double instruction.

***Direct*SOFT**

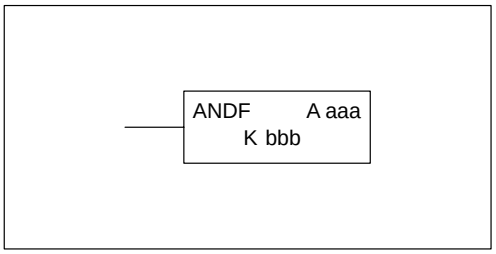


Handheld Programmer Keystrokes

$ 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)

The And Formatted instruction logically ANDs the binary value in the accumulator and 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).



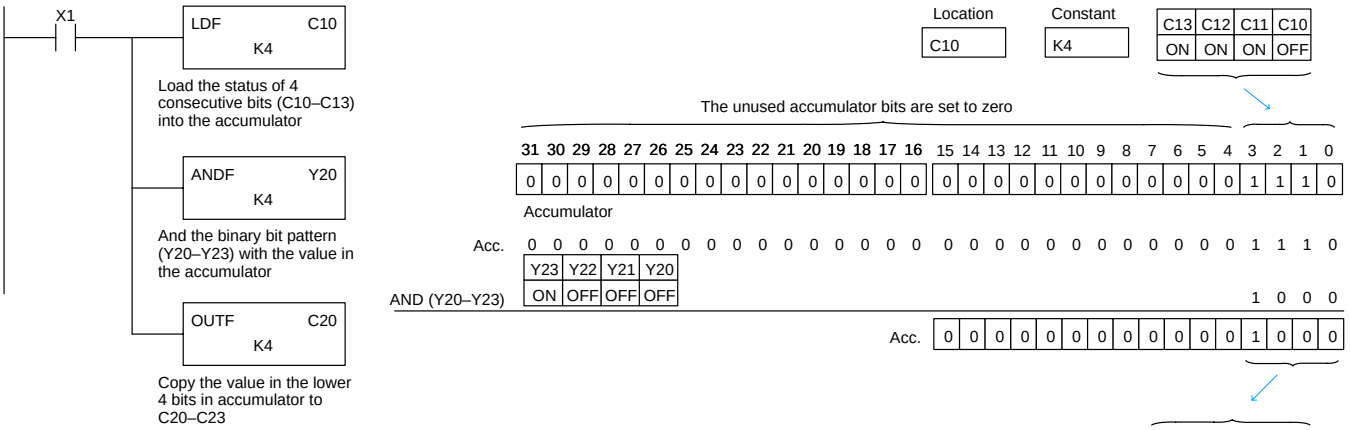
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0–777	—
Constant	K	—	1–32

Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	Will be on is 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 Load Formatted 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 And Formatted instruction. The Out Formatted instruction outputs the accumulator's lower four bits to C20–C23.

DirectSOFT

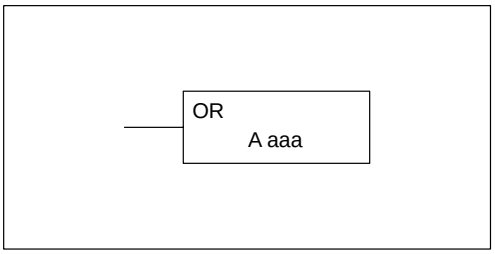


Handheld Programmer Keystrokes

STR	→	1	ENT								
SHFT	L	D	F	→	C	1	0	→	K	4	ENT
AND	SHFT	F	→	Y	2	0	→	K	4	ENT	
OUT	SHFT	F	→	C	2	0	→	K	4	ENT	

Or (OR)

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.



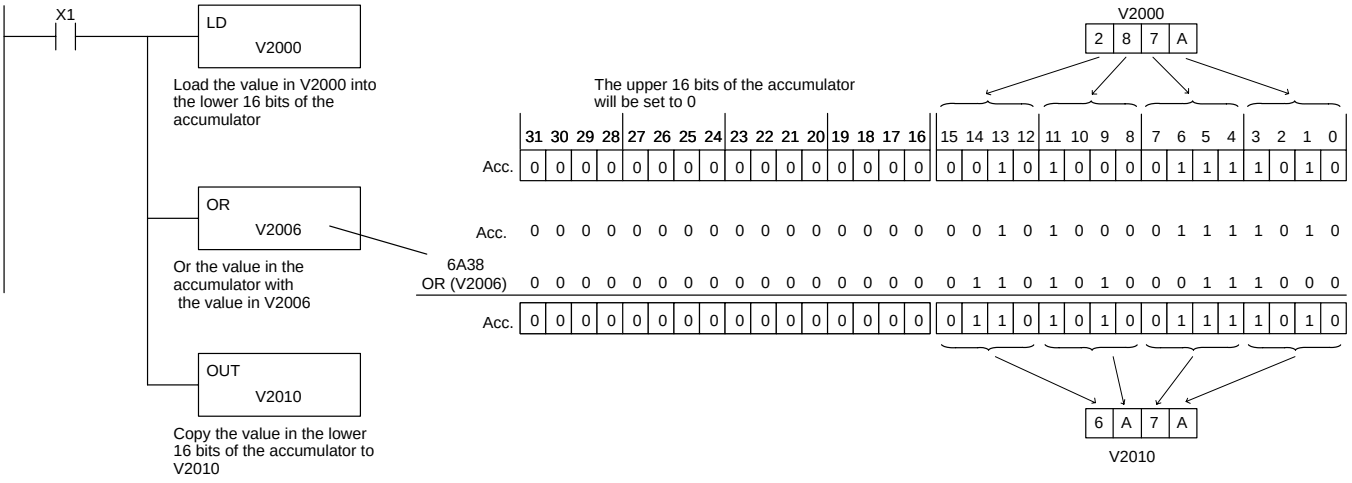
Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero

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 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 are output to V2010 using the Out instruction.

***Direct*SOFT**

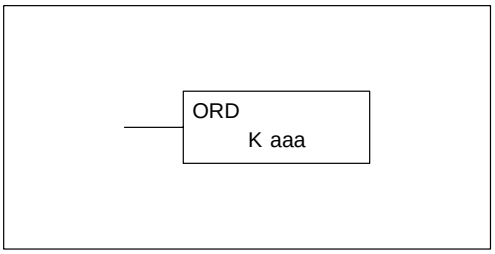


Handheld Programmer Keystrokes

$ 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)

The Or Double is a 32 bit instruction that ors the value in the accumulator with an 8 digit (max.) constant value. The result resides in the accumulator. Discrete status flags indicate if the result of the Or Double is zero or a negative number (the most significant bit is on).



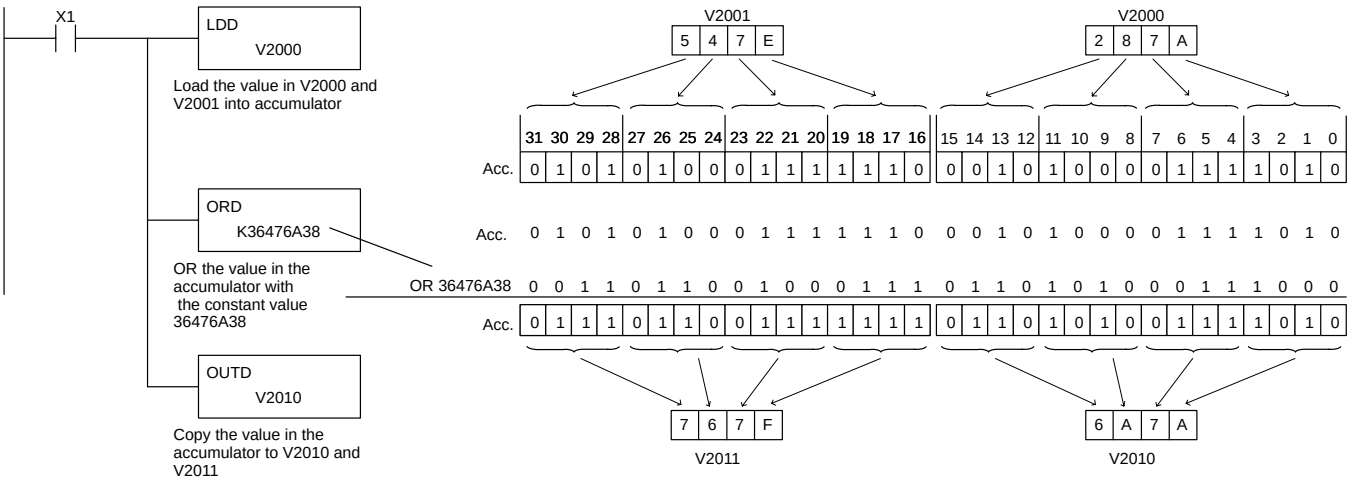
Operand Data Type		DL350 Range
	A	**aaa**
Constant	K	0–FFFF

Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	Will be on is the result in the accumulator is negative

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 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is ored with 36476A38 using the Or Double instruction. The value in the accumulator is output to V2010 and V2011 using the Out Double instruction.

***Direct*SOFT**

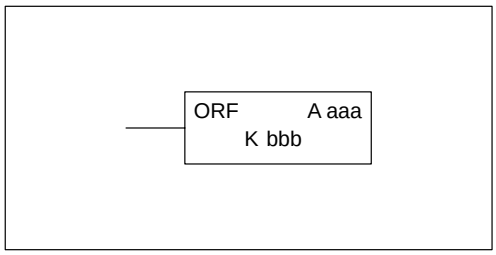


Handheld Programmer Keystrokes

$ 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)

The Or Formatted 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).

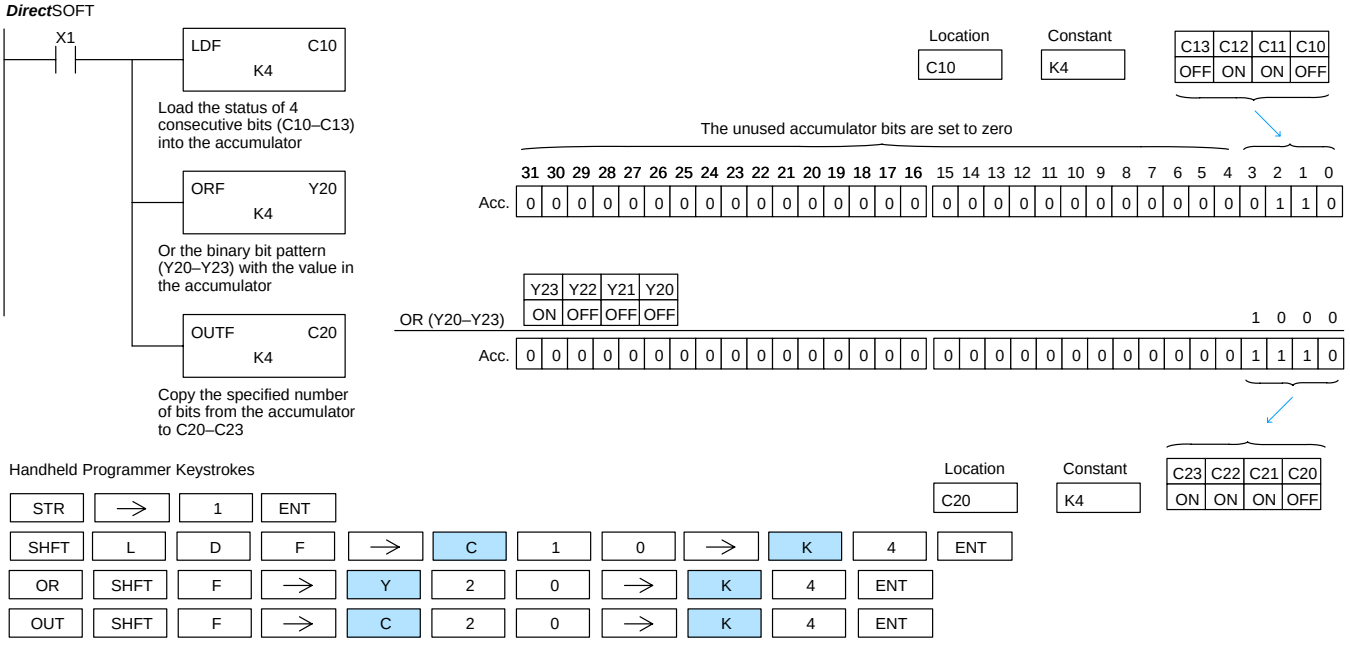


Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0–777	—
Constant	K	—	1–32

Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	Will be on is the result in the accumulator is negative

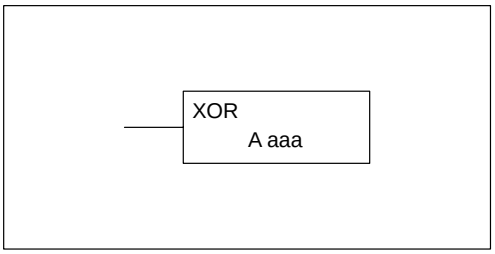
NOTE: Status flags are valid only until another instruction uses the same flag.

In the following example, when X1 is on the Load Formatted instruction loads C10–C13 (4 binary bits) into the accumulator. The Or Formatted instruction logically ORs the accumulator contents with Y20–Y23 bit pattern. The Out Formatted instruction outputs the accumulator's lower four bits to C20–C23.



Exclusive Or (XOR)

The Exclusive Or 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.



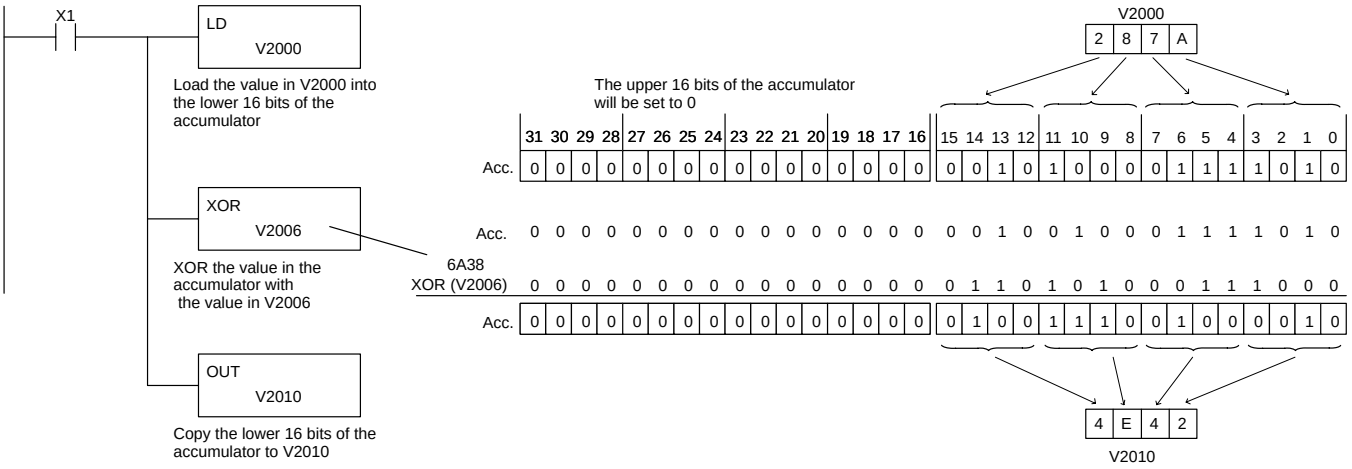
Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero

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 will be loaded into the accumulator using the Load instruction. The value in the accumulator is exclusive ored with V2006 using the Exclusive Or instruction. The value in the lower 16 bits of the accumulator are output to V2010 using the Out instruction.

***Direct*SOFT**

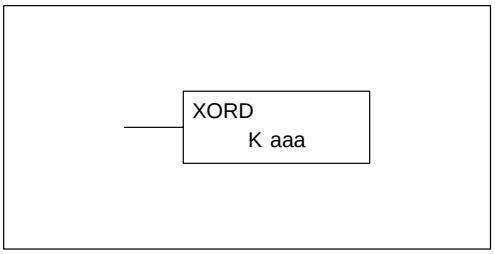


Handheld Programmer Keystrokes

$ 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)

The Exclusive OR Double is a 32 bit instruction that performs an exclusive or of the value in the accumulator and the value (Aaaa), which is a 8 digit (max.) constant. The result resides in the accumulator. Discrete status flags indicate if the result of the Exclusive Or Double is zero or a negative number (the most significant bit is on).

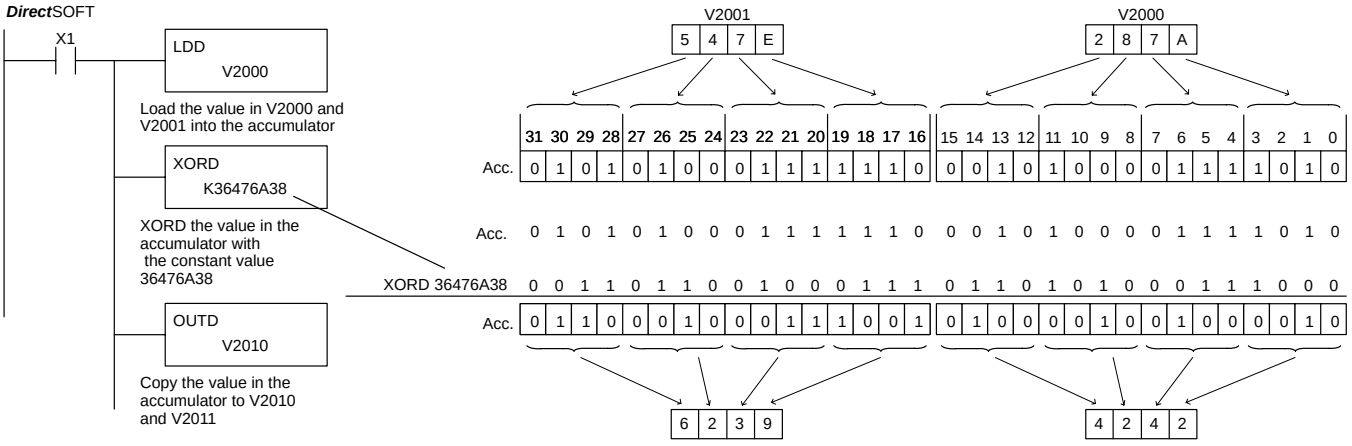


Operand Data Type		DL350 Range
	A	**aaa**
Constant	K	0–FFFF

Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	Will be on is the result in the accumulator is negative

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 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is exclusively ored with 36476A38 using the Exclusive Or Double instruction. The value in the accumulator is output to V2010 and V2011 using the Out Double instruction.

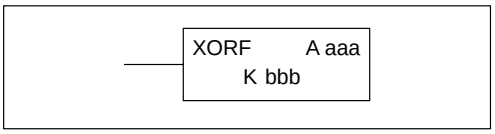


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	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)

The Exclusive Or Formatted instruction performs an exclusive OR of the binary value in the accumulator and a specified range of discrete memory bits (1–32).



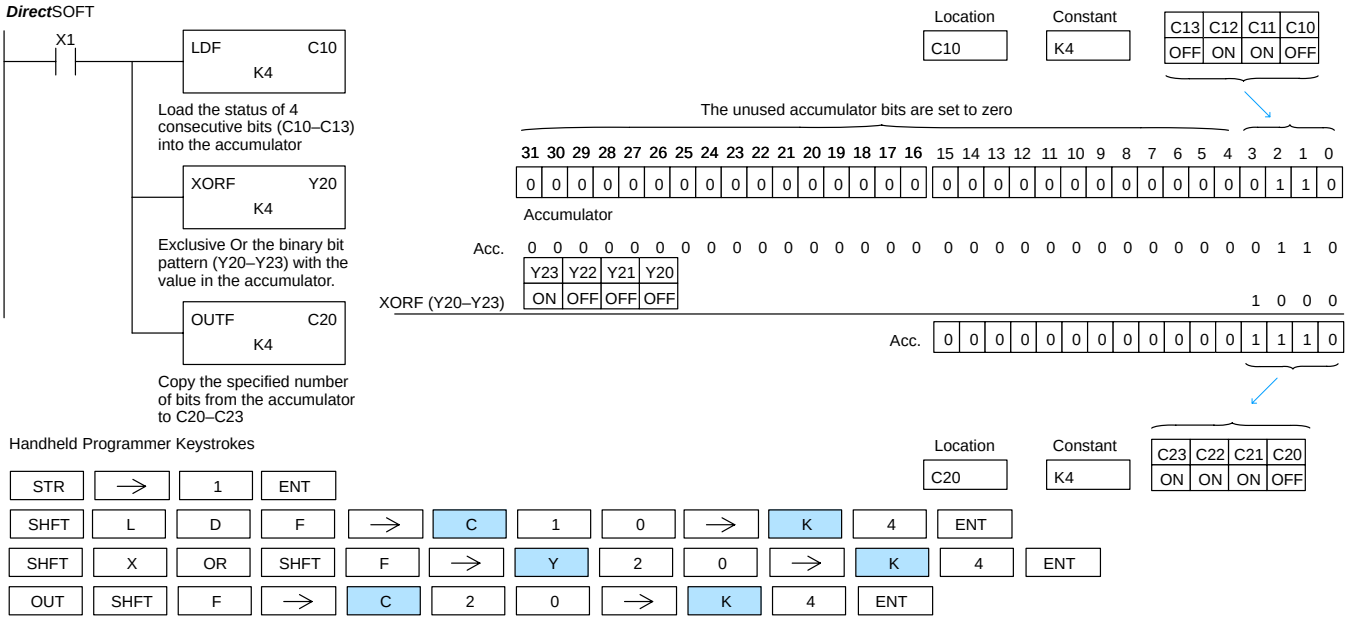
The instruction requires a starting location (Aaaa) and the number of bits (Bbbb) to be exclusive ORed. Discrete status flags indicate if the result of the Exclusive Or Formatted is zero or negative (the most significant bit =1).

Operand Data Type		DL350 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	Will be on if the result in the accumulator is zero
SP70	Will be on is the result in the accumulator is negative

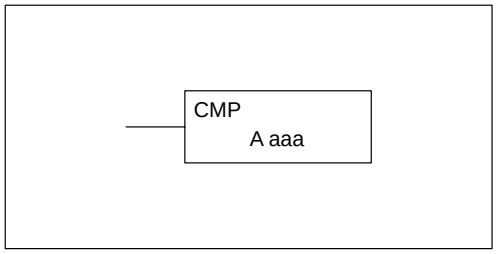
NOTE: Status flags are valid only until another instruction uses the same flag.

In the following example, when X1 is on, the binary pattern of C10–C13 (4 bits) will be loaded into the accumulator using the Load Formatted instruction. The value in the accumulator will be logically Exclusive Ored with the bit pattern from Y20–Y23 using the Exclusive Or Formatted instruction. The value in the lower 4 bits of the accumulator are output to C20–C23 using the Out Formatted instruction.



Compare (CMP)

The compare 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.



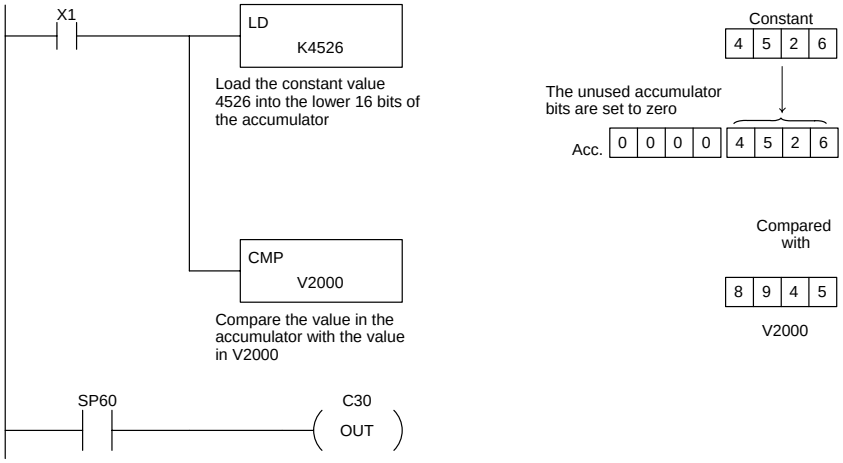
Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

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: The status flags are updated immediately after the instruction is carried out during the scan of the CPU, therefore, it is only valid until another instruction that uses the same flags is executed.

In the following example when X1 is on, the constant 4526 will be loaded into the lower 16 bits of the accumulator using the Load instruction. The value in the accumulator is compared with the value in V2000 using the Compare 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.

DirectSOFT

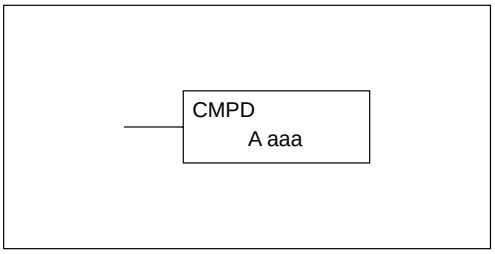


Handheld Programmer Keystrokes

$ 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)

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.

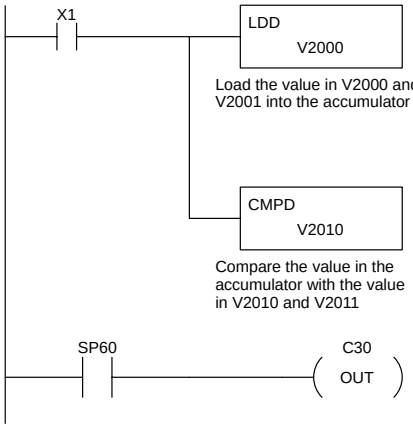


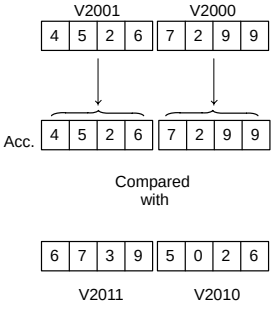
Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	1–FFFFFFFF

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: The status flags are updated immediately after the instruction is carried out during the scan of the CPU, therefore, it is only valid until another instruction that uses the same flags is executed.

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.



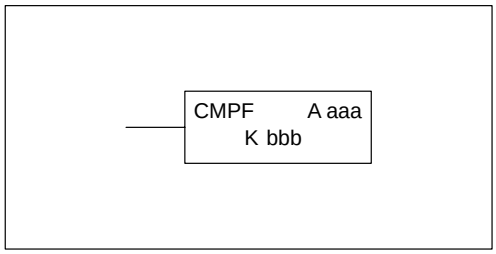


Handheld Programmer Keystrokes

$ 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)

The Compare Formatted 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.



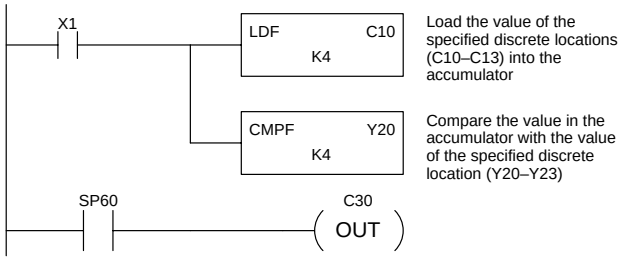
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0–777	—
Constant	K	—	1–32

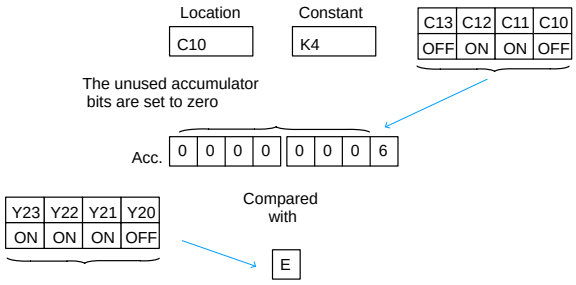
Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.

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.

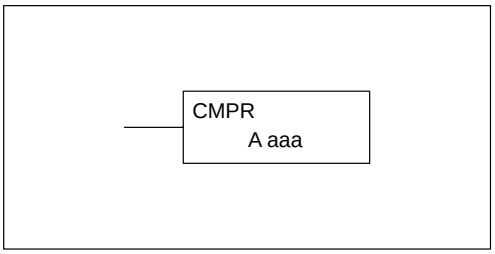
***Direct*SOFT**





Compare Real Number (CMPR)

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.



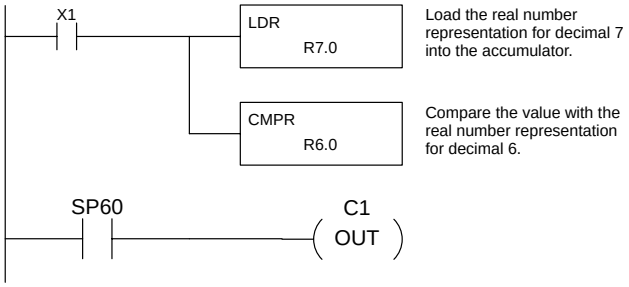
Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)
Constant	R	–3.402823E+038 to + –3.402823E+038

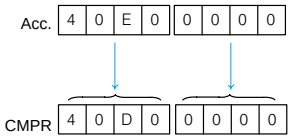
Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP75	On when a real number instruction is executed and a non–real 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 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 SP60).

***Direct*SOFT**

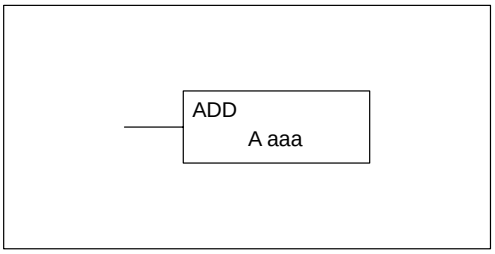




Math Instructions

Add (ADD)

Add is a 16 bit instruction that adds a BCD value in the accumulator with a BCD value in a V memory location (Aaaa). The result resides in the accumulator. You cannot use a constant as the parameter in the box.

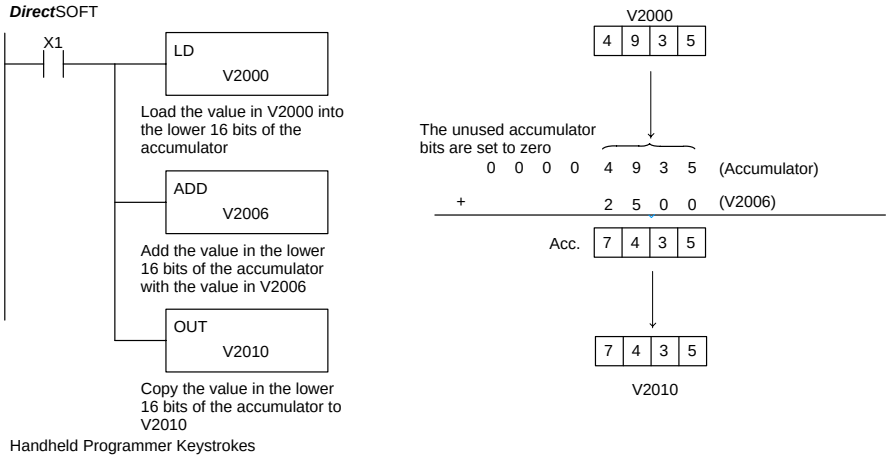


Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

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: 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 will be loaded into the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator are added to the value in V2006 using the Add instruction. The value in the accumulator is copied to V2010 using the Out instruction.

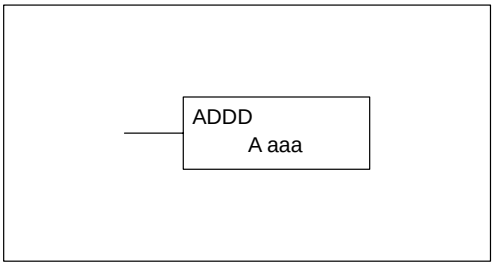


Handheld Programmer Keystrokes

$ 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)

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.

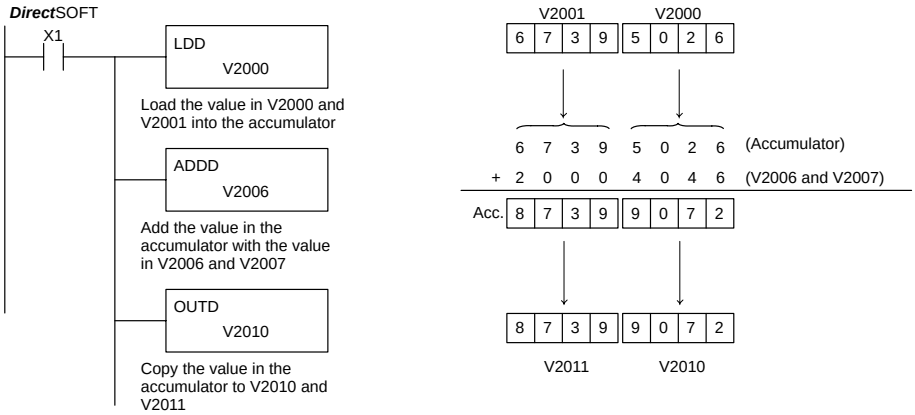


Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	0–99999999

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: 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 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.

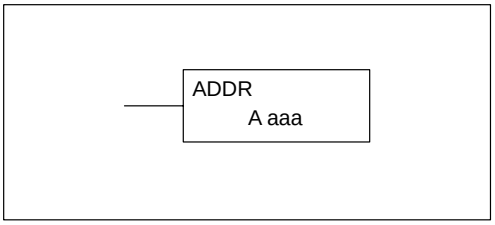


Handheld Programmer Keystrokes

$ 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)

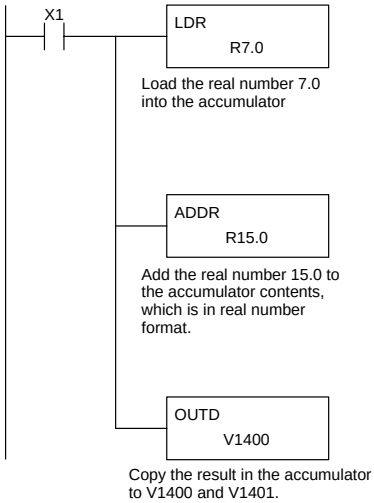
The Add Real instruction adds a real number in the accumulator with either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must conform to the IEEE floating point format.

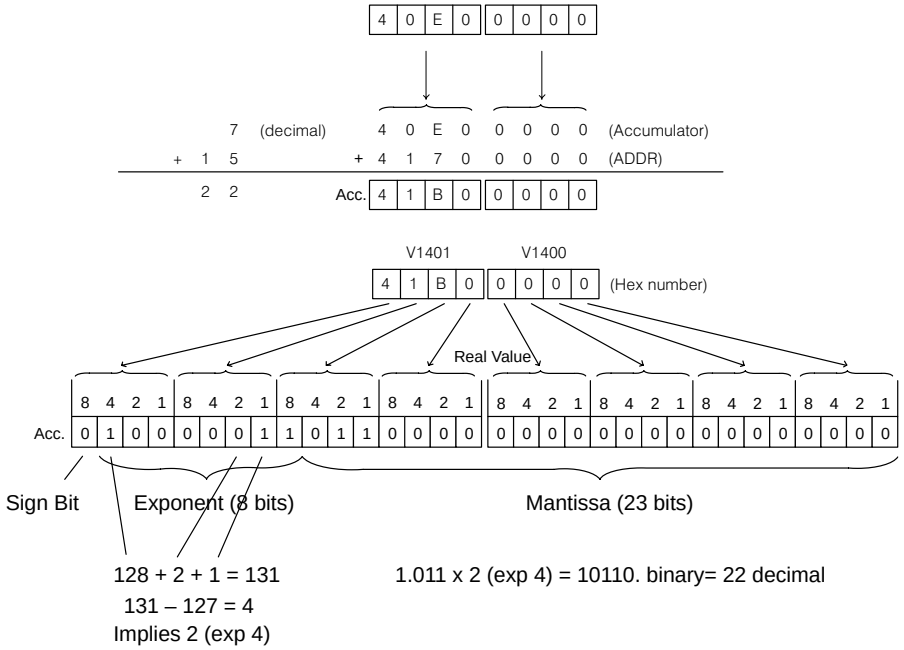


Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Pointer	P	All V mem (See p. 3–29)
Constant	R	–3.402823E+038 to +3.402823E+038

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.
SP75	On when a real number instruction is executed and a non-real number was encountered.

NOTE: Status flags are valid only until another instruction uses the same flag.

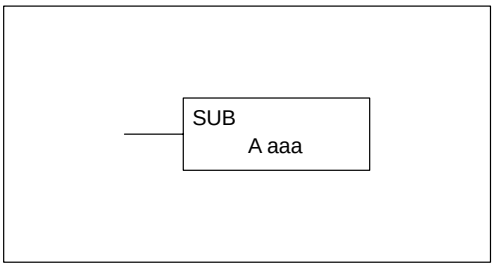




NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use ***Direct***SOFT for this feature.

Subtract (SUB)

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. You cannot use a constant as the parameter in the box.

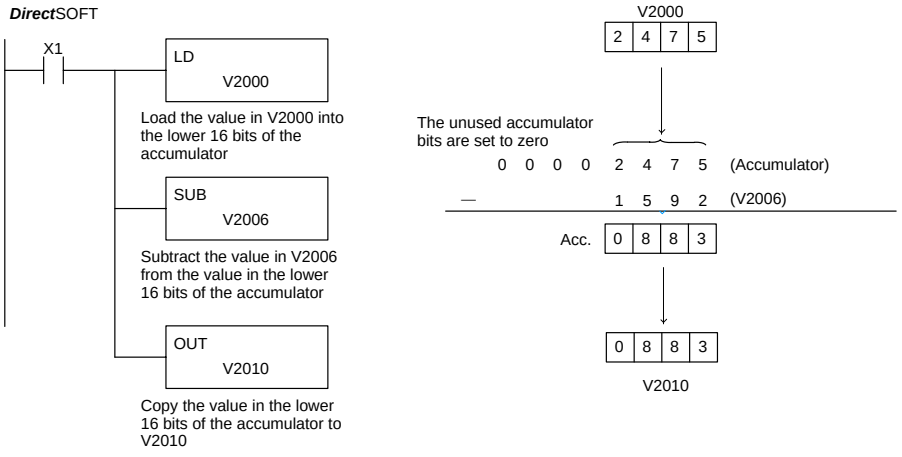


Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

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: 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 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.

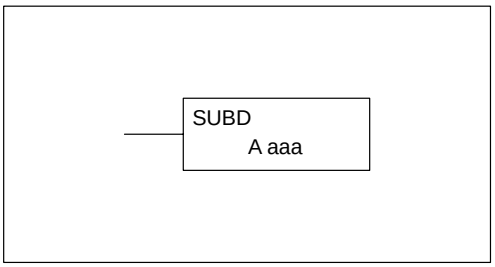


Handheld Programmer Keystrokes

$ 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)

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. The result resides in the accumulator.



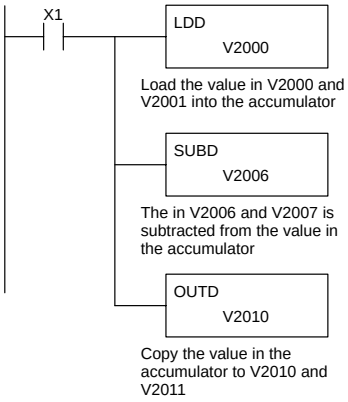
Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	0–99999999

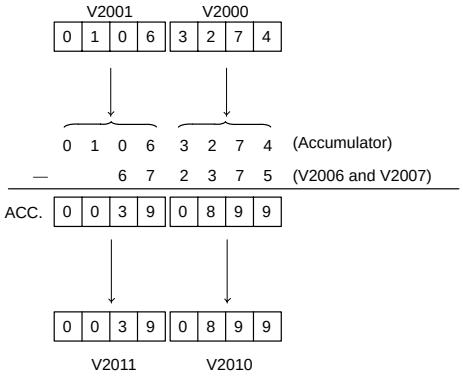
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16 bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

NOTE: 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 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.

DirectSOFT



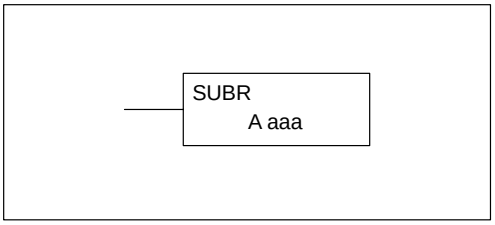


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT								
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT		
SHFT	S RST	SHFT	U ISG	B 1	D 3	→	C 2	A 0	A 0	G 6	ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT			

Subtract Real (SUBR)

The Subtract Real instruction subtracts a real number in the accumulator from either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must conform to the IEEE floating point format.

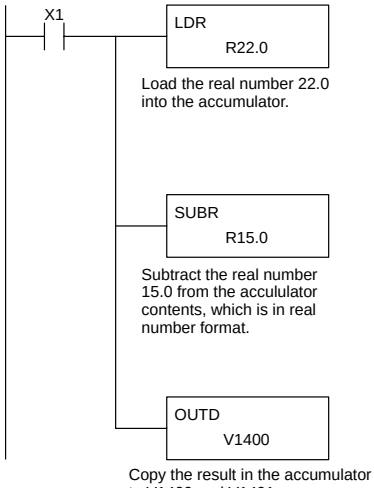


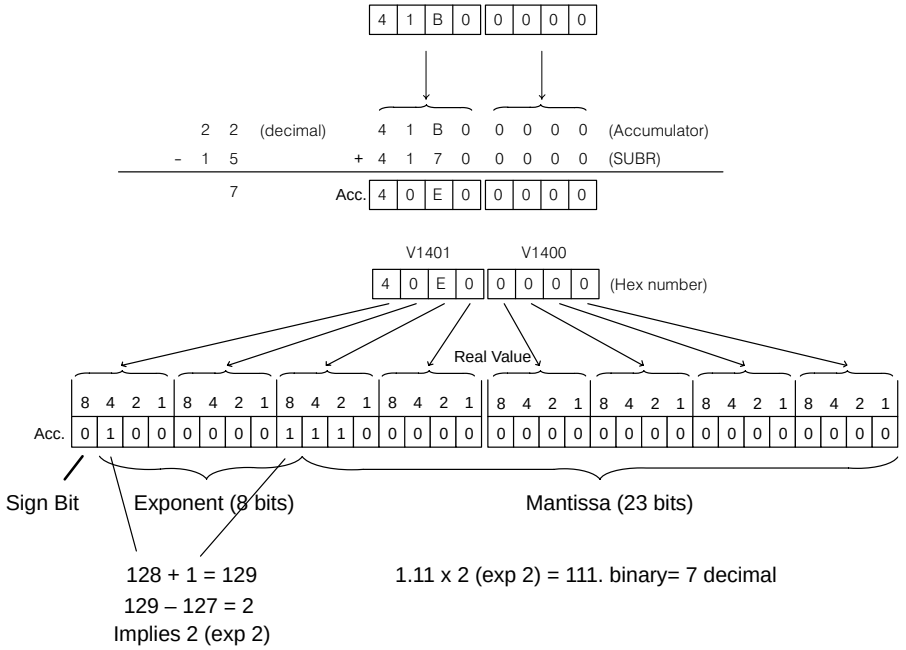
Operand Data Type		DL350 Range
	A	aaa
Vmemory	V	All (See p. 3–29)
Pointer	P	All V mem (See p. 3–29)
Constant	R	–3.402823E+038 to +3.402823E+038

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 a valid 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.
SP75	On when a real number instruction is executed and a non-real number was encountered.

NOTE: Status flags are valid only until another instruction uses the same flag.

***Direct*SOFT Display**

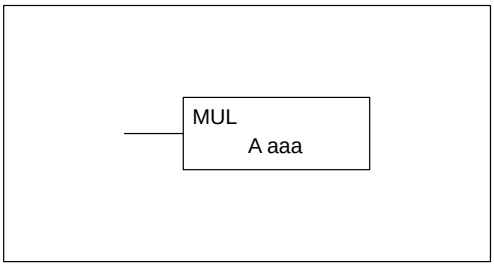




NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use ***Direct*SOFT** for this feature.

Multiply (MUL)

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.



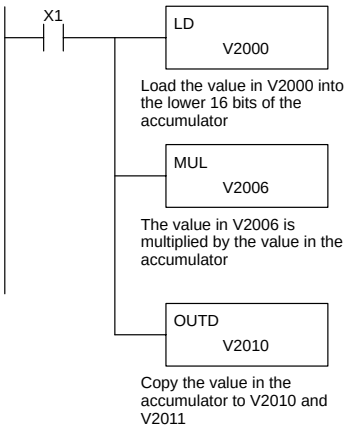
Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	0–9999

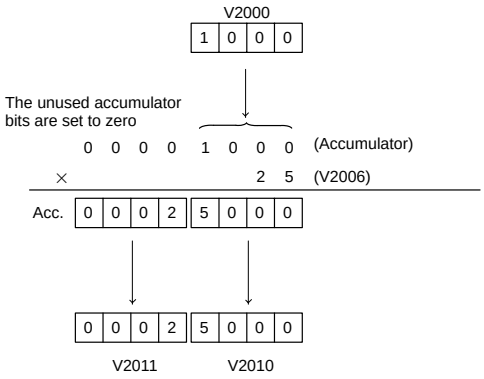
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

NOTE: 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 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.

DirectSOFT



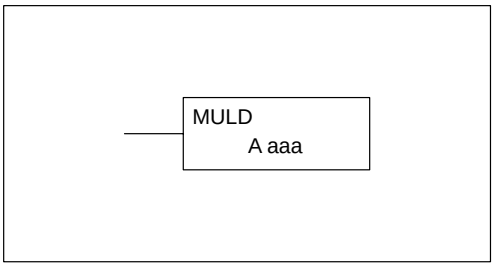


Handheld Programmer Keystrokes

$ 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)

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. You cannot use a constant as the parameter in the box. The lower 8 digits of the results reside in the accumulator. Upper digits of the result reside in the accumulator stack.



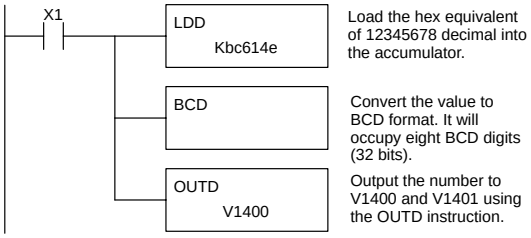
Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Pointer	P	—

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 numberis stored in V1400 and V1401. After loading the constant K2 into the accumulator, we multiply it times 12345678, which is 24691356.

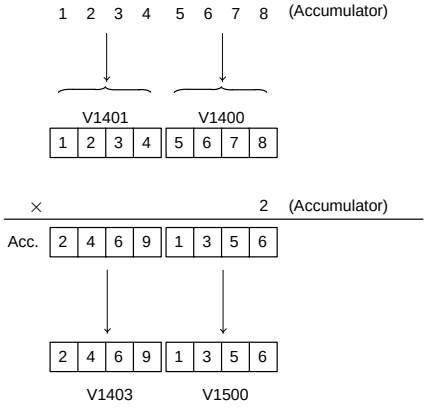
DirectSOFT Display



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.

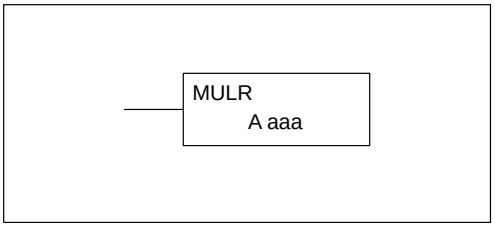


Handheld Programmer Keystrokes

STR	→	1	ENT	SHFT	L	D	D	→
K	SHFT	B(H)	C(H)	SHFT	6	1	4	ENT
SHFT	B	C	D	ENT				
OUT	SHFT	D	→	V	1	4	0	0	ENT

Multiply Real (MULR)

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 conform to the IEEE floating point format.

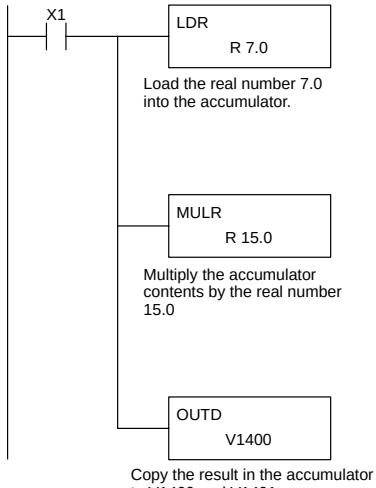


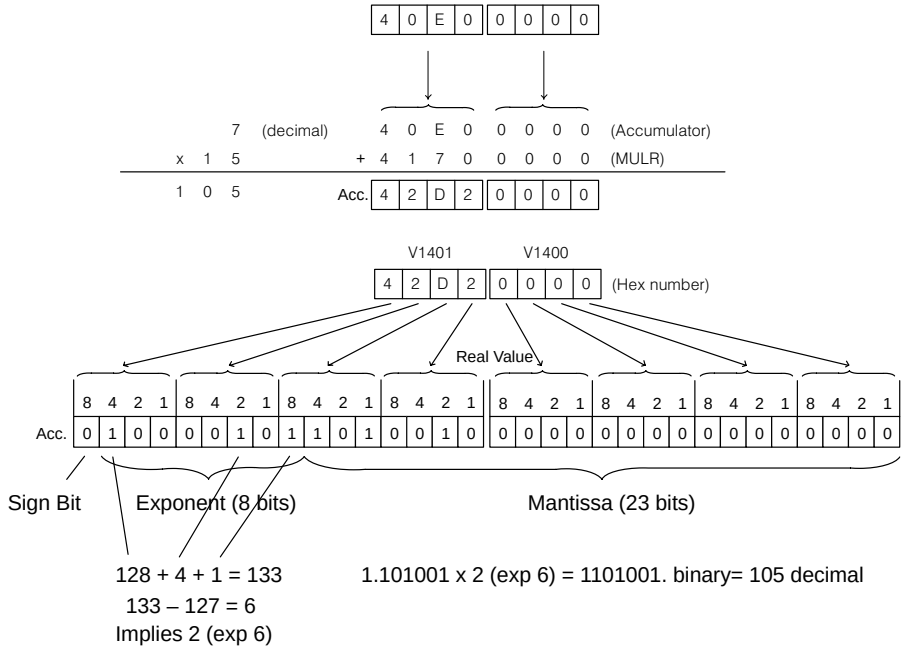
Operand Data Type		DL350 Range
	A	aaa
Vmemory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)
Constant	R	–3.402823E+038 to +3.402823E+038

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 a valid 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.
SP75	On when a real number instruction is executed and a non-real number was encountered.

NOTE: Status flags are valid only until another instruction uses the same flag.

***Direct*SOFT Display**

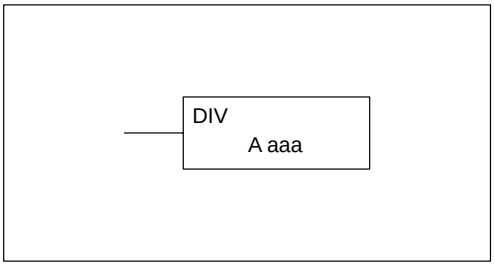




NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use ***Direct*SOFT** for this feature.

Divide (DIV)

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.



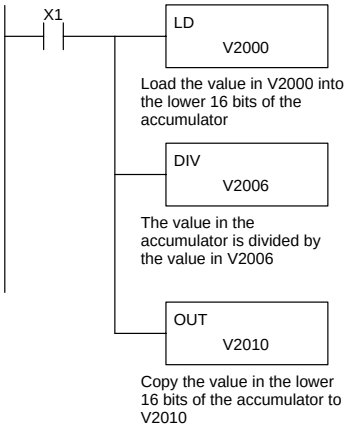
Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	0–9999

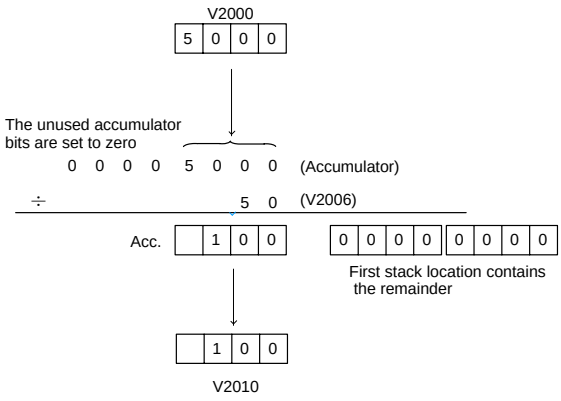
Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

NOTE: 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 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



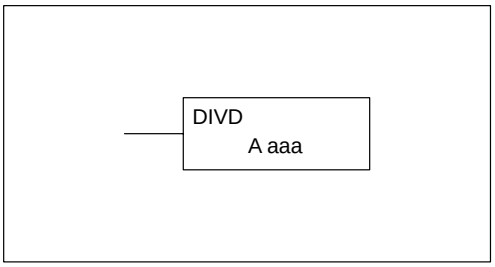


Handheld Programmer Keystrokes

$ 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)

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.



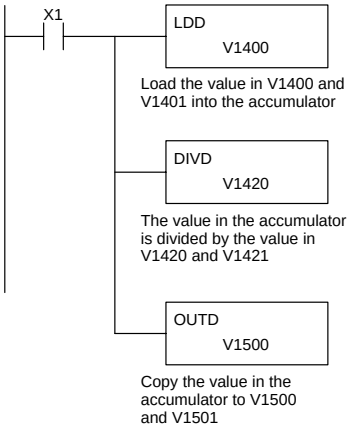
Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)

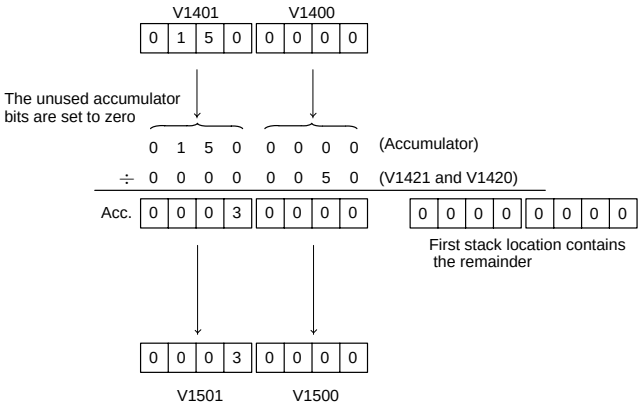
Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed 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 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 an 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 Display



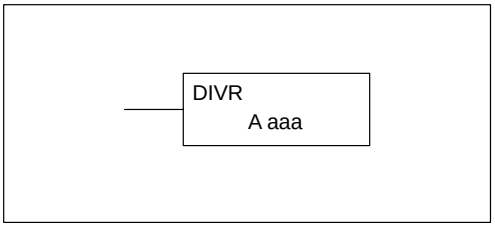


Handheld Programmer Keystrokes

STR	→	1	ENT	SHFT	L	D	D	→		
V	1	4	0	0	ENT					
SHFT	D	I	V	→	V	1	4	2	0	ENT
OUT	SHFT	D	→	V	1	5	0	0	ENT	

Divide Real (DIVR)

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 conform to the IEEE floating point format.

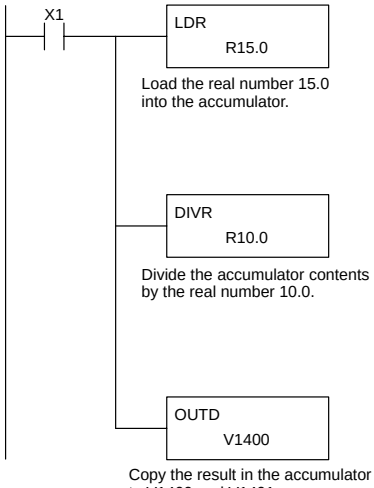


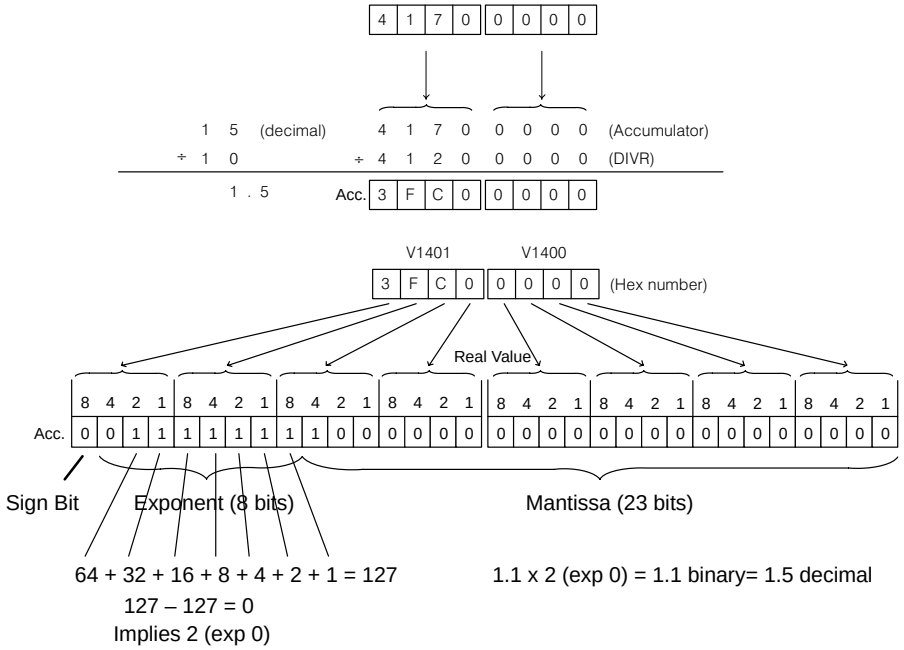
Operand Data Type		DL350 Range
	A	aaa
Vmemory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)
Constant	R	–3.402823E+038 to +3.402823E+038

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 a valid 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.
SP75	On when a real number instruction is executed and a non-real number was encountered.

NOTE: Status flags are valid only until another instruction uses the same flag.

DirectSOFT Display

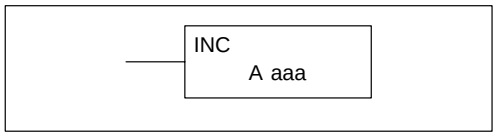




NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use ***Direct***SOFT for this feature.

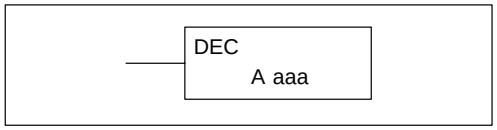
Increment (INC)

The Increment instruction increments a BCD value in a specified V memory location by “1” each time the instruction is executed.



Decrement (DEC)

The Decrement instruction decrements a BCD value in a specified V memory location by “1” each time the instruction is executed.

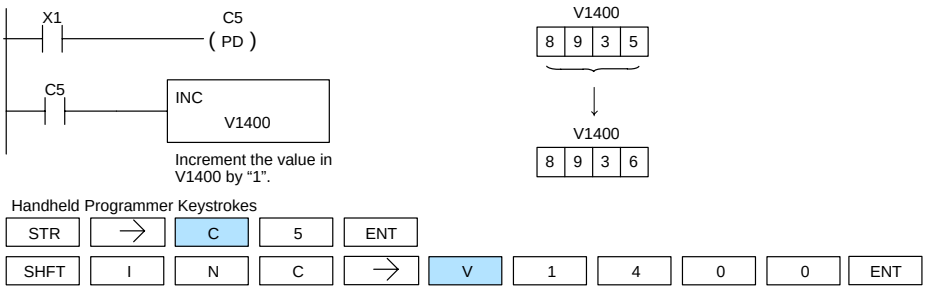


Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)

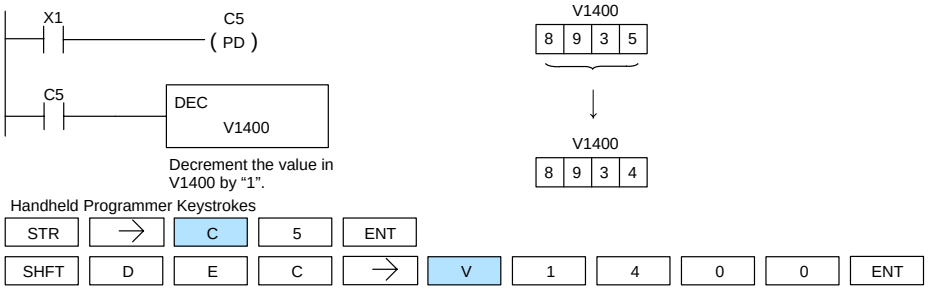
Discrete Bit Flags	Description
SP63	on when the result of the instruction causes the value in the accumulator to be zero.
SP75	on when a BCD instruction is executed and a NON–BCD number was encountered.

NOTE: Status flags are valid only until another instruction uses the same flag.

In the following increment example, when C5 is on the value in V1400 increases by one.

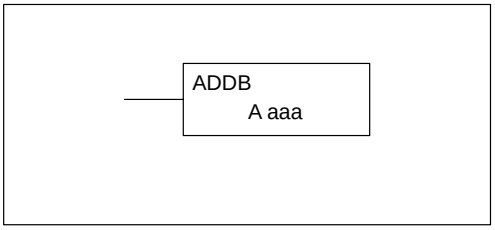


In the following decrement example, when C5 is on the value in V1400 is decreased by one.



Add Binary (ADDB)

Add Binary is a 16 bit instruction that adds the unsigned 2's complement binary value in the lower 16 bits of the accumulator with an unsigned 2's complement binary value (Aaaa), which is either a V memory location or a 16-bit constant. The result can be up to 32 bits (unsigned 2's complement) and resides in the accumulator.



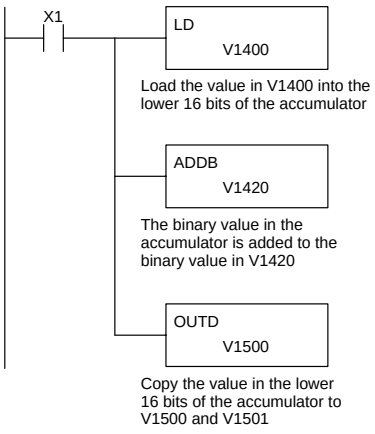
Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Pointer	P	All V mem (See p. 3–29)
Constant	K	0–FFFF

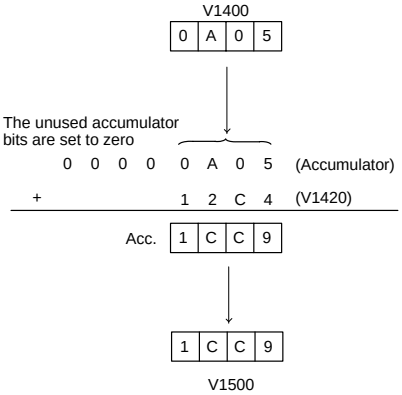
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16 bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in a 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 instruction.

DirectSOFT Display



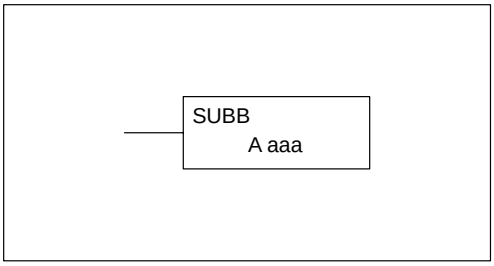


Handheld Programmer Keystrokes

STR	→	X(IN)	1	ENT						
SHFT	L	D	V	1	4	0	0	ENT		
SHFT	A	D	D	B	→	V	1	4	2	0	ENT
OUT	SHFT	D	→	V	1	5	0	0	ENT	

Subtract Binary (SUBB)

Subtract Binary is a 16 bit instruction that subtracts the unsigned 2–s complement binary value (Aaaa), which is either a V memory location or a 16-bit 2's complement binary value, from the binary value in the accumulator. The result resides in the accumulator.



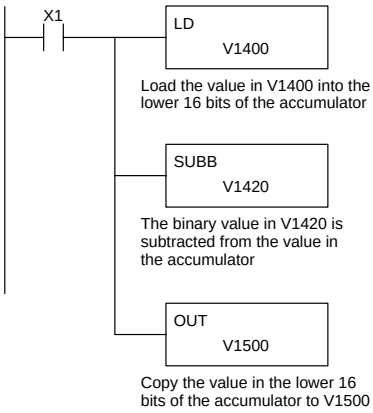
Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)
Constant	K	0–FFFF

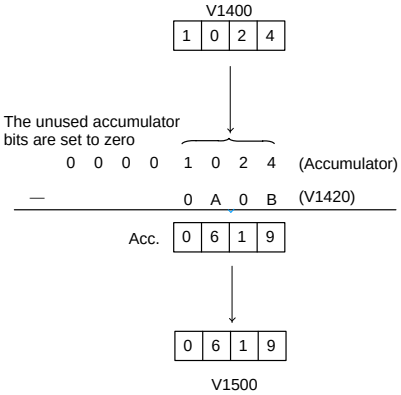
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16 bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator 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.

DirectSOFT Display



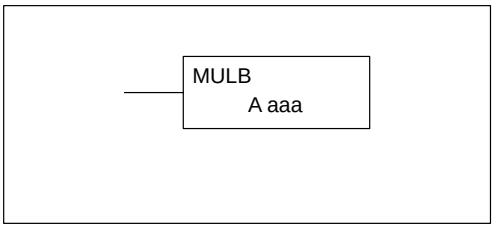


Handheld Programmer Keystrokes

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

Multiply Binary (MULB)

Multiply Binary is a 16 bit instruction that multiplies the unsigned 2's complement binary value (Aaaa), which is either a V memory location or a 16-bit unsigned 2's complement binary constant, by the16-bit binary value in the accumulator The result can be up to 32 bits and resides in the accumulator.



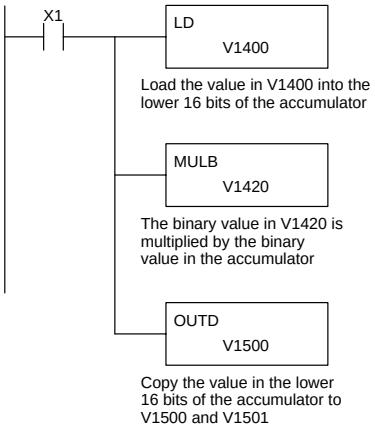
Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)
Constant	K	0–FFFF

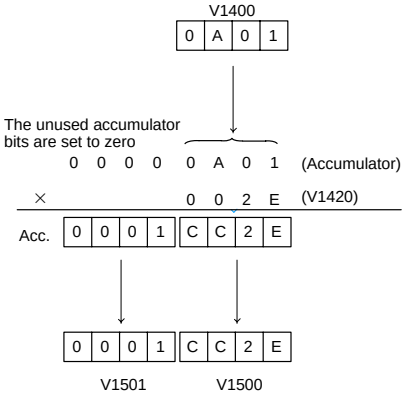
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

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.

DirectSOFT Display



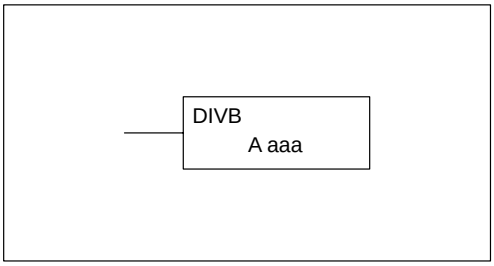


Handheld Programmer Keystrokes

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)

Divide Binary is a 16 bit instruction that divides the unsigned 2's complement binary value in the accumulator by a binary value (Aaaa), which is either a V memory location or a 16-bit unsigned 2's complement binary constant. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.



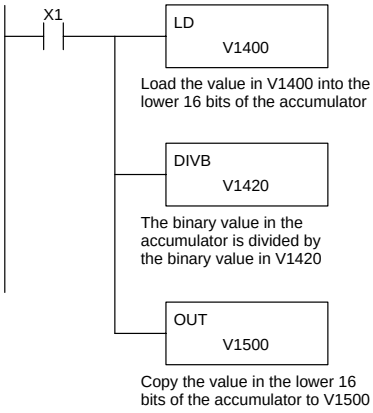
Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)
Constant	K	0–FFFF

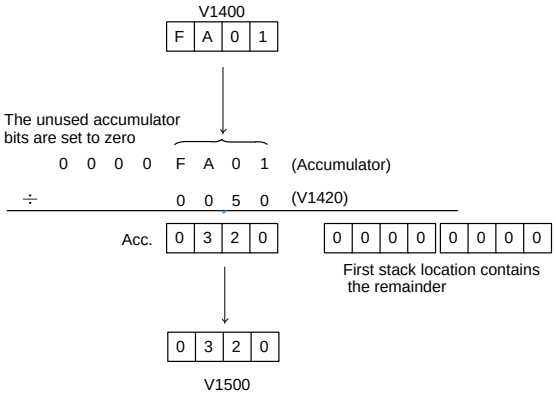
Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

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.

DirectSOFT Display



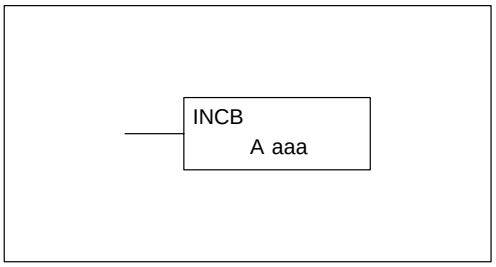


Handheld Programmer Keystrokes

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)

The Increment Binary instruction increments a binary value in a specified V memory location by “1” each time the instruction is executed.



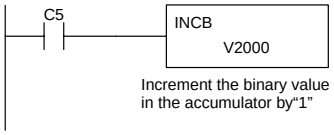
Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

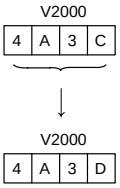
Discrete Bit Flags	Description
SP63	on when the result of the instruction causes the value in the accumulator to be zero.

NOTE: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example when C5 is on, the binary value in V2000 is increased by 1.

DirectSOFT



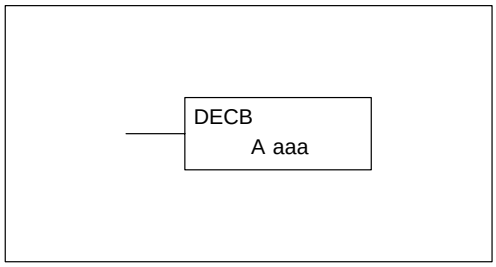


Handheld Programmer Keystrokes

$ STR	→	SHFT	C 2	F 5	ENT					
SHFT	I 8	N TMR	C 2	B 1	→	C 2	A 0	A 0	A 0	ENT

Decrement Binary (DECB)

The Decrement Binary instruction decrements a binary value in a specified V memory location by “1” each time the instruction is executed.

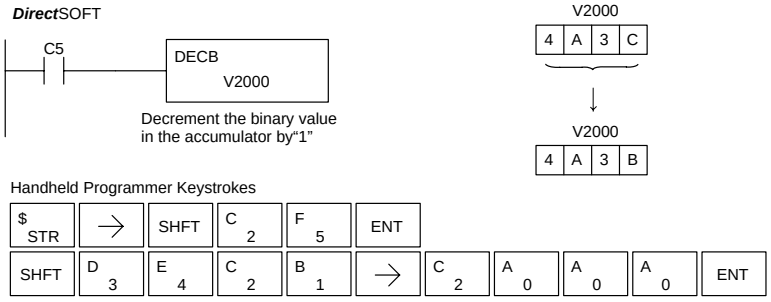


Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

Discrete Bit Flags	Description
SP63	on when the result of the instruction causes the value in the accumulator to be zero.

NOTE: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example when C5 is on, the value in V2000 is decreased by 1.



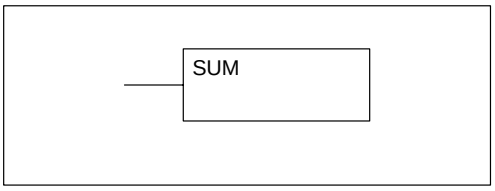
Handheld Programmer Keystrokes

$ STR	→	SHFT	C 2	F 5	ENT					
SHFT	D 3	E 4	C 2	B 1	→	C 2	A 0	A 0	A 0	ENT

Bit Operation Instructions

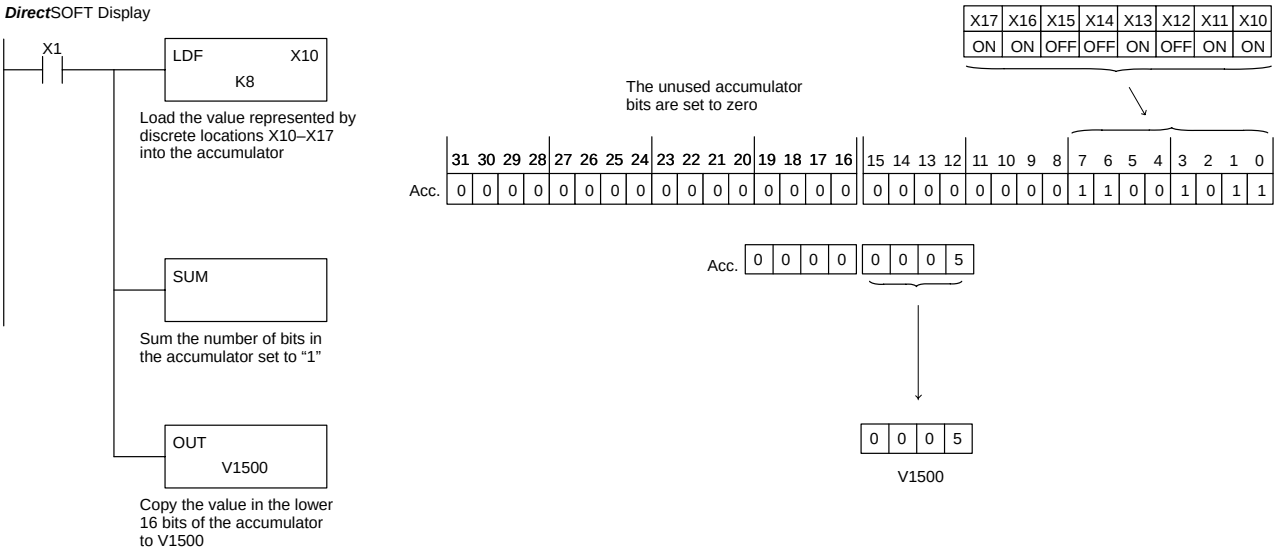
Sum (SUM)

The Sum instruction counts number of bits that are set to “1” in the accumulator. The HEX result resides in the accumulator.

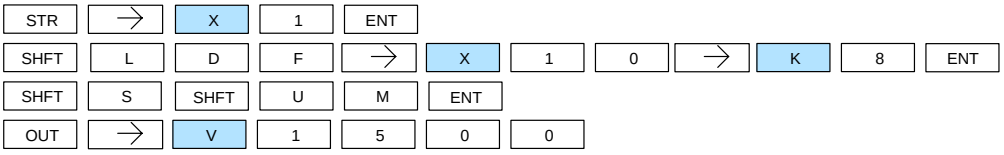


In the following example, when X1 is on, the value formed by discrete locations X10–X17 is loaded into the accumulator using the Load Formatted instruction. The number of bits in the accumulator set to “1” is counted using the Sum instruction. The value in the accumulator is copied to V1500 using the Out instruction.

DirectSOFT Display

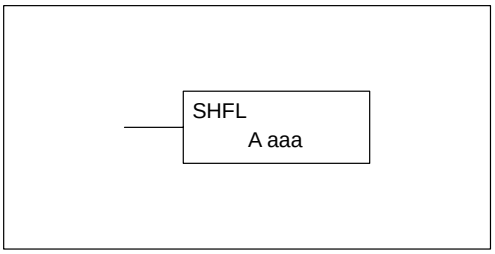


Handheld Programmer Keystrokes



Shift Left (SHFL)

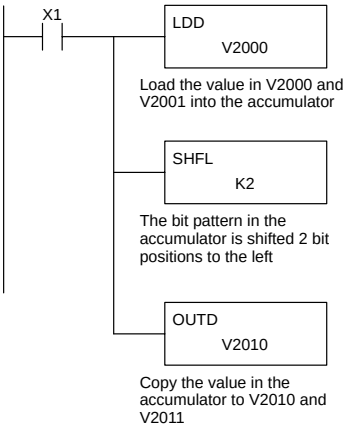
Shift Left is a 32 bit instruction that shifts the bits in the accumulator a specified number (Aaaa) of places to the left. The vacant positions are filled with zeros and the bits shifted out of the accumulator are lost.

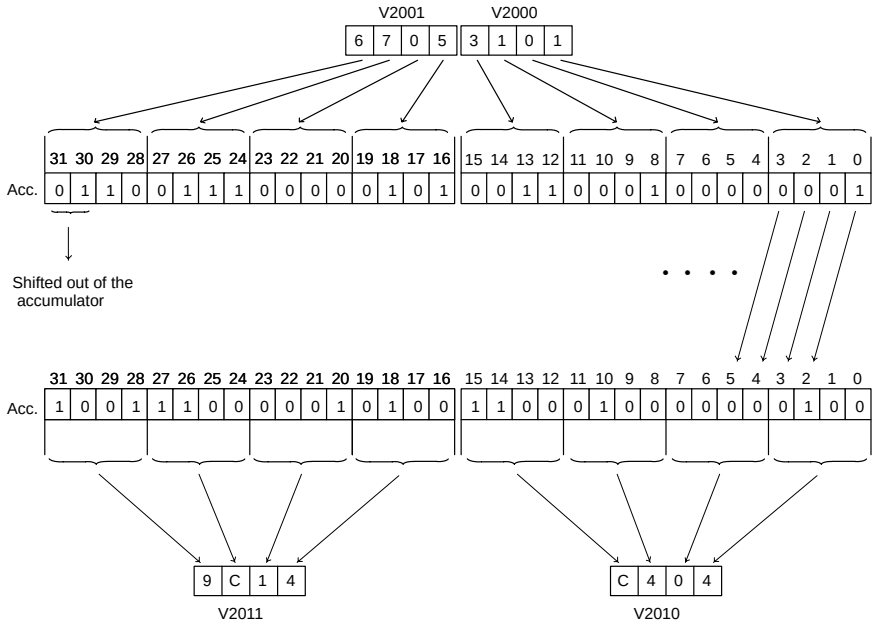


Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Constant	K	1–32

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.

DirectSOFT



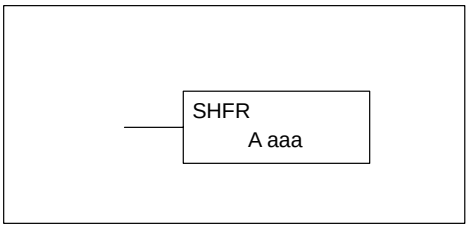


Handheld Programmer Keystrokes

$ 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)

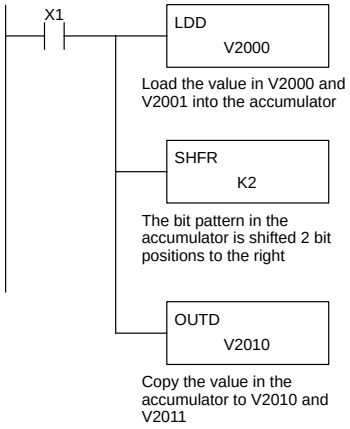
Shift Right is a 32 bit instruction that shifts the bits in the accumulator a specified number (Aaaa) of places to the right. The vacant positions are filled with zeros and the bits shifted out of the accumulator are lost.

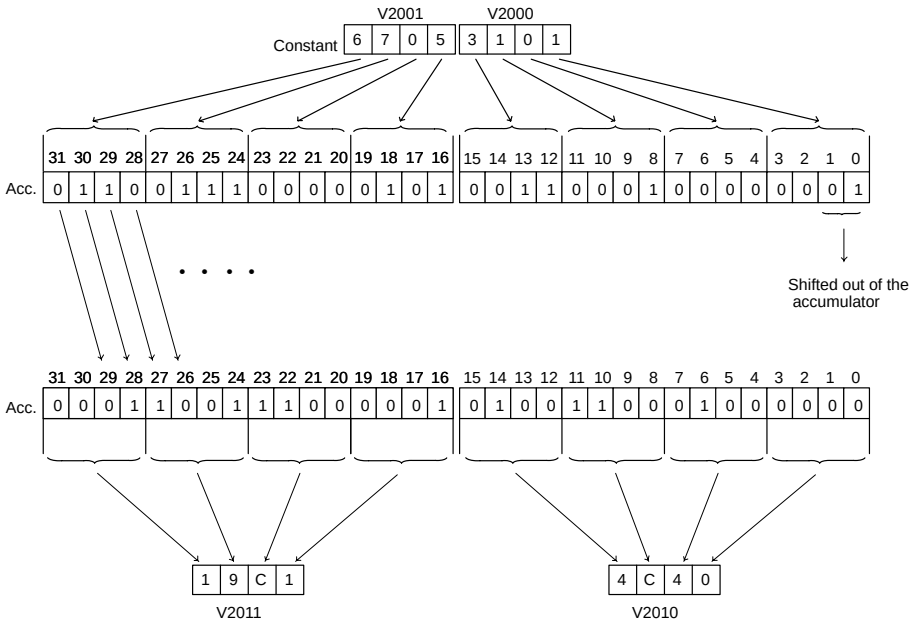


Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Constant	K	1–32

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.

***Direct*SOFT**



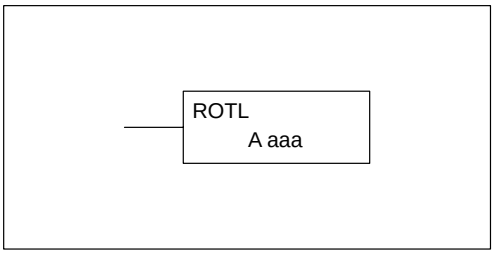


Handheld Programmer Keystrokes

$ 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)

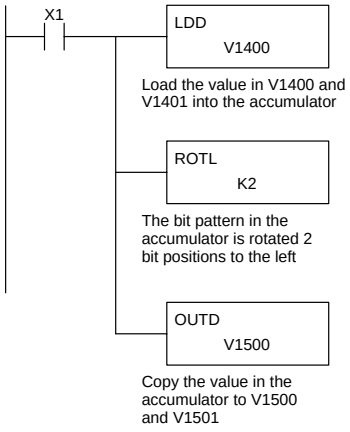
Rotate Left is a 32 bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the left.

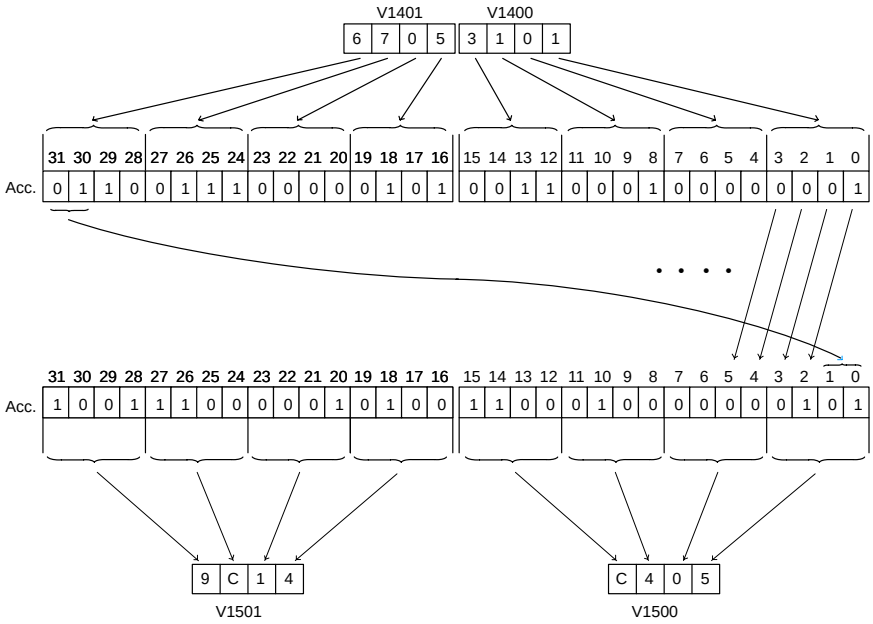


Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Constant	K	1–32

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.

DirectSOFT Display



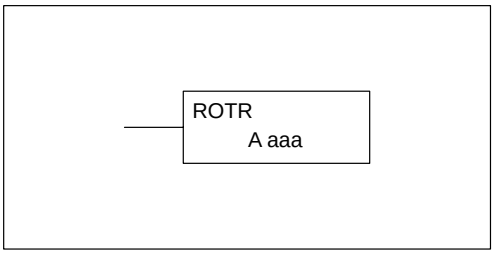


Handheld Programmer Keystrokes

STR	→	X	1	ENT					
SHFT	L	D	D	→	V	1	4	0	0
SHFT	R	O	T	L	→	K	2	ENT	
OUT	SHFT	D	→	V	1	5	0	0	ENT

Rotate Right (ROTR)

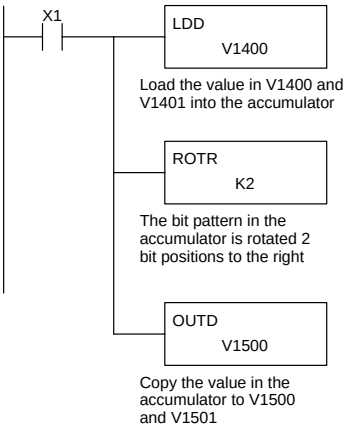
Rotate Right is a 32 bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the right.

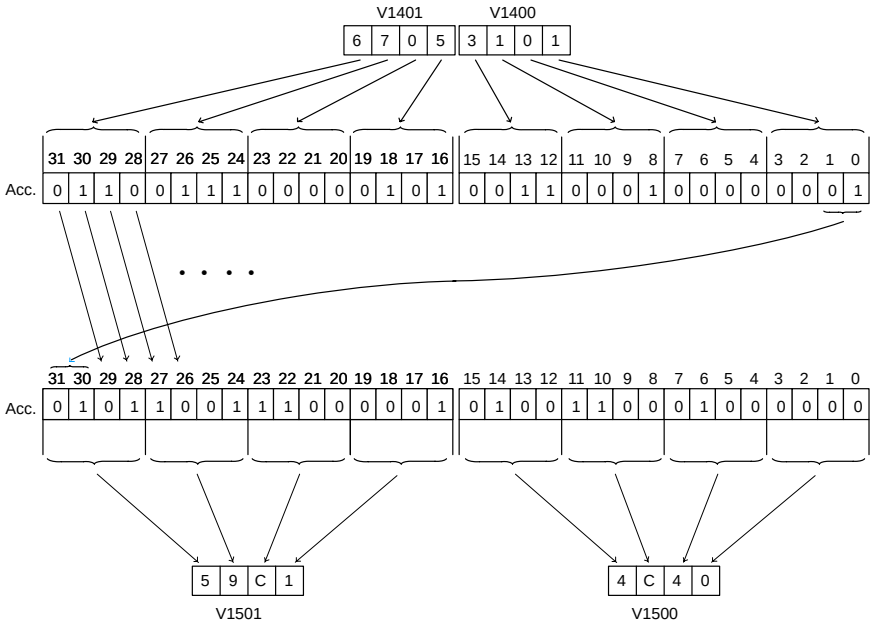


Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)
Constant	K	1–32

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.

DirectSOFT Display



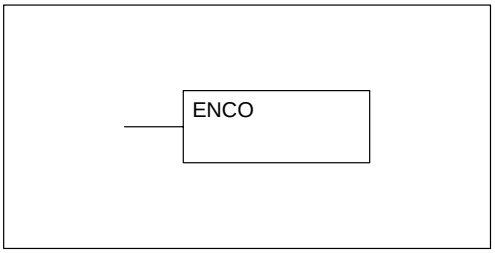


Handheld Programmer Keystrokes

STR	→	X	1	ENT					
SHFT	L	D	D	→	V	1	4	0	0
SHFT	R	O	T	R	→	K	2	ENT	
OUT	SHFT	D	→	V	1	5	0	0	ENT

Encode (ENCO)

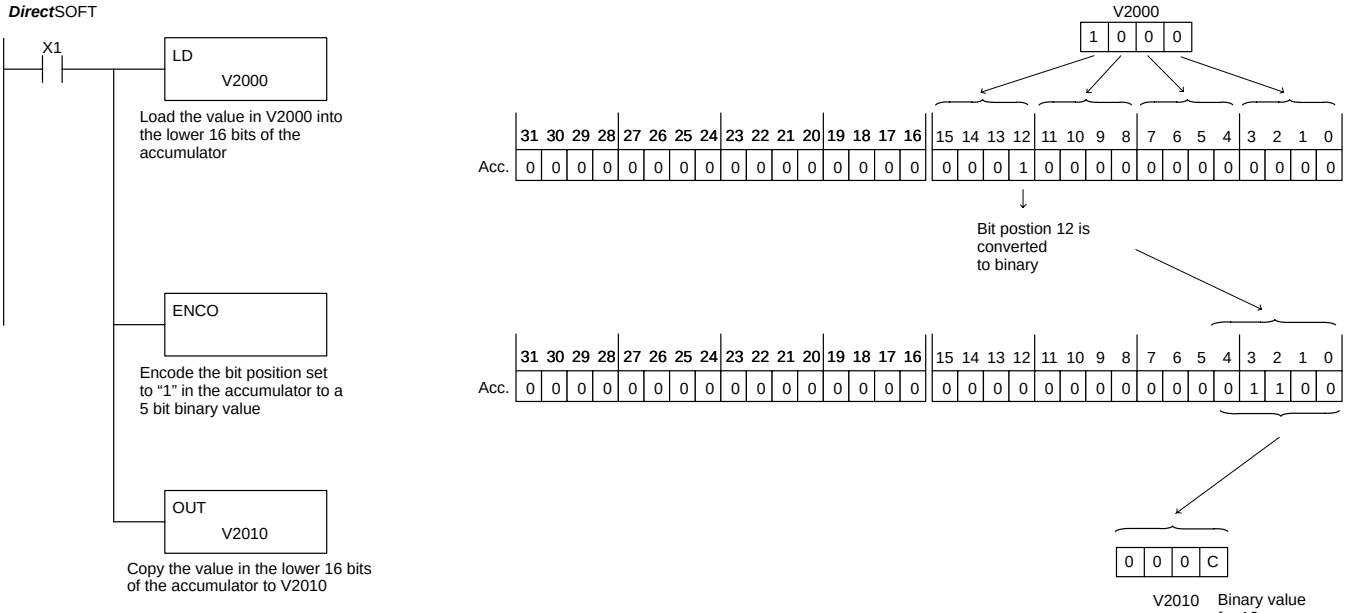
The Encode instruction encodes the bit position in the accumulator having a value of 1, and returns the appropriate binary representation. If the most significant bit is set to 1 (Bit 31), the Encode instruction would place the value HEX 1F (decimal 31) in the accumulator. If the value to be encoded 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.

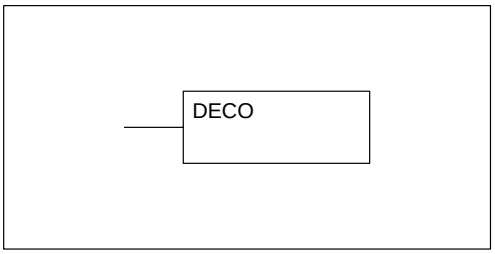


Handheld Programmer Keystrokes

$ 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

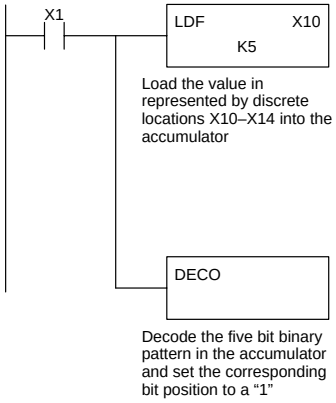
Decode (DECO)

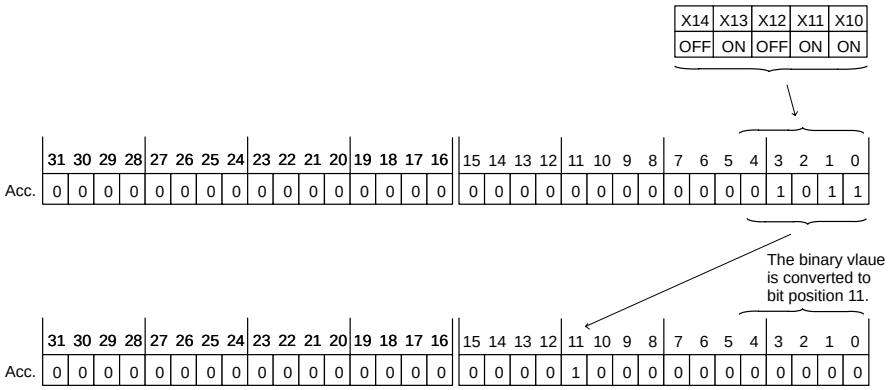
The Decode instruction decodes a 5 bit binary value of 0–31 (0–1F HEX) in the accumulator by setting the appropriate bit position to a 1. If the accumulator contains the value F (HEX), bit 15 will be set in the accumulator. If the value to be decoded is greater than 31, the number is divided by 32 until 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 five bit binary pattern in the accumulator is decoded by setting the corresponding bit position to a “1” using the Decode instruction.

DirectSOFT





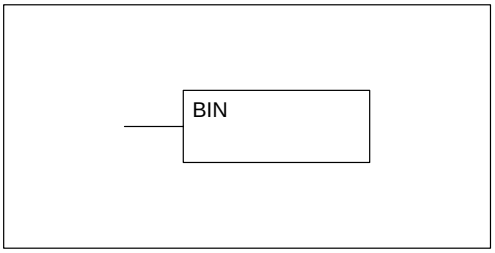
Handheld Programmer Keystrokes

$ 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)

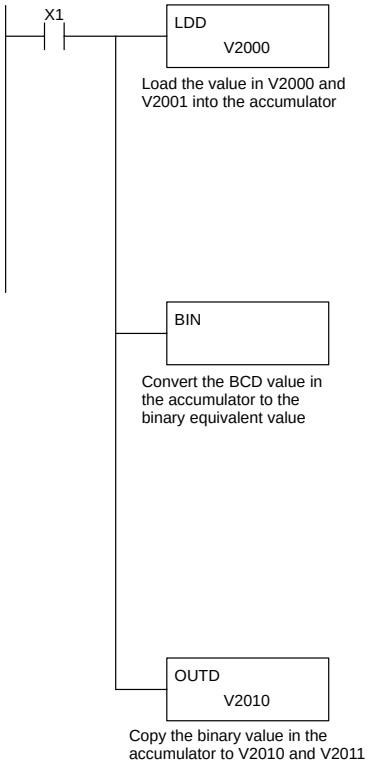
Binary (BIN)

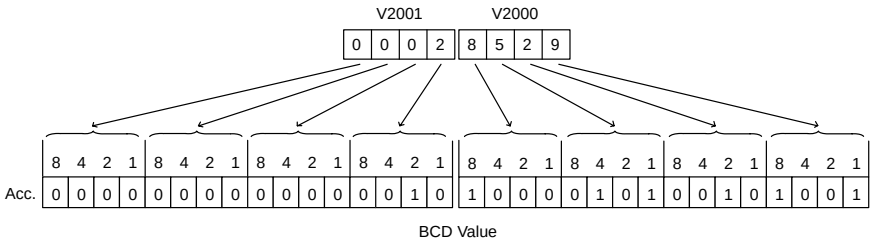
The Binary instruction converts a BCD value in the accumulator to the equivalent binary value. The result resides in the accumulator.



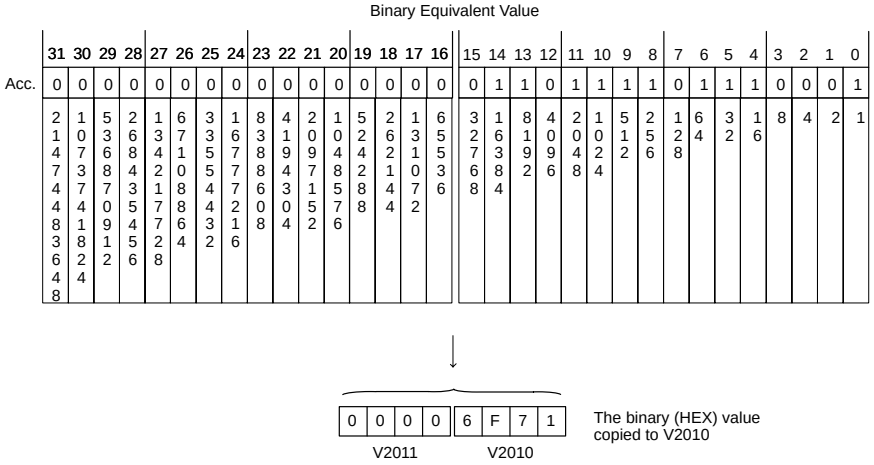
In the following example, when X1 is on, the value in V2000 and V2001 is loaded into the accumulator using the Load Double instruction. The BCD value in the accumulator is converted to the binary (HEX) equivalent using the BIN instruction. The binary value in the accumulator is copied to V2010 and V2011 using the Out Double instruction. (The handheld programmer will display the binary value in V2010 and V2011 as a HEX value.)

DirectSOFT





28529 = 16384 + 8192 + 2048 + 1024 + 512 + 256 + 64 + 32 + 16 + 1

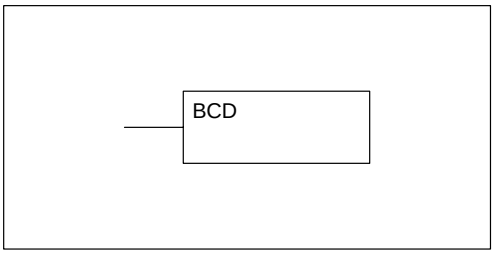


Handheld Programmer Keystrokes

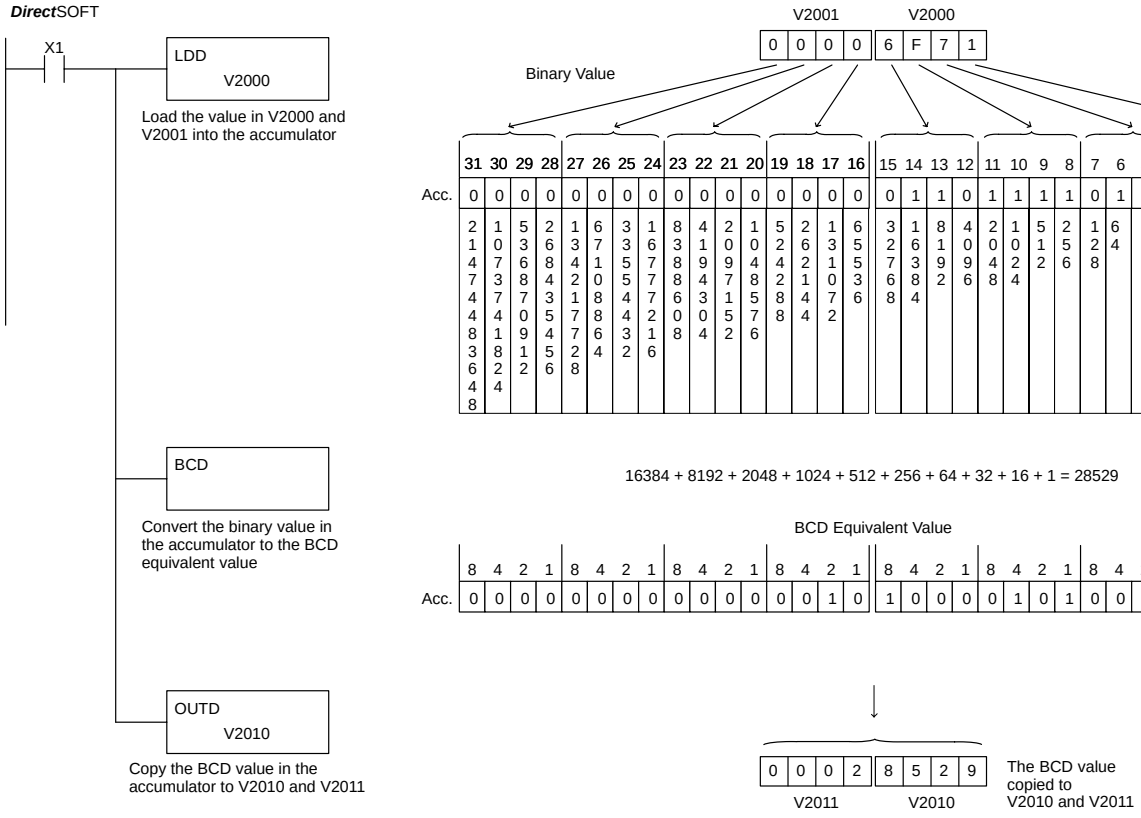
$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	B 1	I 8	N TMR	ENT				
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT

Binary Coded Decimal (BCD)

The Binary Coded Decimal instruction converts a binary value in the accumulator to the equivalent BCD value. The result resides in the accumulator.



In the following example, when X1 is on, the binary (HEX) value in V2000 and V2001 is loaded into the accumulator using the Load Double instruction. The binary 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.

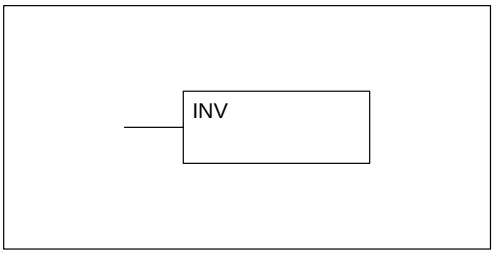


Handheld Programmer Keystrokes

$ 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

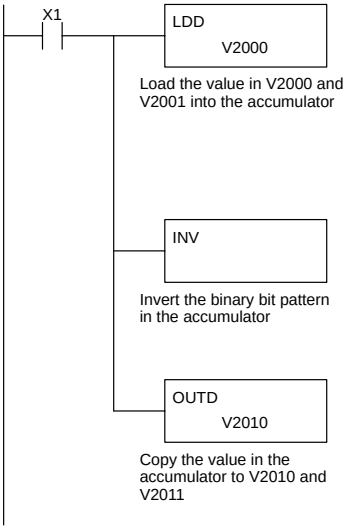
Invert (INV)

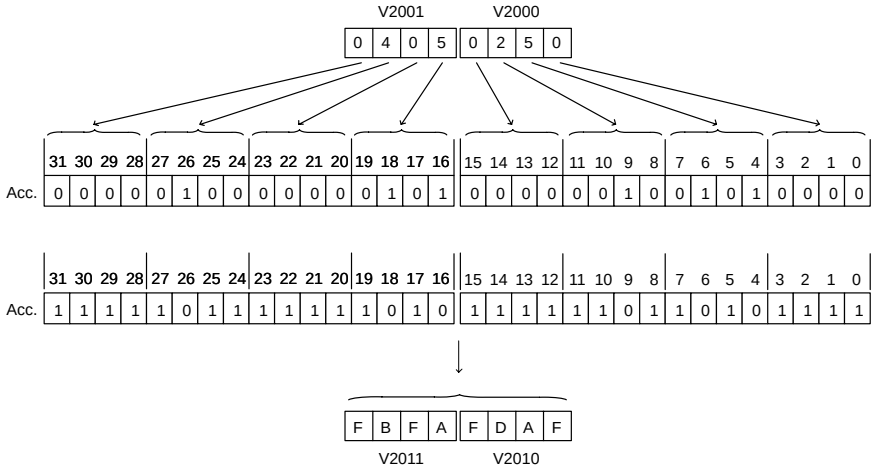
The Invert instruction inverts or takes the one's complement of the 32 bit value in the accumulator. The result resides in the accumulator.



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





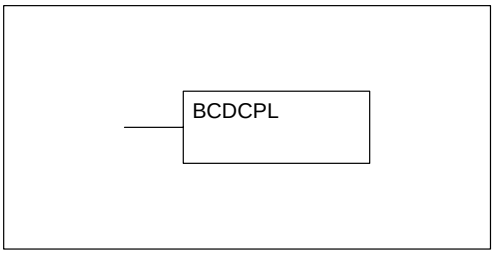
Handheld Programmer Keystrokes

$ 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	

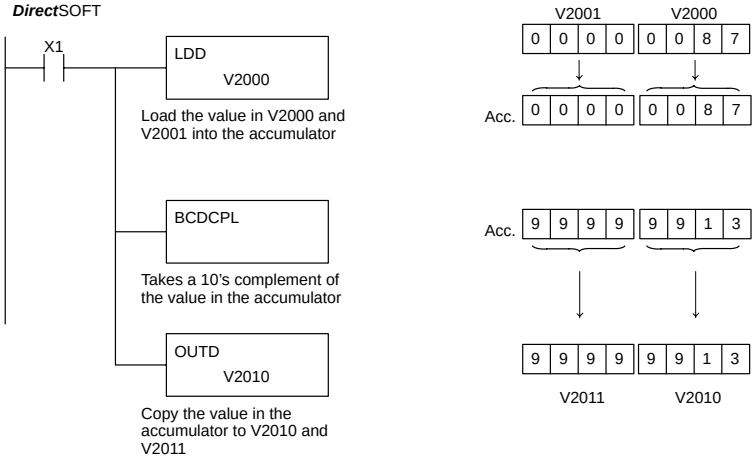
Ten's Complement (BCDCPL)

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 :

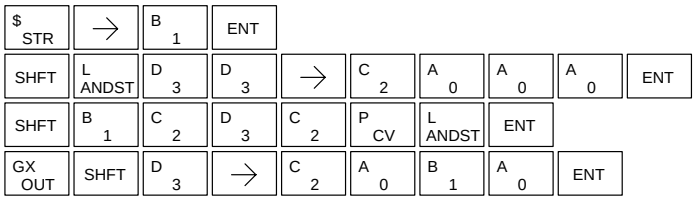
100000000
– accumulator value
10's complement value



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.

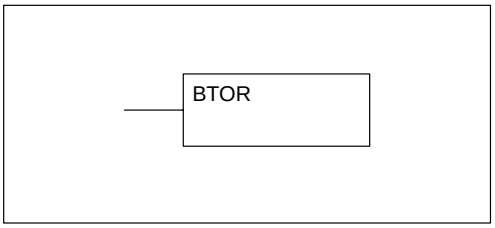


Handheld Programmer Keystrokes



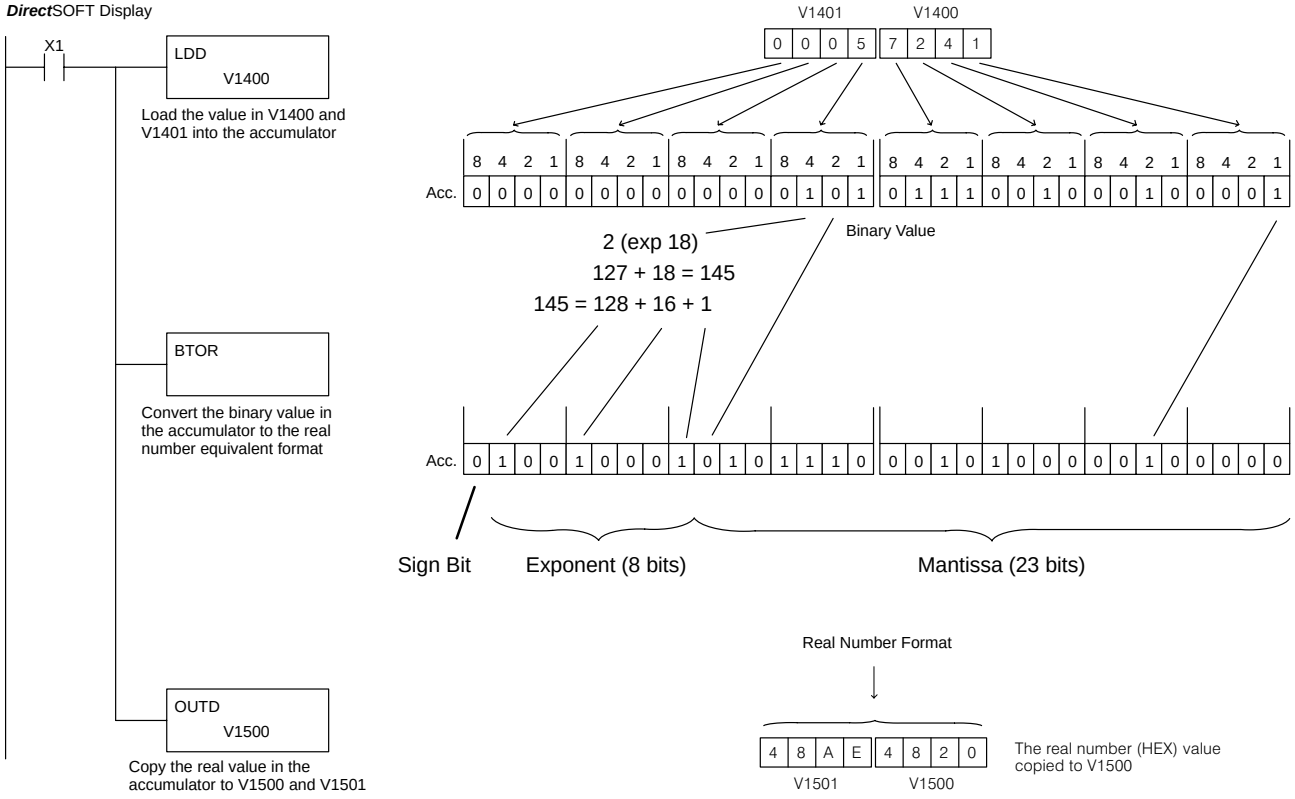
Binary to Real Conversion (BTOR)

The Binary-to-Real instruction converts a binary 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.



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 value in the accumulator 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.

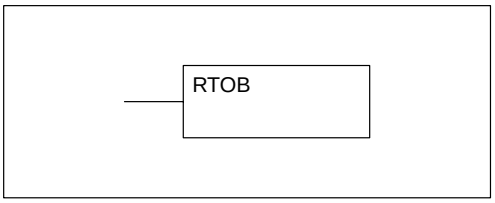


Handheld Programmer Keystrokes

STR	→	X	1	ENT					
SHFT	L	D	D	→	V	1	4	0	0	ENT
SHFT	B	T	O	R	SHFT	ENT				
OUT	SHFT	D	→	V	1	5	0	0	ENT	

Real to Binary Conversion (RTOB)

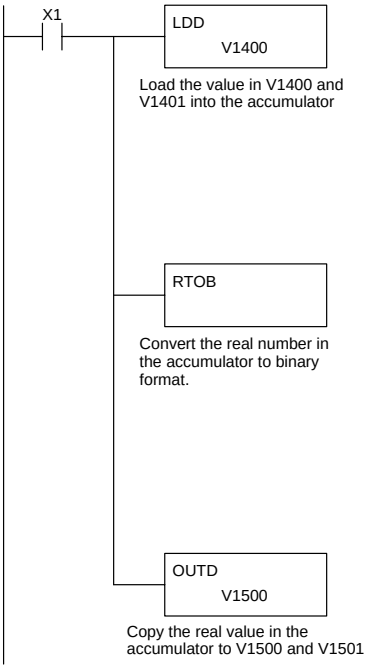
The Real-to-Binary instruction converts the real number in the accumulator to a binary value. The result resides in the accumulator. Both the binary and the real number may use all 32 bits of the accumulator.

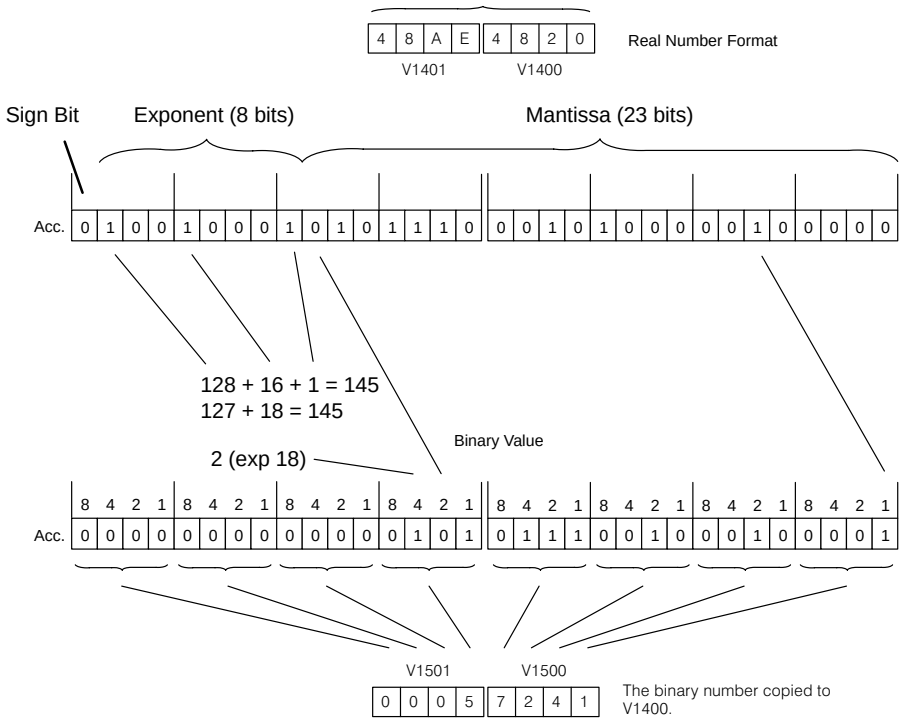


Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP72	On anytime the value in the accumulator is a valid floating point number.
SP73	on when a signed addition or subtraction results in a 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.

DirectSOFT Display



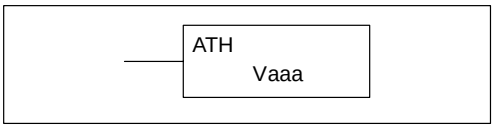


Handheld Programmer Keystrokes

STR	→	X	1	ENT						
SHFT	L	D	D	→	V	1	4	0	0	ENT
SHFT	R	T	O	B	SHFT	ENT				
OUT	SHFT	D	→	V	1	5	0	0	ENT	

ASCII to HEX (ATH)

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.

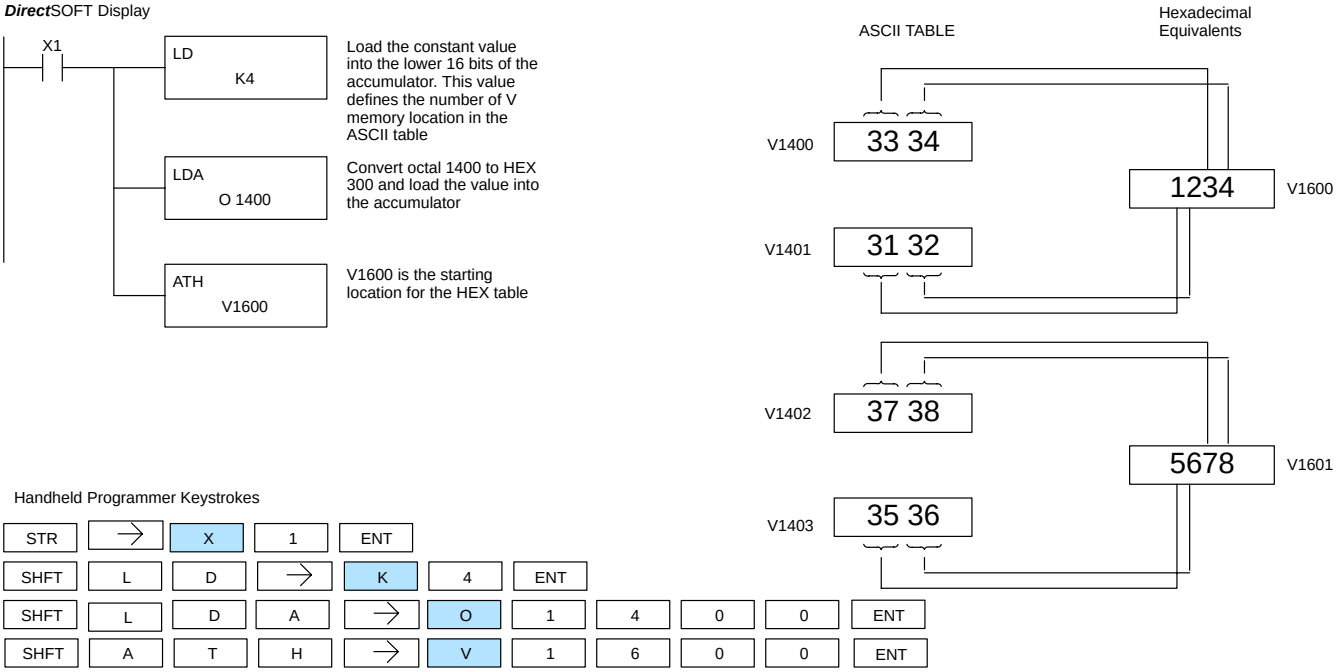
Step 3: — Specify the starting V memory location (Vaaa) for the HEX table in the ATH instruction.

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		DL350 Range
		aaa
Vmemory	V	All (See p. 3–29)

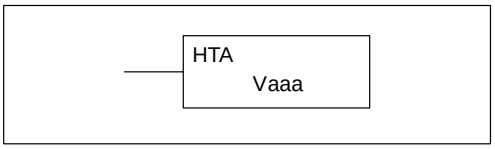
In the example on the following page, when X1 is ON the constant (K4) is loaded into the accumulator using the Load instruction and will be placed in the first level of the accumulator stack when the next Load instruction is executed. The starting location for the ASCII table (V1400) is loaded into the 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



HEX to ASCII (HTA)

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.

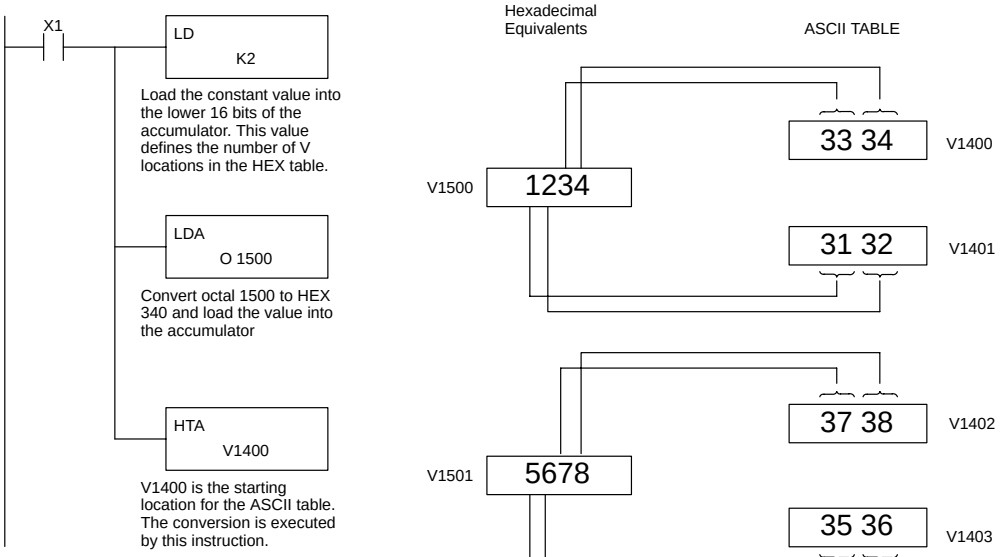
Step 3: — Specify the starting V memory location (Vaaa) for the ASCII table in the HTA instruction.

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		DL350 Range
		aaa
Vmemory	V	All (See p. 3–29)

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 Display



Handheld Programmer Keystrokes

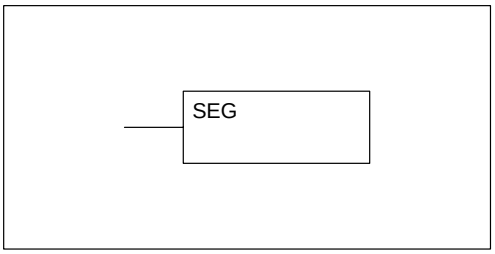
STR	→	X	1	ENT						
SHFT	L	D	→	K	4	ENT				
SHFT	L	D	A	→	O	1	5	0	0	ENT
SHFT	H	T	A	→	V	1	4	0	0	ENT

The table below lists valid ASCII values for HTA conversion.

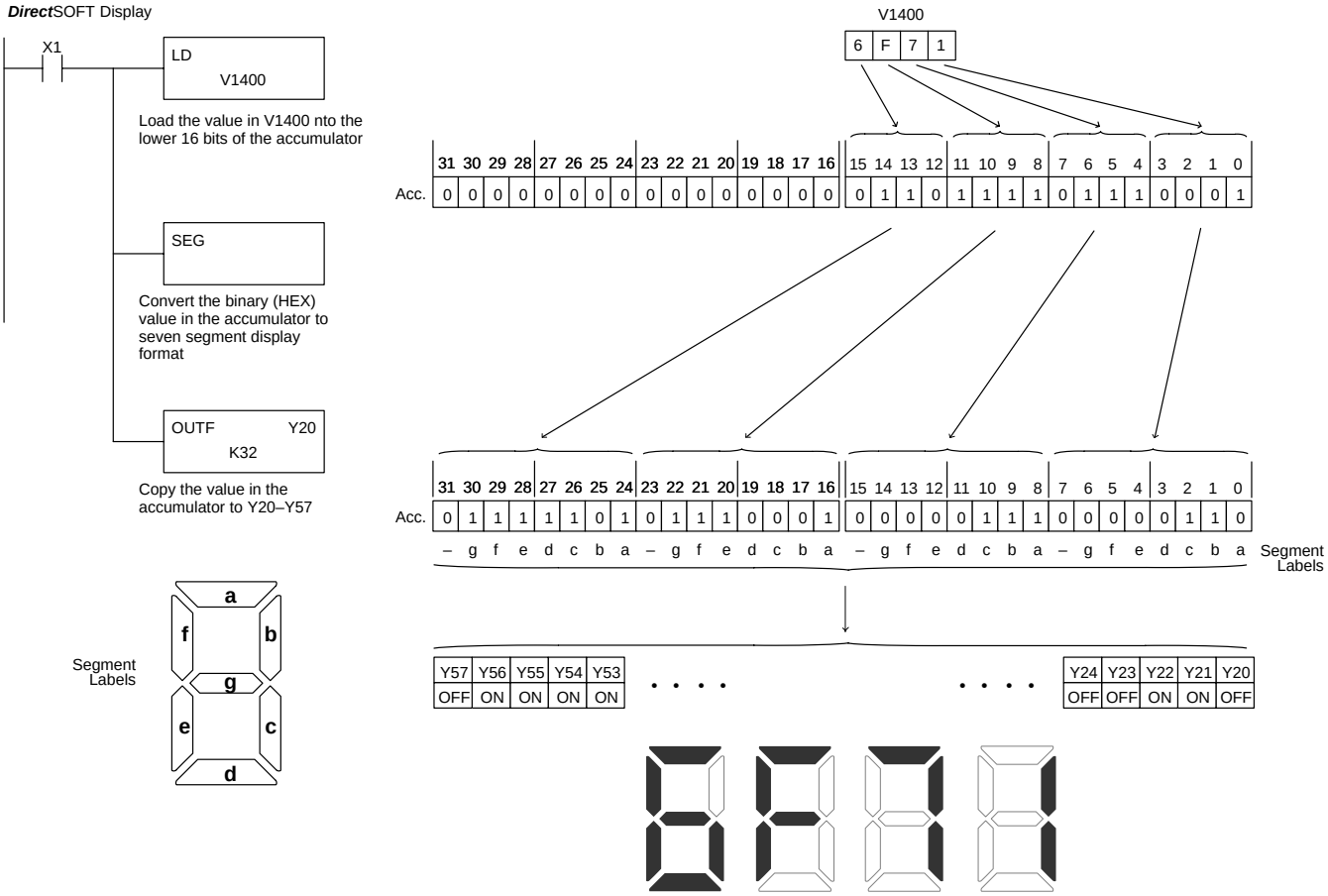
ASCII Values Valid for HTA Conversion			
Hex Value	ASCII Value	Hex Value	ASCII Value
0	30	8	38
1	31	9	39
2	32	A	41
3	33	B	42
4	34	C	43
5	35	D	44
6	36	E	45
7	37	F	46

Segment (SEG)

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 binary (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.

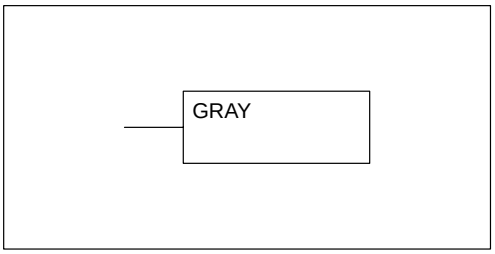


Handheld Programmer Keystrokes

STR	→	X	1	ENT					
L	D	→	V	1	4	0	0	ENT	
SHFT	S	SHFT	E	G	ENT				
OUT	SHFT	F	→	Y	2	0	→	K	3	2

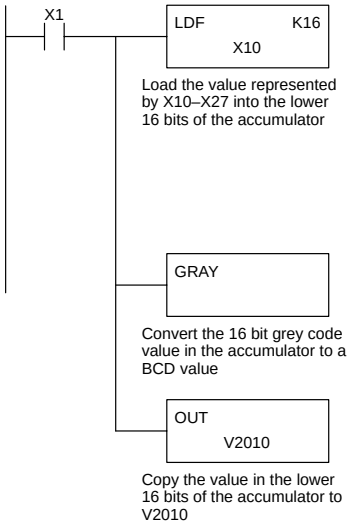
Gray Code (GRAY)

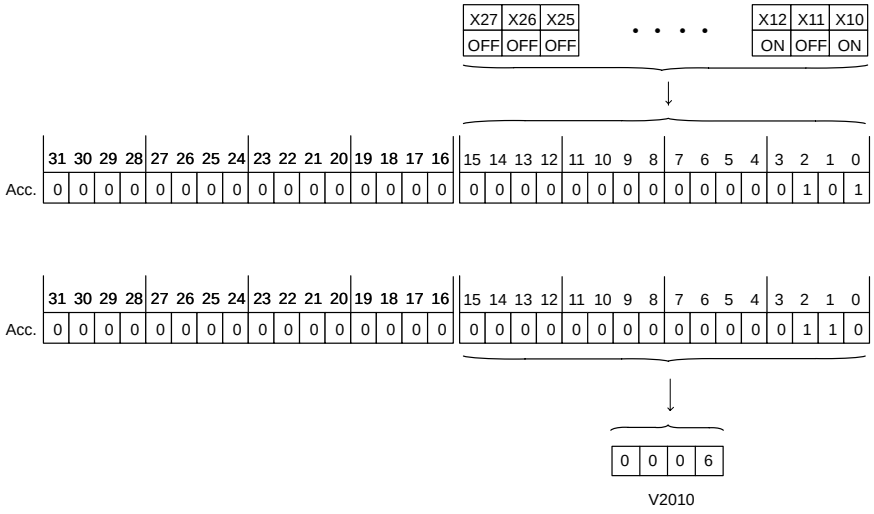
The Gray code instruction converts a 16 bit gray code value to a BCD value. The BCD conversion requires 10 bits of the accumulator. The upper 22 bits are set to "0". This instruction is designed for use with devices (typically encoders) that use the grey code numbering scheme. The Gray Code instruction 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.

DirectSOFT





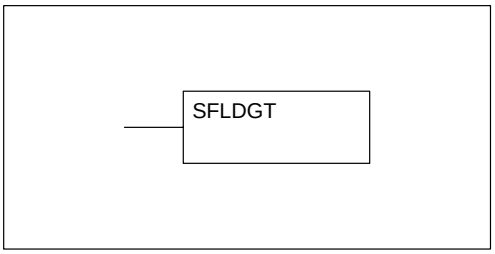
Handheld Programmer Keystrokes

$ 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		

Gray Code	BCD
0000000000	0000
0000000001	0001
0000000011	0002
0000000010	0003
0000000110	0004
0000000111	0005
0000000101	0006
0000000100	0007
•	•
•	•
•	•
1000000001	1022
1000000000	1023

Shuffle Digits (SFLDGT)

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.

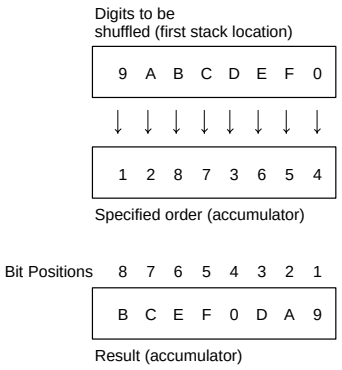
Note:— If the number used to specify the order contains a 0 or 9–F, the corresponding position will be set to 0.
See example on the next page.

Note:—If the number used to specify the order contains duplicate numbers, the most significant duplicate number is valid. The result resides in the accumulator.
See example on the next page.

Step 3:— Insert the SFLDGT instruction.

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 defines 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.

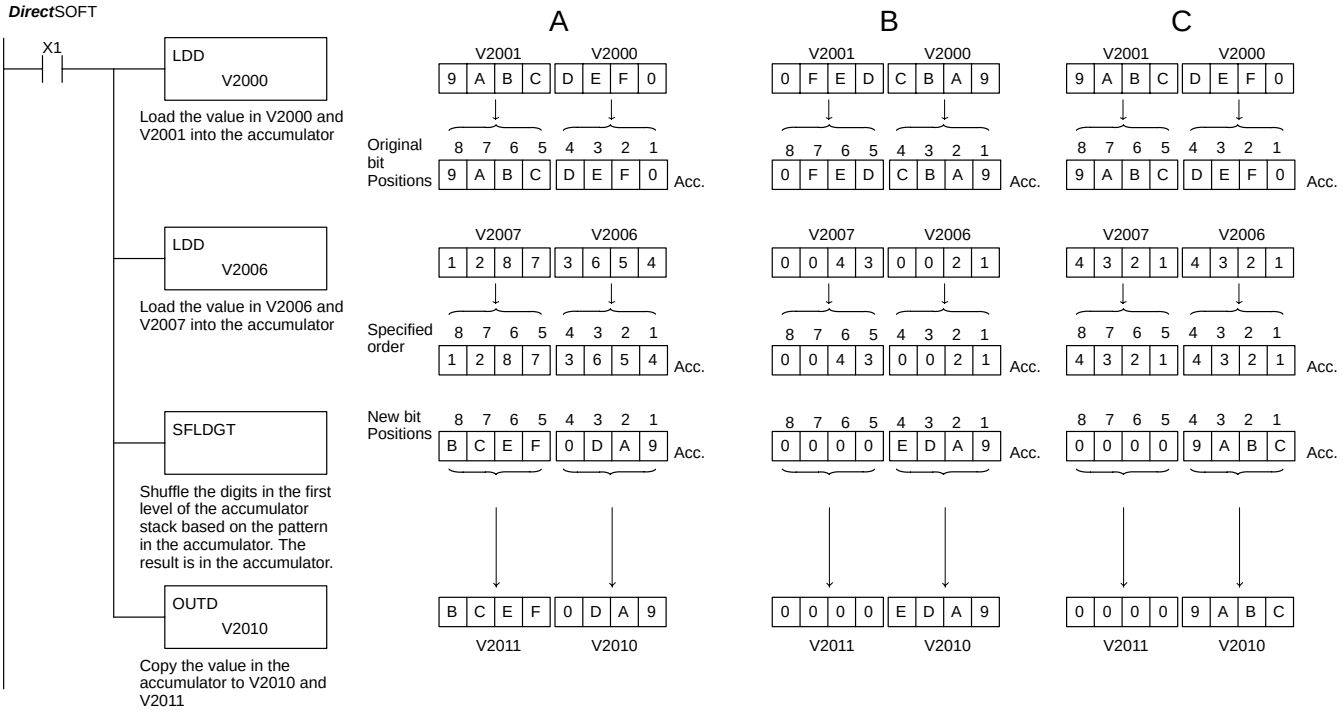


In the following example when X1 is on, The value in the first level of the accumulator stack will be reorganized in the order specified by the value in the accumulator.

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.



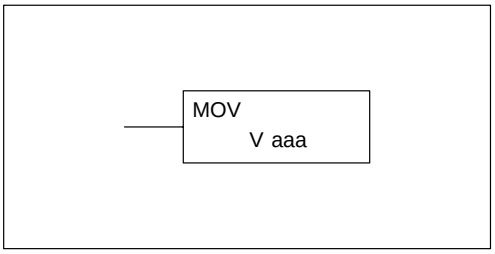
Handheld Programmer Keystrokes

$ 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)

The Move instruction moves the values from a V memory table to another V memory table the same length. 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 Move function.



Step 1:— Load the number of V memory locations to be moved into the first level of the accumulator stack. This parameter must be a HEX value.

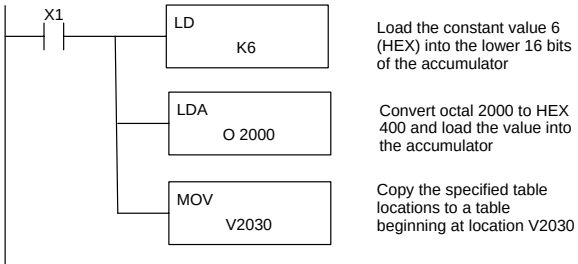
Step 2:— Load the starting V memory location for the locations to be moved into the accumulator. This parameter must be a HEX value.

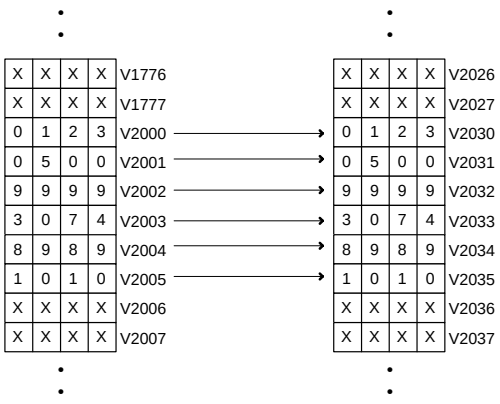
Step 3:— Insert the MOVE instruction which specifies starting V memory location (Vaaa) for the destination table.

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		DL350 Range
		aaa
V memory	V	All (See page 3–29)

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.





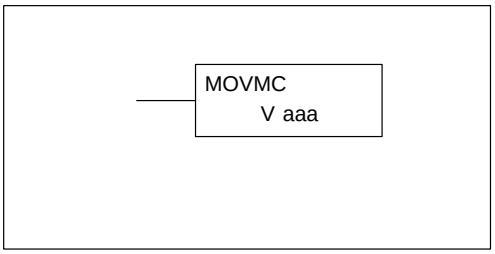
Handheld Programmer Keystrokes

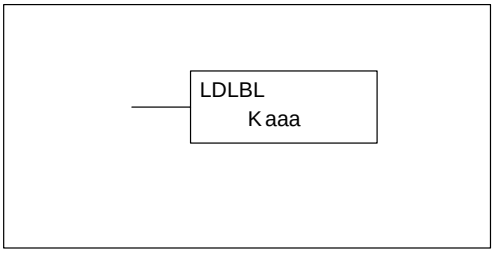
$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	SHFT	K JMP	G 6	ENT	
SHFT	L ANDST	D 3	A 0	→	C 2	A 0	A 0	A 0	ENT
SHFT	M ORST	O INST#	V AND	→	C 2	A 0	D 3	A 0	ENT

Move Memory Cartridge / Load Label (MOVMC) (LDLBL)

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 Move Memory Cartridge and Load Label 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 the program ladder memory and the beginning of the V memory block into the first level of the accumulator stack.

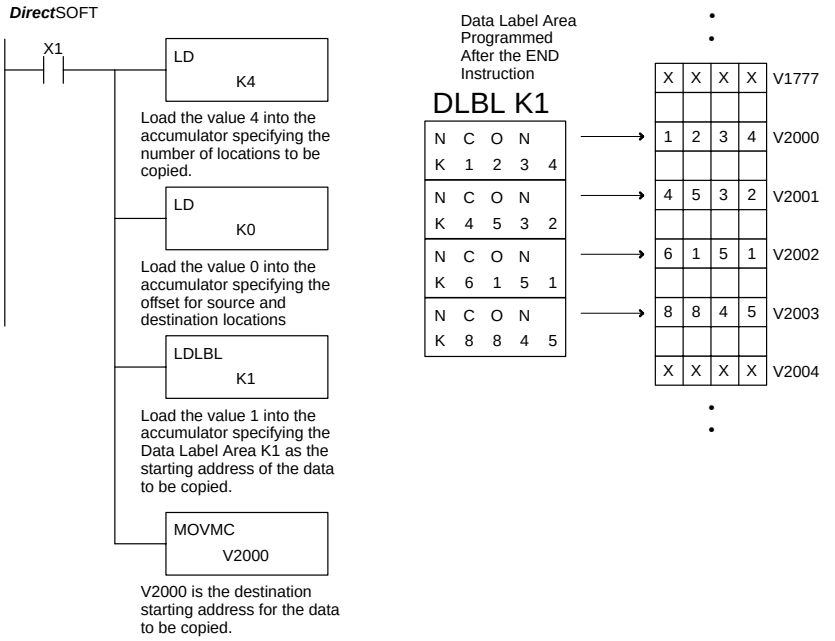
Step 3:— Load the *source data label* (LDLBL Kaaa) into the accumulator when copying data from ladder memory to V memory. Load the *source address* into the accumulator when copying data from V memory to ladder memory. This is where the value will be copied from. If the source address is a V memory location, the value must be entered in HEX.

Step 4:— Insert the MOVMC instruction which specifies destination (Aaaa). This is where the value will be copied to.

Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)

Copy Data From a Data Label Area to V Memory

In the following example, 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 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 using the Load instruction. This value specifies the offset for the source and destination data, and 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.



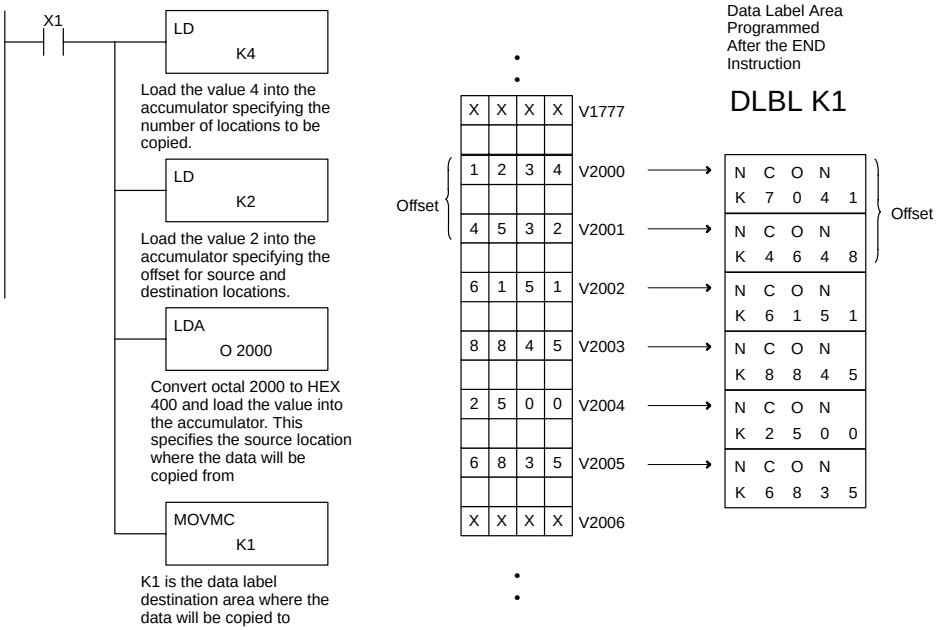
Handheld Programmer Keystrokes

$ 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

Copy Data From V Memory to a Data Label Area

In the following example, data is copied from V memory to a data label area. 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 in the second stack location after the next Load and Load Address instructions are executed. The constant value (K2) is loaded into the accumulator using the Load instruction. This value specifies the offset for the source *and* destination data, and is placed in the first stack location after the Load Address instruction is executed. The source address where data is being copied from is loaded into the accumulator using the Load Address instruction. The MOVMC instruction specifies the destination starting location and executes the copying of data from V memory to the data label area.

DirectSOFT



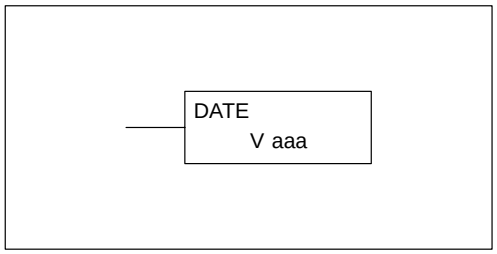
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	ENT			
SHFT	L ANDST	D 3	→	SHFT	K JMP	C 2	ENT			
SHFT	L ANDST	D 3	A 0	→	C 2	A 0	A 0	A 0	ENT	
SHFT	M ORST	O INST#	V AND	M ORST	C 2	→	SHFT	K JMP	B 1	ENT

Clock / Calendar Instructions

Date (DATE)

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).



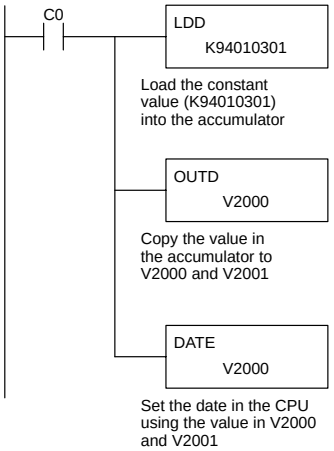
Date	Range	V Memory Location (BCD) (READ Only)
Year	0–99	V7774
Month	1–12	V7773
Day	1–31	V7772
Day of Week	0–06	V7771

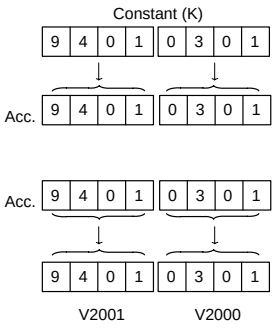
The values entered for the day of week are:
0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday

Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)

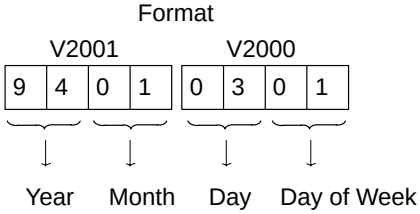
In the following example, when C0 is on, the constant value (K94010301) is loaded into the accumulator using the Load Double instruction (C0 should be a contact from a one shot (PD) instruction). The value in the accumulator is output to V2000 using the Out Double instruction. The Date instruction uses the value in V2000 to set the date in the CPU.

DirectSOFT Display





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)

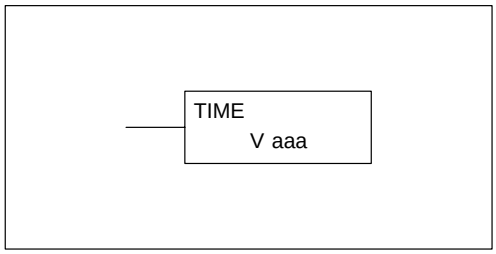


Handheld Programmer Keystrokes

STR	→	C	0	ENT						
SHFT	L	D	D	→	K	9	4	0	1	
0	3	0	1	ENT						
OUT	SHFT	D	→	V	2	0	0	0	ENT	
SHFT	D	A	T	E	→	V	2	0	0	0	ENT

Time (TIME)

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.

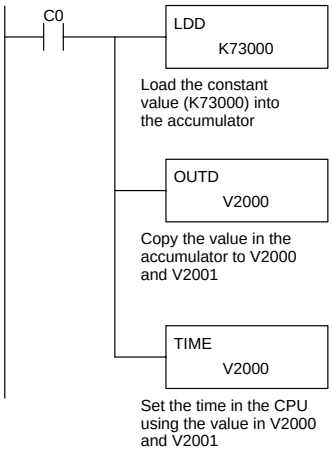


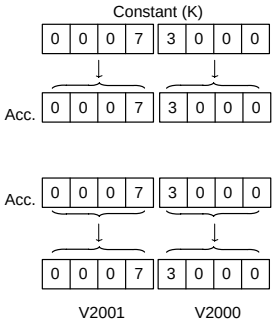
Date	Range	V Memory Location (BCD) (READ Only)
1/100 seconds (10ms)	0–99	V7747
Seconds	0–59	V7766
Minutes	0–59	V7767
Hour	0–23	V7770

Operand Data Type		DL350 Range
	A	**aaa**
Vmemory	V	All (See p. 3–29)

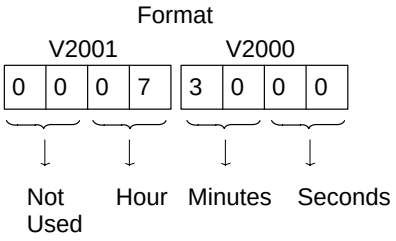
In the following example, when C0 is on, the constant value (K73000) is loaded into the accumulator using the Load Double instruction (C0 should be a contact from a one shot (PD) instruction). The value in the accumulator is output to V2000 using the Out Double instruction. The Time instruction uses the value in V2000 to set the time in the CPU.

***Direct*SOFT Display**





The Time instruction uses the value set in V2000 and V2001 to set the time in the appropriate V memory locations (V7766–V7770)



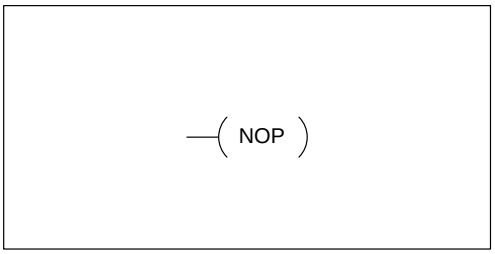
Handheld Programmer Keystrokes

STR	→	C	0	ENT						
SHFT	L	D	D	→	K	7	3	0	0	ENT
OUT	SHFT	D	→	V	2	0	0	0	ENT	
SHFT	T	I	M	E	→	V	2	0	0	0	ENT

CPU Control Instructions

No Operation (NOP)

The No Operation is an empty (not programmed) memory location.



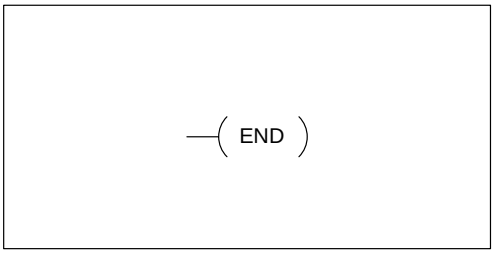
DirectSOFT

Handheld Programmer Keystrokes

SHFT	N TMR	O INST#	P CV	ENT

End (END)

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.



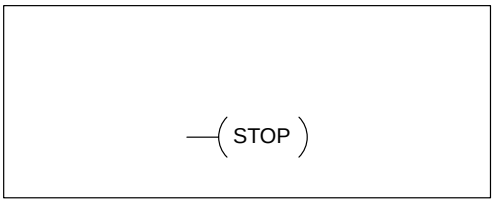
DirectSOFT

Handheld Programmer Keystrokes

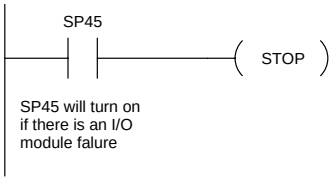
SHFT	E 4	N TMR	D 3	ENT

Stop (STOP)

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 a shutdown condition such as a I/O module failure.



In the following example, when SP45 comes on indicating a I/O module failure, the CPU will stop operation and switch to the program mode.

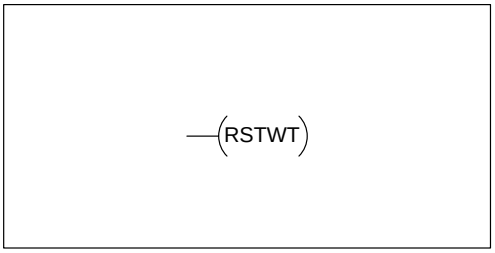


Handheld Programmer Keystrokes

$ STR	→	SHFT	SP STRN	E 4	F 5	ENT
SHFT	S RST	SHFT	T MLR	O INST#	P CV	ENT

Reset Watch Dog Timer (RSTWT)

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.

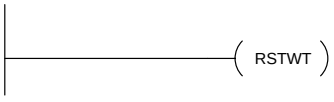


A software timeout error (E003) will occur and the CPU will enter the program mode if the scan time exceeds the watch dog timer setting. Placement of the RSTWT instruction in the program is very important. The instruction has to be executed before the scan time exceeds the watch dog timer's setting.

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.

DirectSOFT



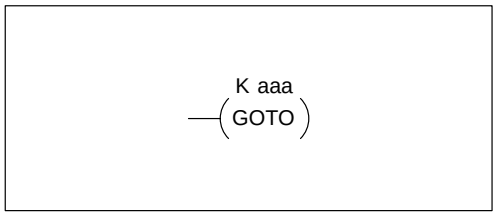
Handheld Programmer Keystrokes

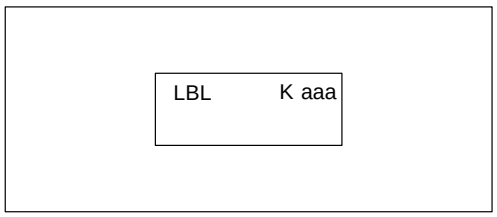
SHFT	R ORN	S RST	T MLR	W ANDN	T MLR	ENT

Program Control Instructions

Goto Label (GOTO) (LBL)

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 128 Goto instructions and 64 LBL instructions can be used in the program.

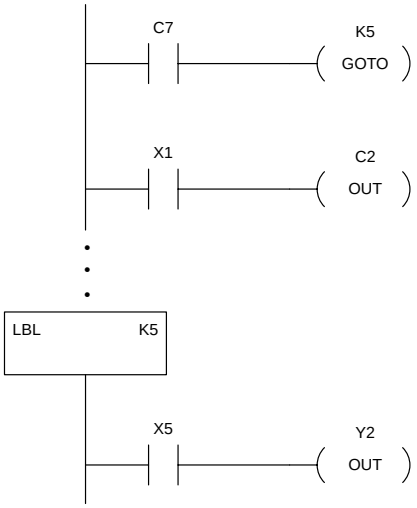




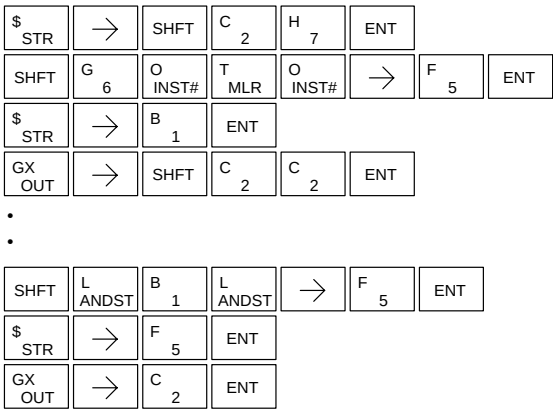
Operand Data Type	DL350 Range
	aaa
Constant K	1–FFFF

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.

DirectSOFT

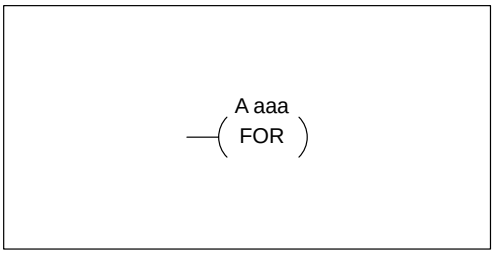


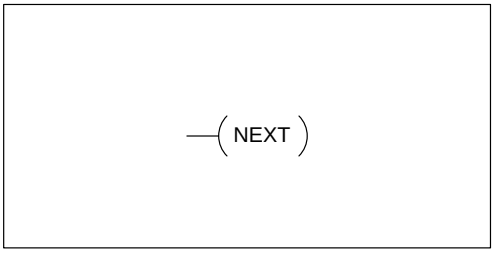
Handheld Programmer Keystrokes



For / Next (FOR) (NEXT)

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.

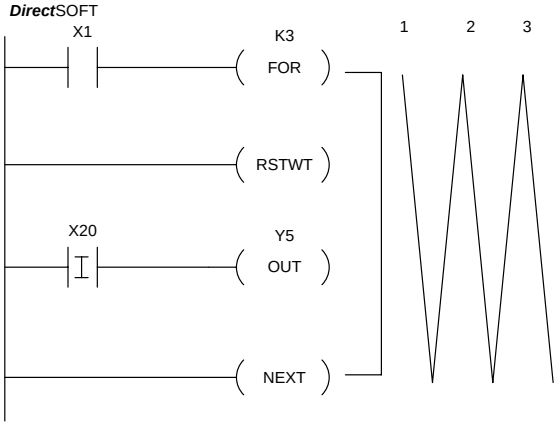




For / Next instructions cannot be nested. Up to 64 For / Next loops may be used in a program. If the maximum number of For / Next loops is exceeded, error E413 will occur. The normal I/O update and CPU housekeeping is suspended while executing the For / Next loop. The program scan can increase significantly, 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		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
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 larger the Watch Dog Timer setting. For more information on the Watch Dog Timer, refer to the RSTWT instruction.



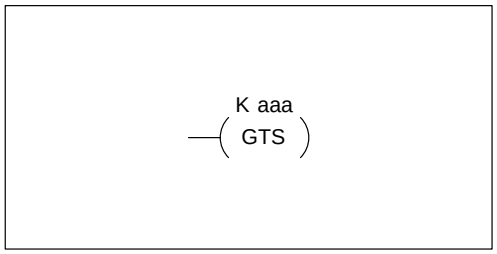
Handheld Programmer Keystrokes

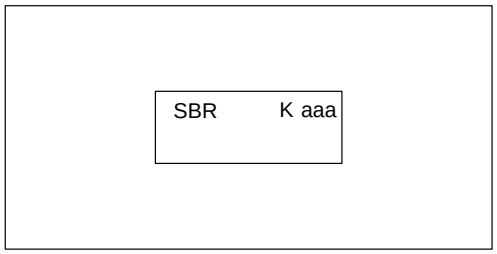
$ STR	→	B 1	ENT			
SHFT	F 5	O INST#	R ORN	→	D 3	ENT
SHFT	R ORN	S RST	T MLR	W ANDN	T MLR	ENT
$ STR	SHFT	I 8	→	C 2	A 0	ENT
GX OUT	→	F 5	ENT			
SHFT	N TMR	E 4	X SET	T MLR	ENT	

Goto Subroutine (GTS) (SBR)

The Goto Subroutine instruction allows a section of ladder logic to be placed outside the main body of the program execute only when needed. There can be a maximum of 128 GTS instructions and 64 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.

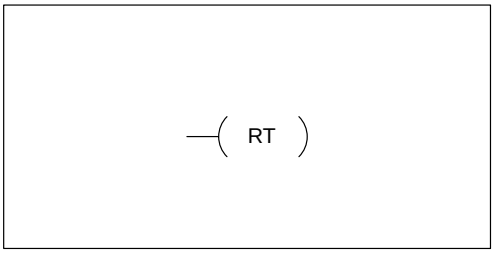




Operand Data Type		DL350 Range
		aaa
Constant	K	1–FFFF

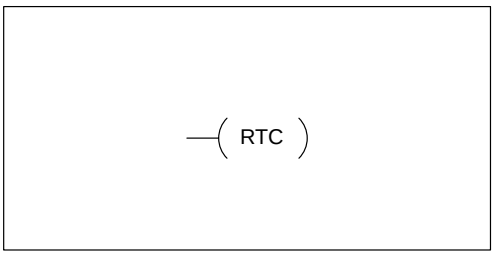
Subroutine Return (RT)

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 which must be the last instruction in the subroutine and is a stand alone instruction (no input contact on the rung).

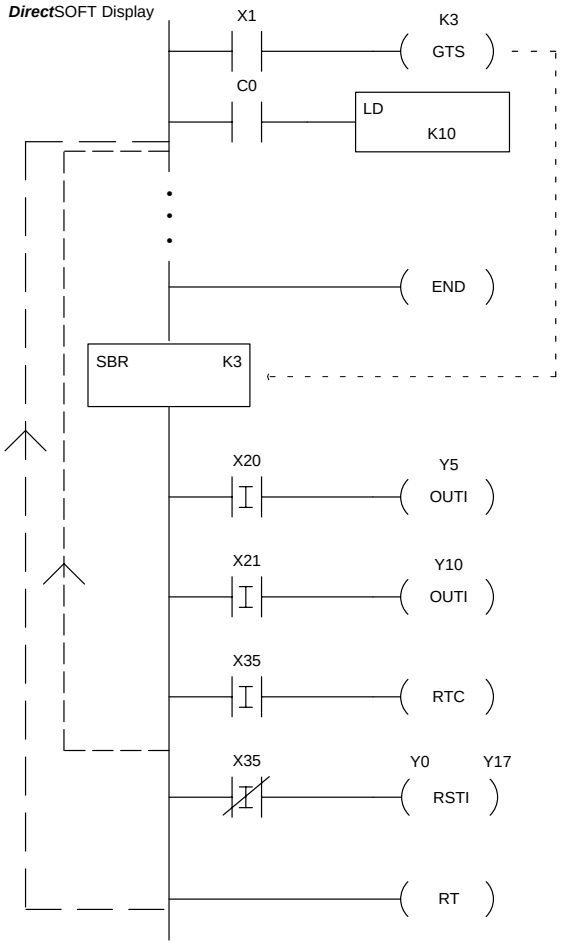


Subroutine Return Conditional (RTC)

The Subroutine Return Conditional instruction is a optional instruction used with a 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 then the CPU will return to the main body of the program.

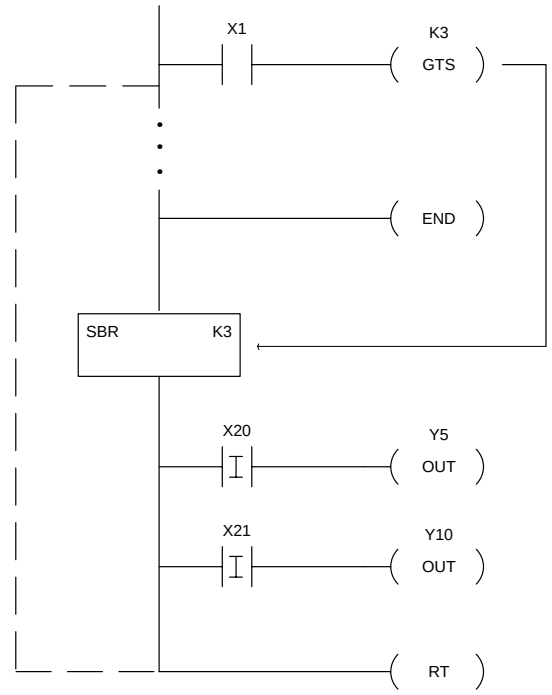


Handheld Programmer Keystrokes

STR	→	X	1	ENT				
SHFT	G	T	S	→	K	3	ENT	
SHFT	E	N	D	ENT				
SHFT	S	SHFT	B	R	→	K	3	ENT
STR	SHFT	I	→	X	2	0	ENT	
OUT	SHFT	I	→	Y	5	ENT		
STR	SHFT	I	→	X	2	1	ENT	
OUT	SHFT	I	→	Y	1	0	ENT	
STR	SHFT	I	→	X	3	5	ENT	
SHFT	R	T	C	ENT				
STRN	SHFT	I	→	X	3	5	ENT	
RST	SHFT	I	→	Y	0	Y	1	7	ENT
SHFT	R	T	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.

DirectSOFT



Handheld Programmer Keystrokes

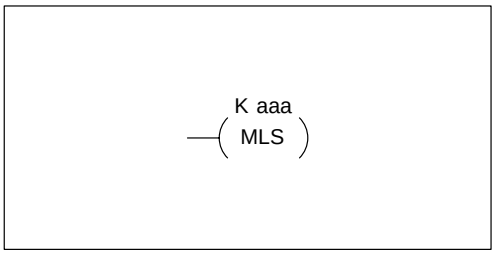
$ STR	→	B 1	ENT				
SHFT	G 6	T MLR	S RST	→	D 3	ENT	

•
•

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)

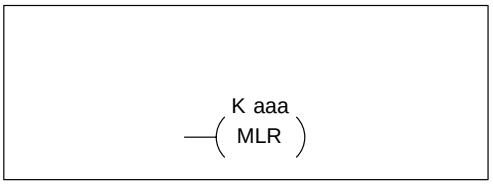
The Master Line Set instruction allows the program to control sections of ladder logic by forming a new power rail controlled by the main left power rail. The main left rail is always master line 0. When a MLS K1 instruction is used, a new power rail is created at level 1. Master Line Sets and Master Line Resets can be used to nest power rails up to seven levels deep. Note that unlike stages in RLL*PLUS,* the logic within the master control relays is still scanned and updated even though it will not function if the MLS is off.



Operand Data Type		DL350 Range
		aaa
Constant	K	1–7

Master Line Reset (MLR)

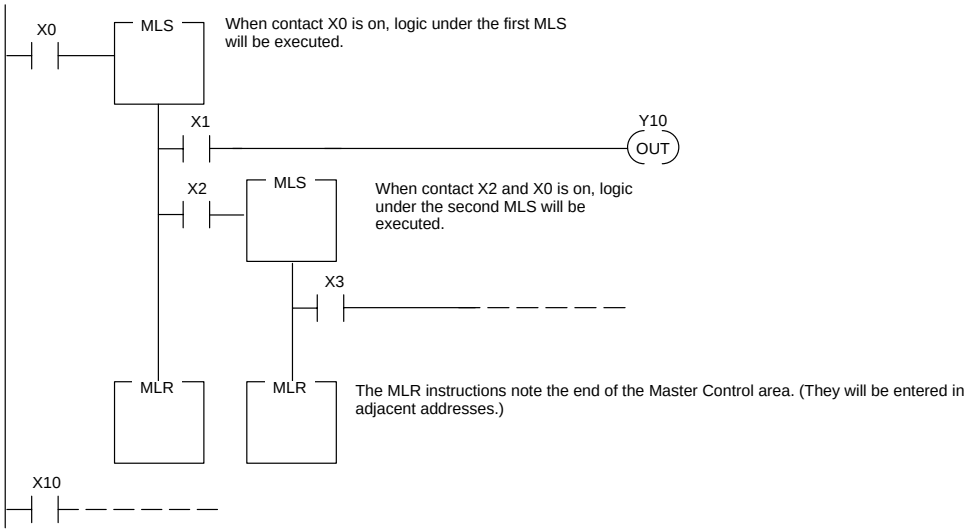
The Master Line Reset instruction marks the end of control for the corresponding MLS instruction. The MLR reference is one less than the corresponding MLS.



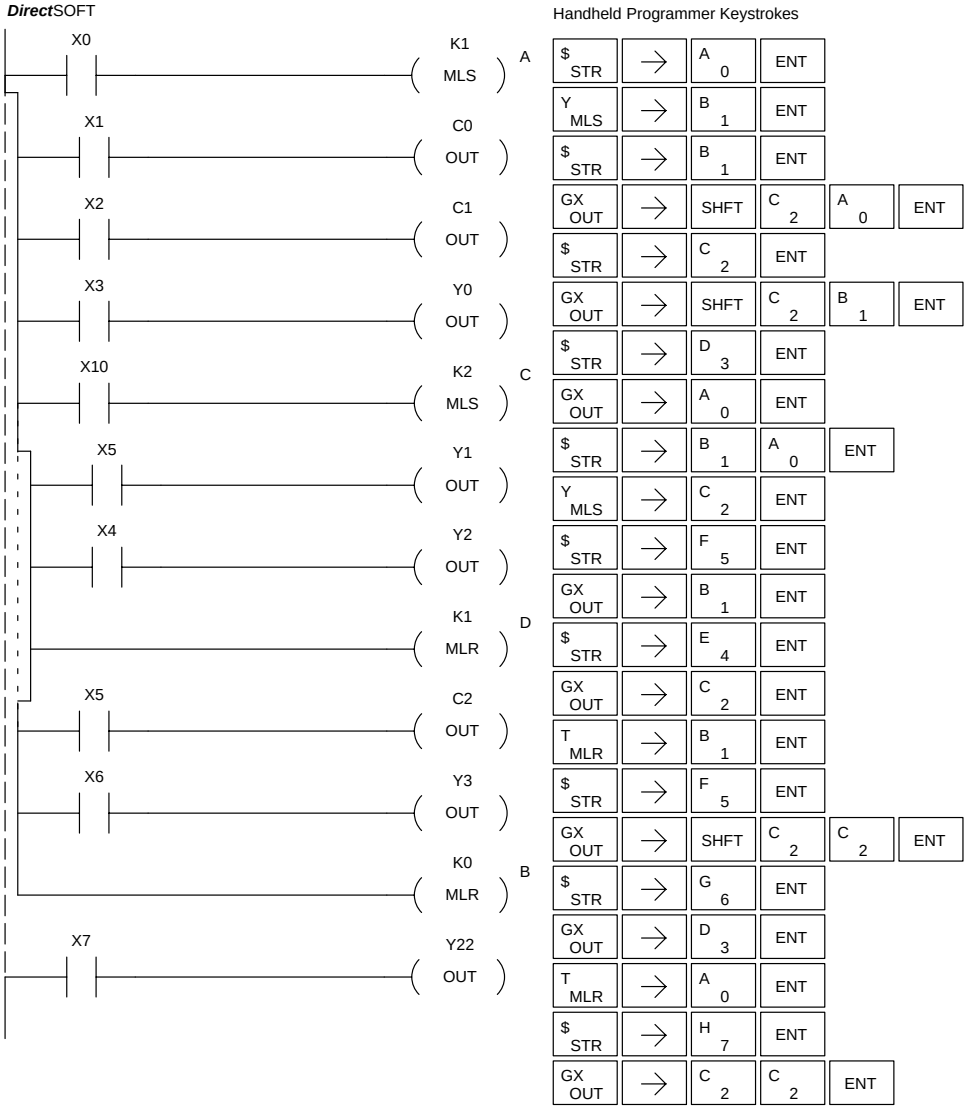
Operand Data Type		DL350 Range
		aaa
Constant	K	0–7

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.



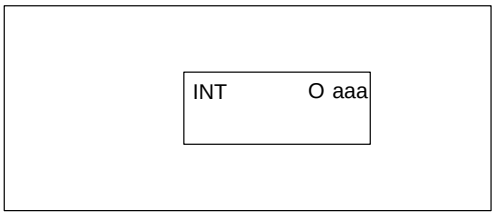
MLS/MLR Example In the following MLS/MLR example logic between the first MLS K1 (A) and MLR K0 (B) will function only if input X0 is on. The logic between the MLS K2 (C) and MLR K1 (D) will function only if input X10 and X0 is on. The last rung is not controlled by either of the MLS coils.



Interrupt Instructions

Interrupt (INT)

The Interrupt instruction allows a section of ladder logic to be placed outside the main body of the program and executed when needed. Interrupts can be called from the program or by external interrupts via the counter interface module which provides 4 interrupts.



Typically, interrupts will be used in an application where a fast response to an input is needed or a program section needs to 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 the interrupt routine is called from the interrupt module or software interrupt, the CPU will complete execution of the instruction it is currently processing in ladder logic then execute the designated interrupt routine. Interrupt module interrupts are labeled in octal to correspond with the hardware input signal (X1 will initiate interrupt INT1). There is only one software interrupt and it is labeled INT 0. The program execution will continue from where it was before the interrupt occurred once the interrupt is serviced.

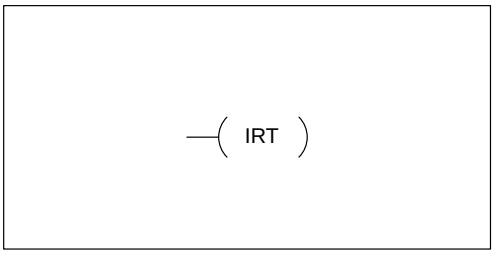
The software interrupt is setup by programming the interrupt time in V7634. The valid range is 3–999 ms. The value must be a BCD value. The interrupt will not execute if the value is out of range.

NOTE: See the example program of a software interrupt.

Operand Data Type		DL350 Range
		aaa
Constant	O	0–3

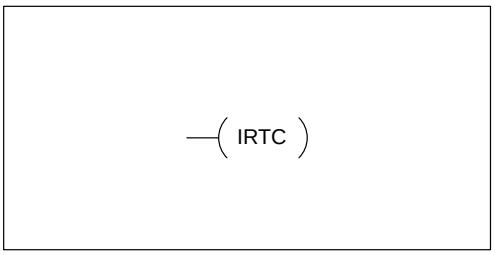
Interrupt Return (IRT)

When an Interrupt Return is executed in the interrupt routine the CPU will return to the point in the main body of the program from which it was called. The Interrupt Return is programmed as the last instruction in an interrupt routine and is a stand alone instruction (no input contact on the rung).



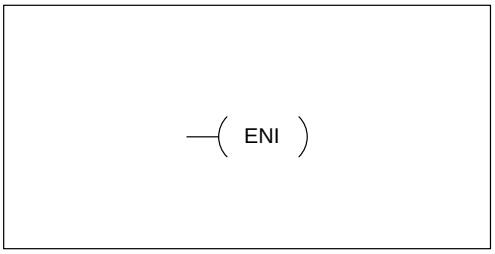
Interrupt Return Conditional (IRTC)

The Interrupt Return Conditional instruction is a optional instruction used with an input contact to implement a condtional return from the interrupt routine. The Interrupt Return is required to terminate the interrupt routine.



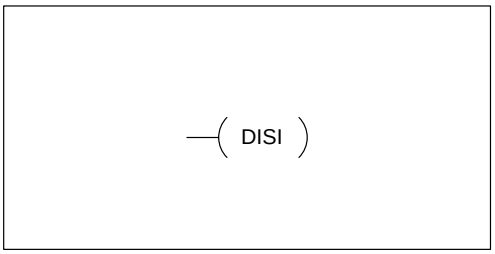
Enable Interrupts (ENI)

The Enable Interrupt instruction is programmed in the main body of the application program (before the End instruction) to enable hardware or software interrupts. Once the coil has been energized interrupts will be enabled until the interrupt is disabled by the Disable Interrupt instruction.



Disable Interrupts (DISI)

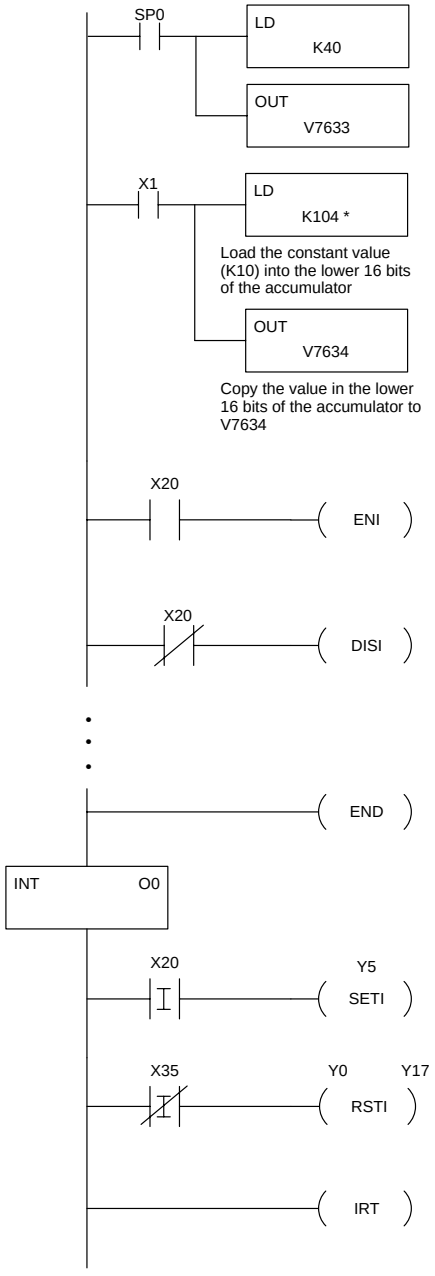
The Disable Interrupt instruction is programmed in the main body of the application program (before the End instruction) to disable both hardware or software interrupts. Once the coil has been energized interrupts will be disabled until the interrupt is enabled by the Enable Interrupt instruction.



Interrupt Example for Software Interrupt

In the following example, when X1 is on, the value 10 is copied to V7634. This value sets the software interrupt to 10 ms. When X20 turns on, the interrupt will be enabled. When X20 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 X35 is not on Y0–Y17 will be reset to off and then the CPU will return to the main body of the program.

***Direct*SOFT**



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 4	A 0	ENT
GX OUT	→	SHFT	V AND	H 7	G 6	D 3	E 4	ENT
$ STR	→	C 2	A 0	ENT				
SHFT	E 4	N TMR	I 8	ENT				
SP STRN	→	C 2	A 0	ENT				
SHFT	D 3	I 8	S RST	I 8	ENT			

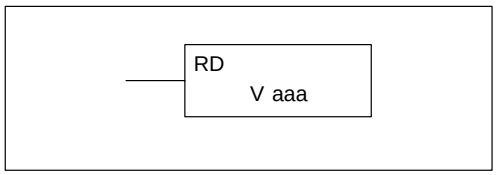
•
•

SHFT	E 4	N TMR	D 3	ENT				
SHFT	I 8	N TMR	T MLR	→	A 0	ENT		
$ STR	SHFT	I 8	→	C 2	A 0	ENT		
X SET	SHFT	I 8	→	F 5	ENT			
SP STRN	SHFT	I 8	→	D 3	F 5	ENT		
X SET	SHFT	I 8	→	B 1	A 0	→	B 1	H 7	ENT
SHFT	I 8	R ORN	T MLR	ENT				

* The value entered, 0–999 must be followed by the digit 4 to complete the instruction.

Intelligent I/O Instructions

Read from Intelligent Module (RD)



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.

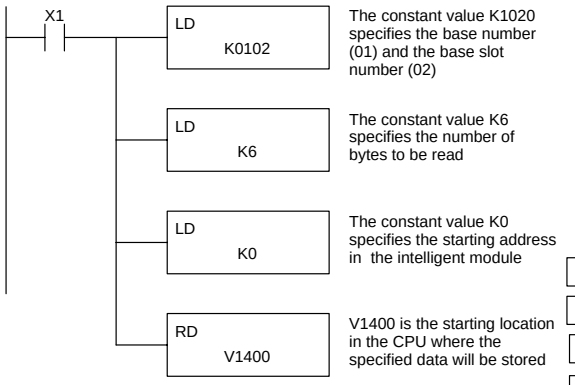
Operand Data Type	DL350 Range
	aaa
Vmemory V	All (See p. 3–29)

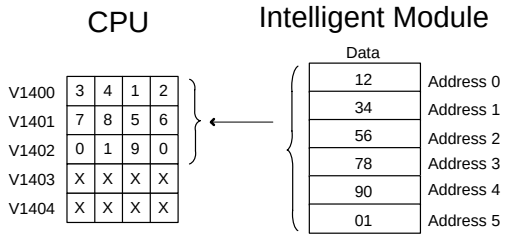
Discrete Bit Flags	Description
SP54	on when RX, WX, RD, WT instructions are executed with the wrong parameters.

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 locations V1400–V1402.

DirectSOFT Display



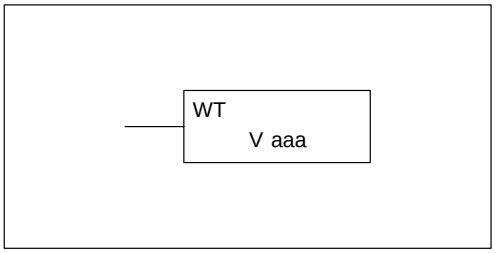


Handheld Programmer Keystrokes

STR	→	X	1	ENT				
SHFT	L	D	K	0	1	0	2	ENT
SHFT	L	D	K	6	ENT			
SHFT	L	D	K	0	ENT			
SHFT	R	D	→	V	1	4	0	0	ENT

Write to Intelligent Module (WT)

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 necessary 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.

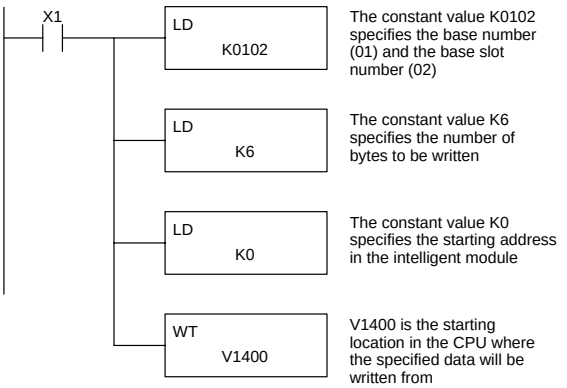
Operand Data Type	DL350 Range
	aaa
Vmemory V	All (See p. 3–29)

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 information from Vmemory locations V1400–V1402.

***Direct*SOFT Display**

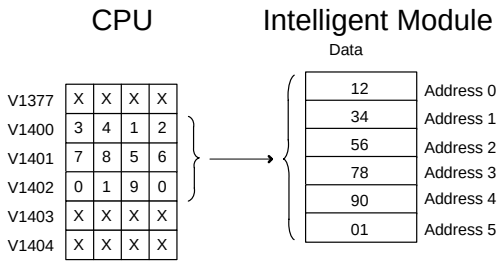


The constant value K0102 specifies the base number (01) and the base slot number (02)

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

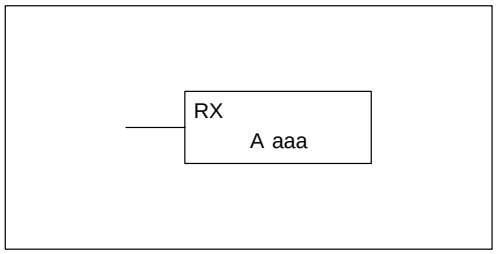


Handheld Programmer Keystrokes

STR	→	X	1	ENT				
SHFT	L	D	K	0	1	0	2	ENT
SHFT	L	D	K	6	ENT			
SHFT	L	D	K	0	ENT			
SHFT	W	T	→	V	1	4	0	0	ENT

Network Instructions

Read from Network (RX)



The Read from Network instruction is used by the master device on a network to read a block of data from another 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 necessary to program the Read from Intelligent module function.

Step 1: — Load the slave address (0–90 BCD) into the first byte and the slot number of the master DCM (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.

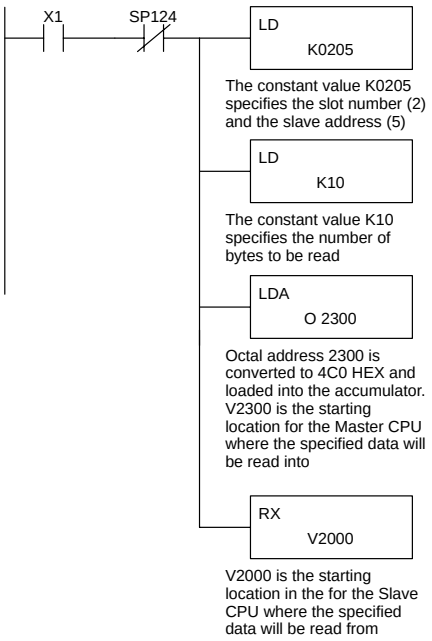
Step 4: — Insert the RX instruction which specifies the starting V memory location (Aaaa) where the data will be read from in the slave.

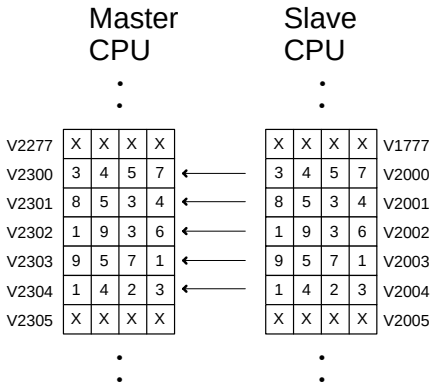
Helpful Hint: — For parameters that require HEX values, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
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	$	0–7679 (7.5K program mem.) 0–15873 (15.5K program mem.)

In the following example, when X1 is on and the module busy relay SP124 (see special relays) is not on, the RX instruction will access a DCM operating as a master in slot 2. 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 DCM.

DirectSOFT

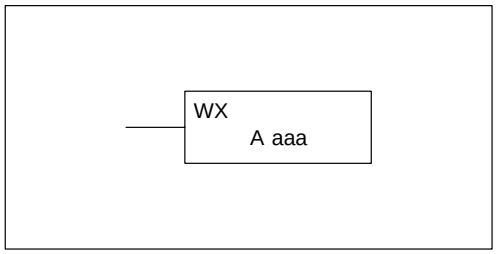




Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
W ANDN	→	SHFT	SP STRN	B 1	C 2	E 4	ENT		
SHFT	L ANDST	D 3	→	SHFT	K JMP	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)



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 first and second level of the accumulator stack and the accumulator by three additional instructions. Listed below are the steps necessary to program the Write to Network function.

Step 1: — Load the slave address (0–90 BCD) into the first byte and the slot number of the master DCM (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 in the master that is to be written to the network into the accumulator. This parameter requires a HEX value.

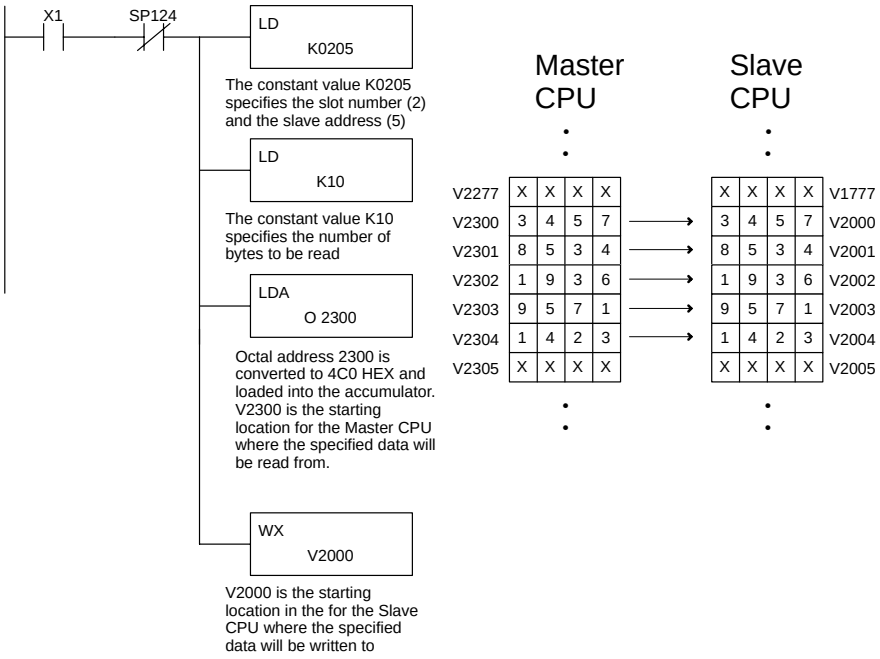
Step 4: — Insert the WX instruction which specifies the starting V memory location (Aaaa) where the data will be written to the slave.

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		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
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	$	0–7679 (7.5K program mem.) 0–15873 (15.5K program mem.)

In the following example when X1 is on and the module busy relay SP124 (see special relays) is not on, the RX instruction will access a DCM operating as a master in slot 2. 10 consecutive bytes of data is read from the CPU at station address 5 and copied to V memory locations V2000–V2004 in the slave CPU.

DirectSOFT



Handheld Programmer Keystrokes

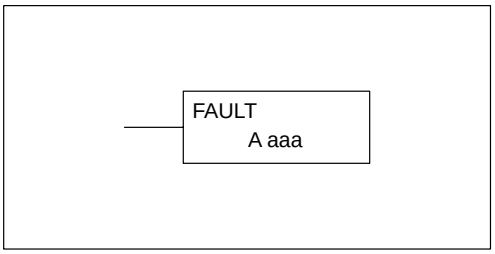
$ STR	→	B 1	ENT							
W ANDN	→	SHFT	SP STRN	B 1	C 2	E 4	ENT			
SHFT	L ANDST	D 3	→	SHFT	K JMP	C 2	A 0	F 5	ENT	
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 1	A 0	ENT		
SHFT	L ANDST	D 3	A 0	→	SHFT	O INST#	C 2	D 3	A 0	A 0	ENT
SHFT	W ANDN	X SET	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT	

Message Instructions

Fault (FAULT)

The Fault instruction is used to display a message on the handheld programmer or ***Direct***SOFT. 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.



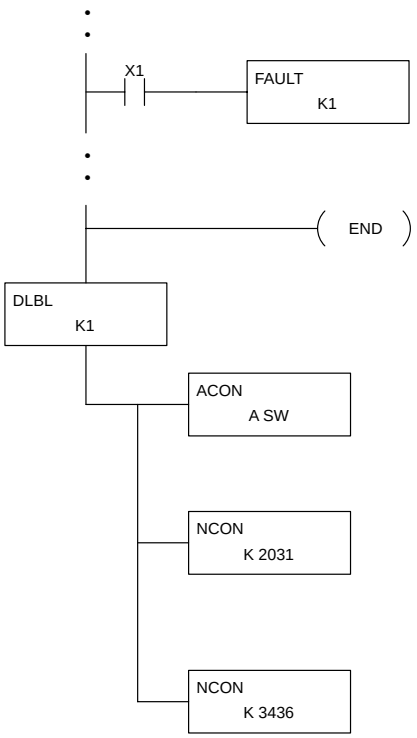
Operand Data Type		DL350 Range
	A	**aaa**
V memory	V	All (See page 3–29)
Constant	K	1–FFFF

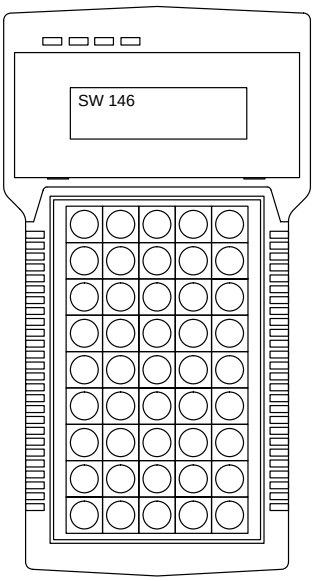
NOTE: The FAULT instruction takes a considerable amount of time to execute. This is because the FAULT parameters are stored in EEPROM. Make sure you consider the instructions execution times (shown in Appendix C) if you are attempting to use the FAULT instructions in applications that require faster than normal execution cycles.

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 ...)

DirectSOFT





Handheld Programmer Keystrokes

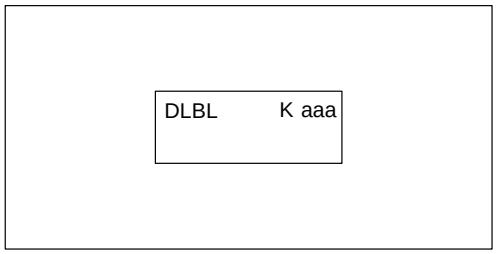
$ STR	→	B 1	ENT					
SHFT	F 5	A 0	U ISG	L ANDST	T MLR	→	B 1	ENT

•
•

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)

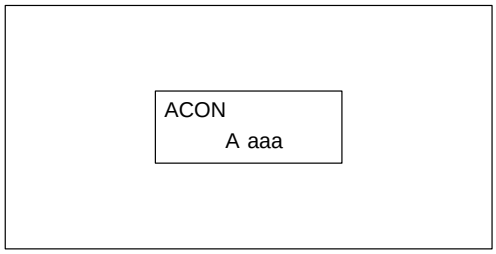
The Data Label instruction marks the beginning of an ASCII / numeric data area. DLBLs are programmed after the End statement. A maximum of 64 DL350 DLBL instructions can be used in a program. Multiple NCONs and ACONs can be used in a DLBL area.



Operand Data Type		DL350 Range
		aaa
Constant	K	1–FFFF

ASCII Constant (ACON)

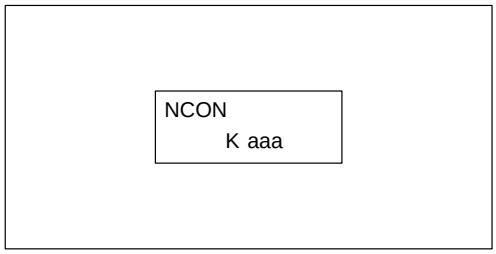
The ASCII Constant instruction is used with the DLBL instruction to store ASCII text for use with other instructions. Two ASCII characters can be stored in an ACON instruction. If only one character is stored in a ACON a leading space will be printed in the Fault message.



Operand Data Type		DL350 Range
		aaa
ASCII	A	0–9 A–Z

Numerical Constant (NCON)

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.

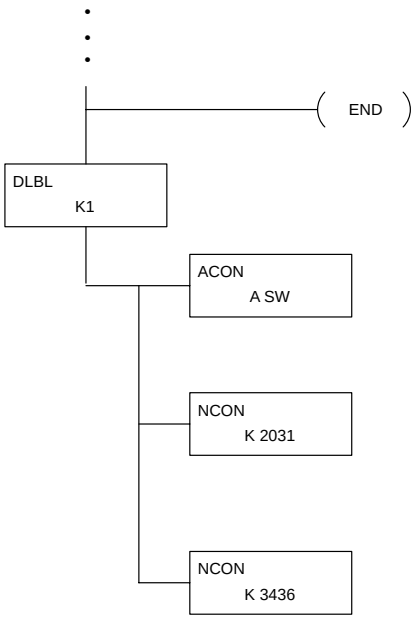


Operand Data Type		DL350 Range
		aaa
Constant	K	0–FFFF

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.

DirectSOFT

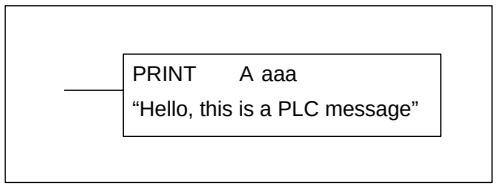


Handheld Programmer Keystrokes

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

Print Message (PRINT)

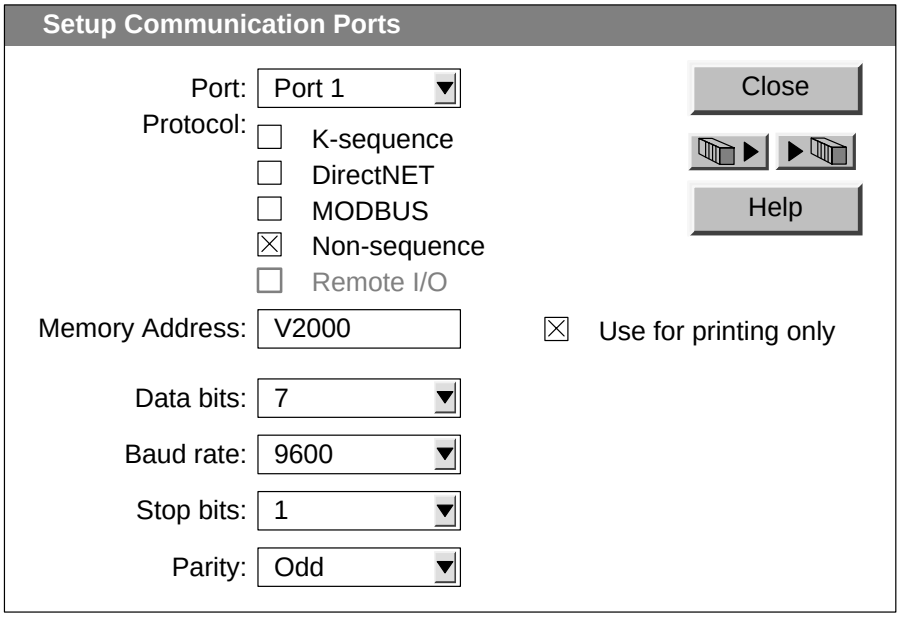
The Print Message instruction prints the embedded text or text/data variable message to Port 2 on the DL350 CPU, which must have the communications port configured.



Data Type		DL350 Range
	A	aaa
Constant	K	1

You may recall from the CPU specifications in Chapter 3 that the DL350’s ports are capable of several protocols. To configure a port 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.



- **Memory Address:** Choose a V-memory address for ***Direct***SOFT to use to store the port setup information. You will need to reserve 9 words in V-memory for this purpose. Select “Use for printing” if you are only using the port for print instructions output.
- **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.

Then click the button indicated to send the Port 2 configuration to the CPU, and click Close. Then see Chapter 3 for port wiring information, in order to connect your printer to the DL350.

Port 2 on the DL350 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

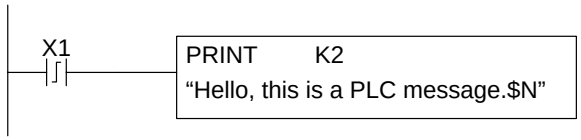
The following examples show various syntax conventions and the length of the output to the printer.

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 all capital letters.

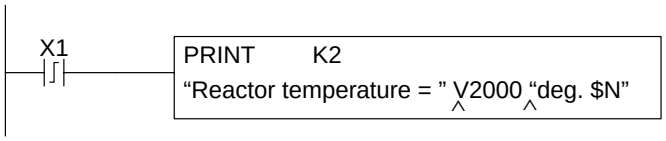
NOTE: There must be a space entered before and after the V-memory address to separate it from the text string. Failure to do this will result in an error code 499.

#	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

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.



Print the message to Port 2 when X1 makes an off-to-on transition.

^ represents a space

Message will read:
Reactor temperature = 0156 deg

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 C00 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 DL350 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.

6

Drum Instruction Programming

In This Chapter. . . .

- — Introduction
- — Step Transitions
- — Overview of Drum Operation
- — Drum Control Techniques
- — Drum Instructions

Introduction

Purpose

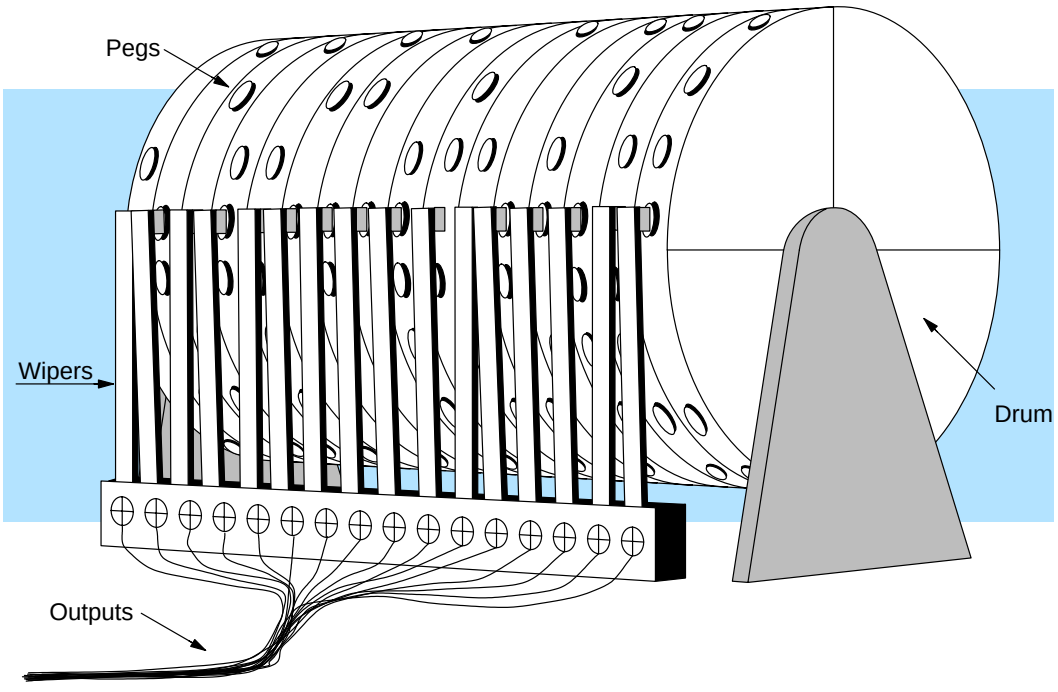
The four drum instructions available in the DL350 CPU electronically simulate an electro-mechanical drum sequencer. The instructions offer slight variations on the basic principle.

Drum Terminology

Drum instructions are best suited for repetitive processes consisting of a finite number of steps. They can do the work of many rungs of ladder logic with simplicity. Therefore, drums can save a programming and debugging time.

We introduce some terminology associated with drum instructions by describing the original electro-mechanical drum pictured below. The mechanical **drum** generally has pegs on its curved surface. The pegs are populated in a particular **pattern**, representing a set of desired actions for machine control. A motor or solenoid rotates the drum a precise amount at specific times. During rotation, stationary wipers sense the presence of pegs (present = on, absent = off). This interaction makes or breaks electrical contact with the wipers, creating electrical **outputs** from the drum. The outputs are wired to devices on a machine for On/Off control.

Drums usually have a finite number of positions within one rotation, called **steps**. Each step represents some process step. At powerup, the drum **resets** to a particular step. The drum rotates from one step to the next based on a **timer**, or on some external **event**. During special conditions, a machine operator can manually increment the drum step using a **jog** control on the drum's drive mechanism. The contact closure of each wiper generates a unique on/off pattern called a **sequence**, designed for controlling a specific machine. Because the drum is circular, it automatically repeats the sequence once per rotation. Applications vary greatly, and a particular drum may rotate once per second, or as slowly as once per week.



Electronic drums provide the benefits of mechanical drums and more. For example, they have a **preset** feature that is impossible for mechanical drums: The preset function lets you move from the present step *directly* to any other step on command!

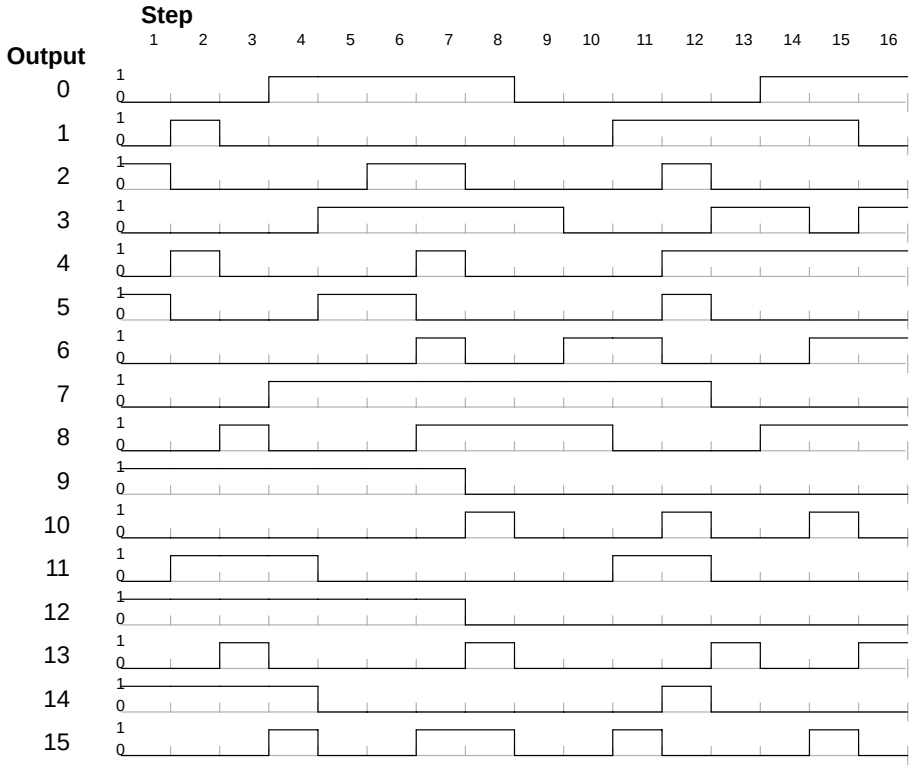
Drum Chart Representation

For editing purposes, the electronic drum is presented in chart form in ***Direct***SOFT and in this manual. Imagine slicing the surface of a hollow drum cylinder between two rows of pegs, then pressing it flat. Now you can view the drum as a chart as shown below. Each row represents a step, numbered 1 through 16. Each column represents an output, numbered 0 through 15 (to match word bit numbering). The solid circles in the chart represent pegs (On state) in the mechanical drum, and the open circles are empty peg sites (Off state).

STEP	OUTPUTS 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	○	●	○	●	○	○	●	○	○	○	●	○	○	●	○	○
2	○	●	○	●	●	○	●	○	○	○	○	●	○	○	●	○
3	○	●	●	●	●	○	●	●	○	○	○	○	○	○	○	○
4	●	●	○	●	●	○	●	○	●	○	○	○	○	○	○	●
5	○	○	○	●	○	○	●	○	●	○	●	○	●	○	○	●
6	○	○	○	●	○	○	●	○	●	○	●	○	●	●	○	●
7	●	○	○	●	○	○	●	●	●	●	○	●	●	●	○	●
8	●	○	●	○	○	●	○	●	●	○	○	○	●	○	○	●
9	○	○	○	○	○	○	○	●	●	○	○	○	●	○	○	○
10	○	○	○	○	○	○	○	●	●	●	○	○	○	○	○	○
11	●	○	○	○	●	○	○	○	○	●	○	○	○	○	●	○
12	○	●	○	○	●	●	○	○	●	○	●	●	○	●	●	○
13	○	○	●	○	○	○	○	○	○	○	○	●	●	○	●	○
14	○	○	○	○	○	○	○	●	○	○	○	●	●	○	●	●
15	●	○	○	○	○	●	○	●	○	●	○	●	○	○	●	●
16	○	○	●	○	○	○	○	●	○	●	○	●	●	○	○	●

Output Sequences

The mechanical drum sequencer derives its name from sequences of control changes on its electrical outputs. The following figure shows the sequence of On/Off controls generated by the drum pattern above. Compare the two, and you will find they are equivalent! If you can see their equivalence, you are on your way to understanding drum instruction operation.



Step Transitions

Drum Instruction Types

There are four types of Drum instructions in the DL350 CPU:

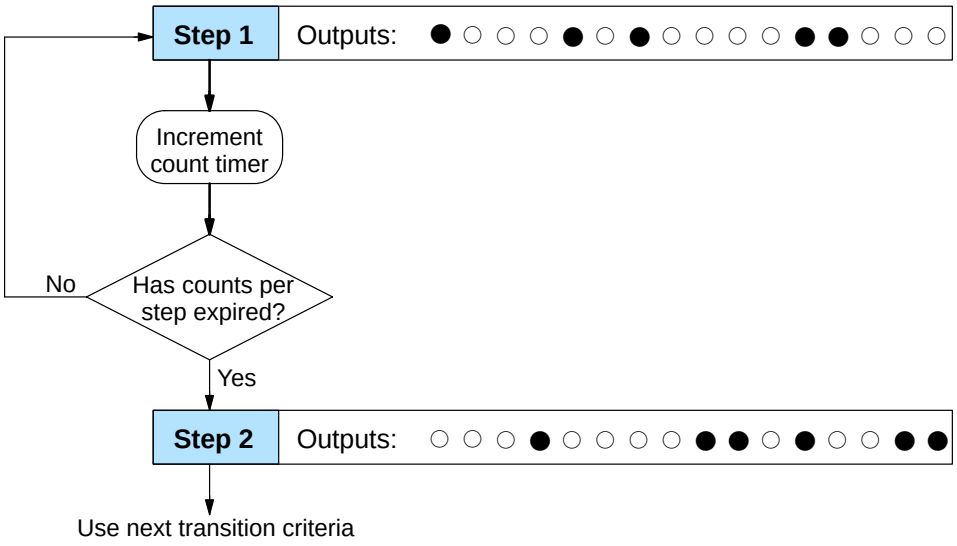
- Timed Drum with Discrete Outputs (DRUM)
- Time and Event Drum with Discrete Outputs (EDRUM)
- Masked Event Drum with Discrete Outputs (MDRUMD)
- Masked Event Drum with Word Output (MDRUMW)

The four drum instructions all include time-based step transitions, and three include event-based transitions as well. Other options include outputs defined as a single word or as individual bits, and an output mask (individual output disable/enable).

Each drum has 16 steps, and each step has 16 outputs. Refer to the figure below. Each output can be either an X, Y, or C coil, offering programming flexibility. We assign Step 1 an arbitrary unique output pattern (○= Off, ●= On) as shown. When programming a drum instruction, you also determine both the output assignment and the On/Off state (pattern) at that time. All steps use the same output assignment, but each step may have its own unique output pattern.

Timer-Only Transitions

Drums move from step to step based on time and/or an external event (input). All four drum types offer timer step transitions, and three types also offer events. The figure below shows how timer-only transitions work.



The drum stays in each step for a specific duration (user-programmable). The timebase of the timer is programmable, from 0.01 seconds to 99.99 seconds. This establishes the resolution, or the duration of each “tick of the clock”. Each step uses the same timebase, but has its own unique counts per step, which you program. The drum spends a specific amount of time in each step, given by the formula:

Time in step = 0.01 seconds X Timebase x Counts per step

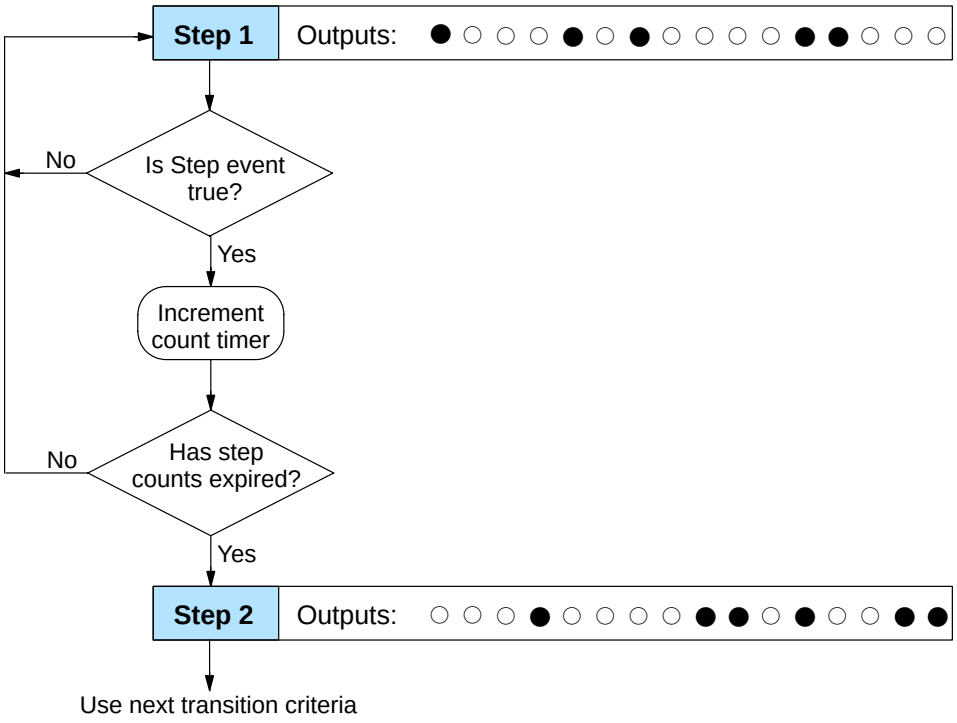
For example, if you program a 5 second time base and 12 counts for Step 1, the drum will spend 60 seconds in Step 1. The maximum time for any step is given by the formula:

Max Time per step = 0.01 seconds X 9999 X 9999
= 999,800 seconds = 277.7 hours = 11.6 days

NOTE: When first choosing the timebase resolution, a good rule is to make it approximately 1/10 the duration of the shortest step in your drum. You will be able to optimize the duration of that step in 10% increments. Other steps with longer durations allow optimizing by even smaller increments (percentage-wise). Also, note the drum instruction executes once per CPU scan. Therefore, it is pointless to specify a drum timebase faster than the CPU scan time.

Timer and Event Transitions

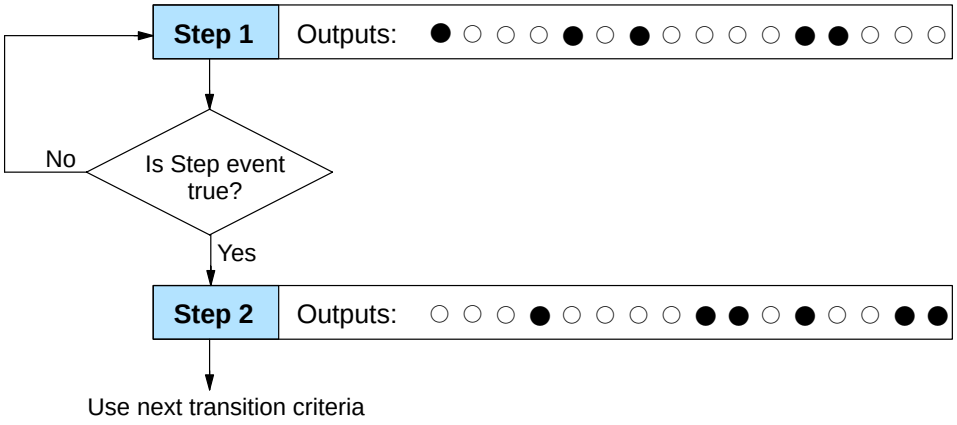
Time and Event Drums move from step to step based on time and/or external events. The figure below shows how step transitions work for these drums.



When the drum enters Step 1, the output pattern shown is set. It begins polling the external input programmed for that step. You can define event inputs as X, Y, or C discrete point types. Suppose we select X0 for the Step 1 event input. If X0 is off, then the drum remains in Step 1. When X0 is On, the event criteria is met and the timer increments. The timer increments as long as the event remains true. When the counts for Step 1 have expired, the drum moves to Step 2. The outputs change immediately to match the new pattern for Step 2.

Event-Only Transitions

Time and Event drums do not have to possess both the event and the timer criteria programmed for each step. You have the option of programming one of the two, and even mixing transition types among all the steps of the drum. For example, you might want Step 1 to transition on an event, Step 2 to transition on time only, and Step 3 to transition on both time and an event. Furthermore, you may elect to use only part of the 16 steps, and only part of the 16 outputs.



Counter Assignments

Each drum instruction uses the resources of four counters in the CPU. When programming the drum instruction, you select the first counter number. The drum also uses the next three counters automatically. The counter bit associated with the first counter turns on when the drum has completed its cycle, going off when the drum is reset. These counter values and counter bit precisely indicate the progress of the drum instruction, and can be monitored by your ladder program.

Suppose you program a timer drum to have 8 steps, and we select CT10 for the counter number (remember, counter numbering is in octal). Counter usage is shown to the right. The right column holds typical values, interpreted below.

Counter Assignments

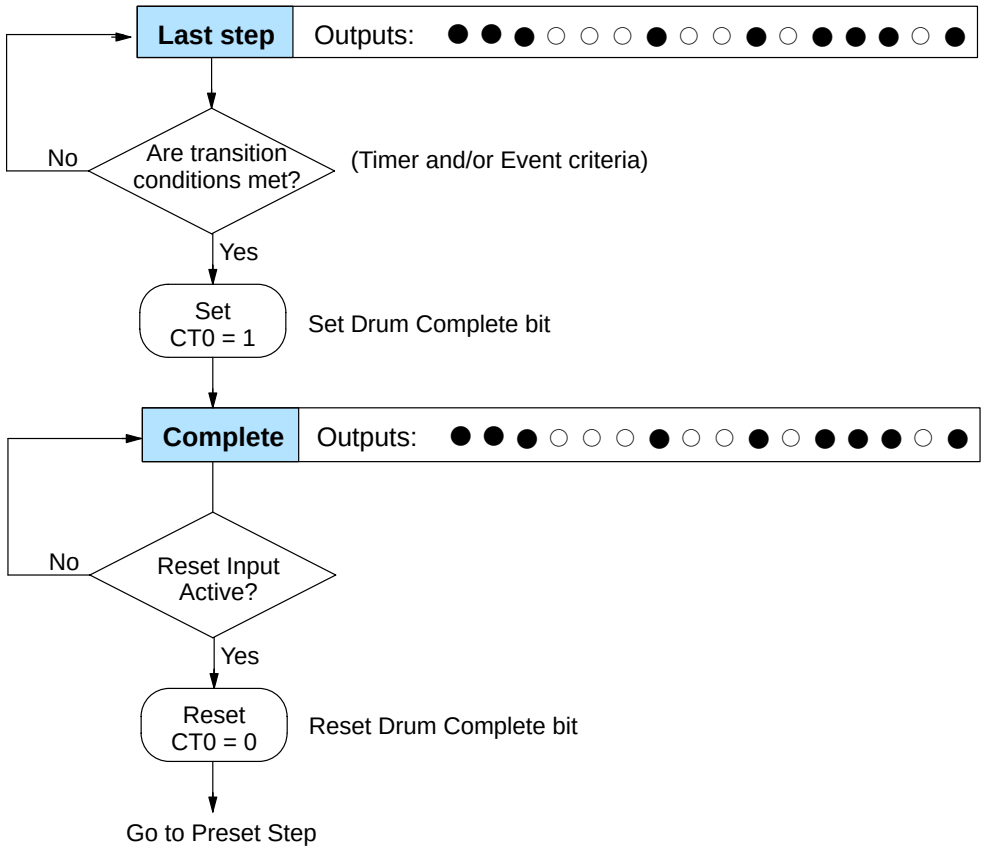
CT10	Counts in step	V1010	1528
CT11	Timer Value	V1011	0200
CT12	Preset Step	V1012	0001
CT13	Current Step	V1013	0004

CT10 shows you are at the 1528th count in the current step, which is step 4 (shown in CT13). If we have programmed step 4 to have 3000 counts, the step is over half completed. CT11 is the count timer, shown in units of 0.01 seconds. So, each least-significant-digit change represents 0.01 seconds. The value of 200 means you have been in the current count (1528) for 2 seconds (0.01 x 100). Finally, CT12 holds the preset step value which was programmed into the drum instruction. When the drum's Reset input is active, it presets to step 1 in this case. The value of CT12 does not change without a program edit. Counter bit CT10 turns on when the drum cycle is complete, and turns off when the drum is reset.

Last Step Completion

The last step in a drum sequence may be any step number, since partial drums are valid. Refer to the following figure. When the transition conditions of the last step are satisfied, the drum sets the counter bit corresponding to the counter named in the drum instruction box (such as CT0). Then it moves to a final “drum complete” state. The drum outputs remain in the pattern defined for the last step (including any output mask logic). Having finished a drum cycle, the Start and Jog inputs have no effect at this point.

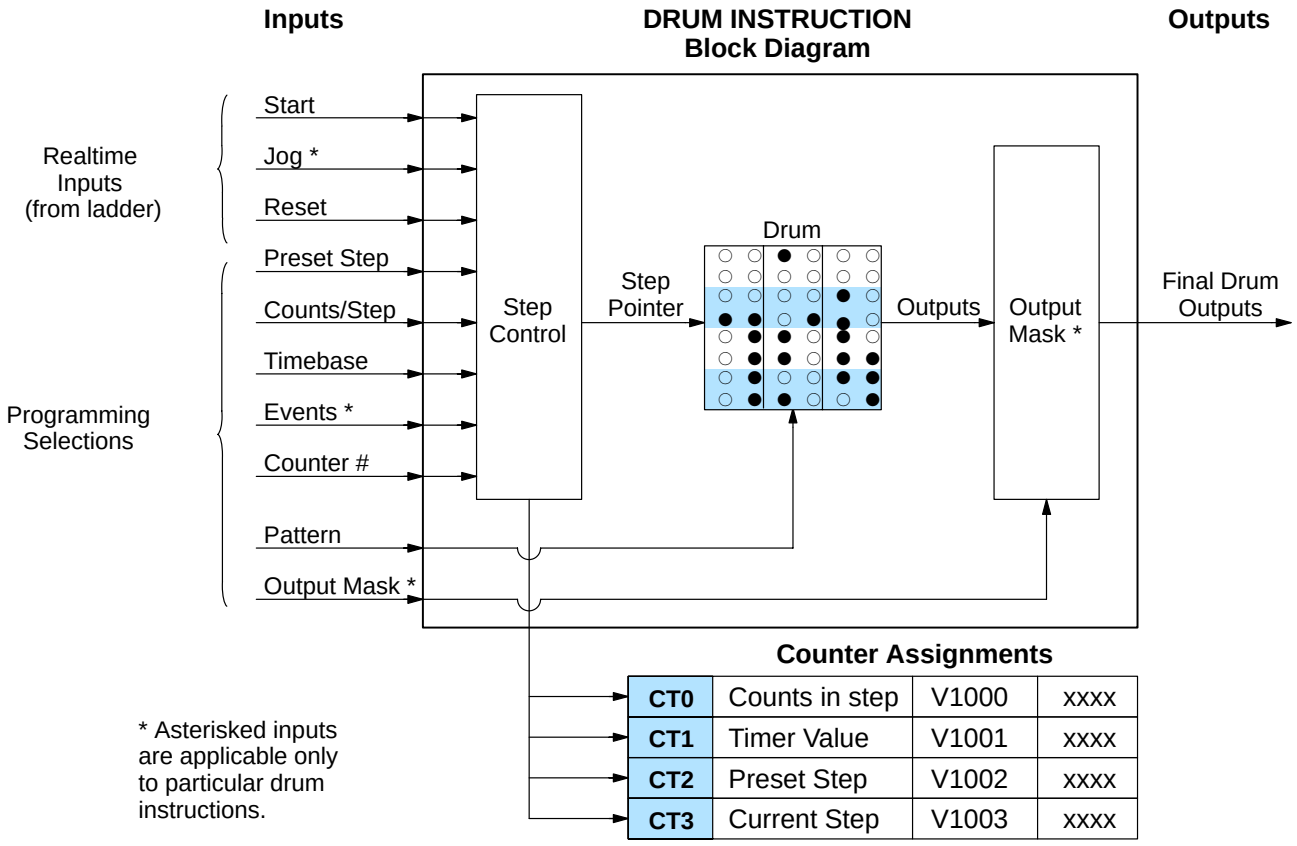
The drum leaves the “drum complete” state when the Reset input becomes active (or on a program-to-run mode transition). It resets the drum complete bit (such as CT0), and then goes directly to the appropriate step number defined as the preset step.



Overview of Drum Operation

Drum Instruction Block Diagram

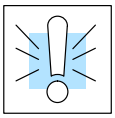
The drum instruction utilizes various inputs and outputs in addition to the drum pattern itself. Refer to the figure below.



The drum instruction accepts several inputs for step control, the main control of the drum. The inputs and their functions are:

- **Start** – The Start input is effective only when Reset is off. When Start is on, the drum timer runs if it is in a timed transition, and the drum looks for the input event during event transitions. When Start is off, the drum freezes in its current state (Reset must remain off), and the drum outputs maintain their current on/off pattern.
- **Jog** – The jog input is only effective when Reset is off (Start may be either on or off). The jog input increments the drum to the next step on each off-to-on transition. Note that only the basic timer drum does not have a jog input.
- **Reset** – The Reset input has priority over the Start input. When Reset is on, the drum moves to its preset step. When Reset is off, then the Start input operates normally.
- **Preset Step** – A step number from 1 to 16 that you define (typically is step 1). The drum moves to this step whenever Reset is on, and whenever the CPU first enters run mode.

- **Counts/Step** – The number of timer counts the drum spends in each step. Each step has its own counts parameter. However, programming the counts/step is optional on Timer/Event drums.
- **Timer Value** – the current value of the counts/step timer.
- **Counter #** – The counter number specifies the first of four consecutive counters which the drum uses for step control. You can monitor these to determine the drum's progress through its control cycle.
- **Events** – Either an X, Y, C, S, C, CT, or SP type discrete point serves as step transition inputs. Each step has its own event. However, programming the event is optional on Timer/Event drums.



WARNING: The outputs of a drum are enabled any time the CPU is in Run Mode. The Start Input **does not** have to be on, and the Reset input does not disable the outputs. Upon entering Run Mode, drum outputs automatically turn on or off according to the pattern of the preset step. This includes any effect of the output mask when applicable.

Powerup State of Drum Registers

The choice of the starting step on powerup and program-to-run mode transitions are important to consider for your application. Please refer to the following chart. If the counter memory is configured as non-retentive, the drum is initialized the same way on every powerup or program-to-run mode transition. However, if the counter memory is configured to be retentive, the drum will stay in its previous state.

Counter Number	Function	Initialization on Powerup	
		Non-Retentive Case	Retentive Case
CT(n)	Current Step Count	Initialize = 0	Use Previous (no change)
CT(n + 1)	Counter Timer Value	Initialize = 0	Use Previous (no change)
CT(n + 2)	Preset Step	Initialize = Preset Step #	Use Previous (no change)
CT(n + 3)	Current Step #	Initialize = Preset Step #	Use Previous (no change)

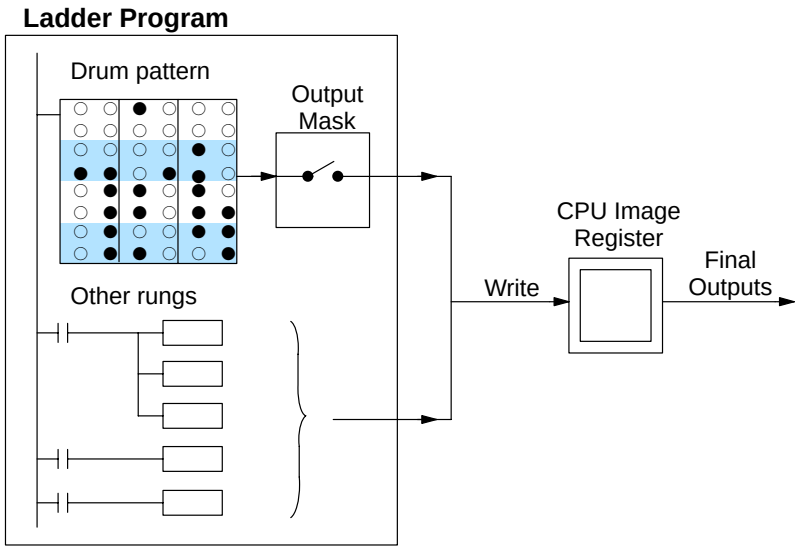
Applications with relatively fast drum cycle times typically will need to be reset on powerup, using the non-retentive option. Applications with relatively long drum cycle times may need to resume at the previous point where operations stopped, using the retentive case. The default option is the retentive case. This means that if you initialize scratchpad V-memory, the memory will be retentive.

Output Mask Operation

Sometimes we need more flexibility in controlling outputs than standard drum output patterns provide. The output mask feature lets you disable drum pattern control of selected outputs on selected steps, allowing those outputs to be controlled by other ladder logic. Two of the four drum instructions have the “output mask” feature:

- **MDRUMD** – Masked Event Drum with Discrete Outputs
- **MDRUMW** – Masked Event Drum with Word Output

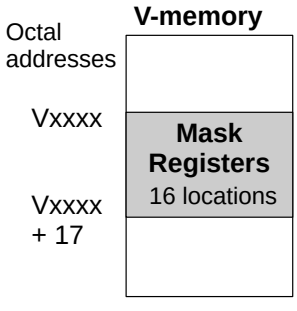
The output mask is simply a bit-by-bit enable/disable control for the drum writing to the image register of the sixteen outputs. Refer to the figure below. The image register contains the official current status of all I/O points. At the end of each PLC scan, the CPU uses the image register status to write to the actual output points.



Practical applications for drum output masking include:

- **Nested Sequence** – a particular output can perform a specialized sequence “inside” a particular step, while the other drum outputs remain static. Rather than consume additional steps, we mask off the output and control it elsewhere in ladder logic during the step duration.
- **Manual Override** – occasionally we need to do manual control of some output(s) in a particular step. Masking the appropriate drum outputs will manual inputs to take over the control through ladder logic.

Each step has its own mask word! Each bit of the word masks the corresponding output point. A 16-register table in V-memory will contain the mask values as shown to the right. In the drum instruction, you specify the starting location of the table. For example, a table which begins at V2000 will extend to V2017. Multiple MDRUMD or MDRUMW drums must have separate mask tables.

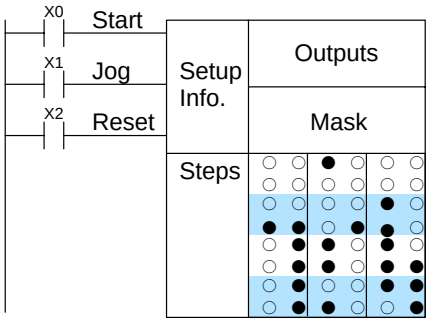


When a mask bit = 1, the drum controls the output point. when the mask bit =0, the drum cannot write to the image register, *so the output remains in its current state.*

Drum Control Techniques

Drum Control Inputs

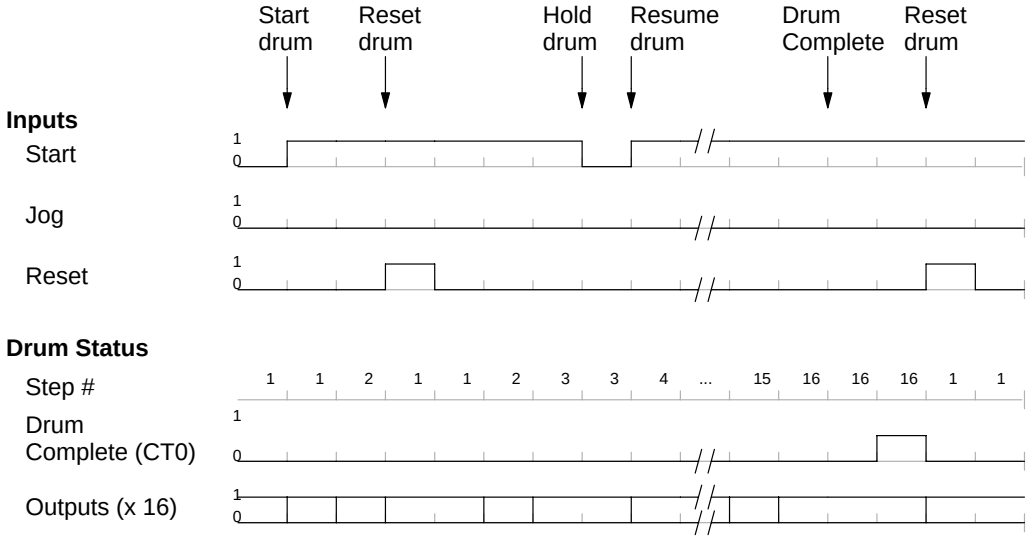
Now we are ready to put together the concepts on the previous pages and demonstrate general control of the drum instruction box. The drawing to the right shows a simplified generic drum instruction. Inputs from ladder logic control the Start, Jog, and Reset Inputs. The first counter bit of the drum (CT0, for example) indicates the drum cycle is done.



The timing diagram below shows an arbitrary timer drum input sequence and how the drum responds. As the CPU enters run mode it initializes the step number to the preset step number (typically is Step 1). When the Start input goes high the drum begins running, looking for an event and/or running the count timer (depending on the drum type and setup).

After the drum enters Step 2, Reset turns On while Start is still On. Since Reset has priority over Start, the drum goes to the preset step (Step 1). Note the drum is *held* in the preset step during Reset, and that step *does not run* (respond to events or run the timer) until Reset turns off.

After the drum has entered step 3, the Start input goes off momentarily, halting the drum's timer until Start turns on again.

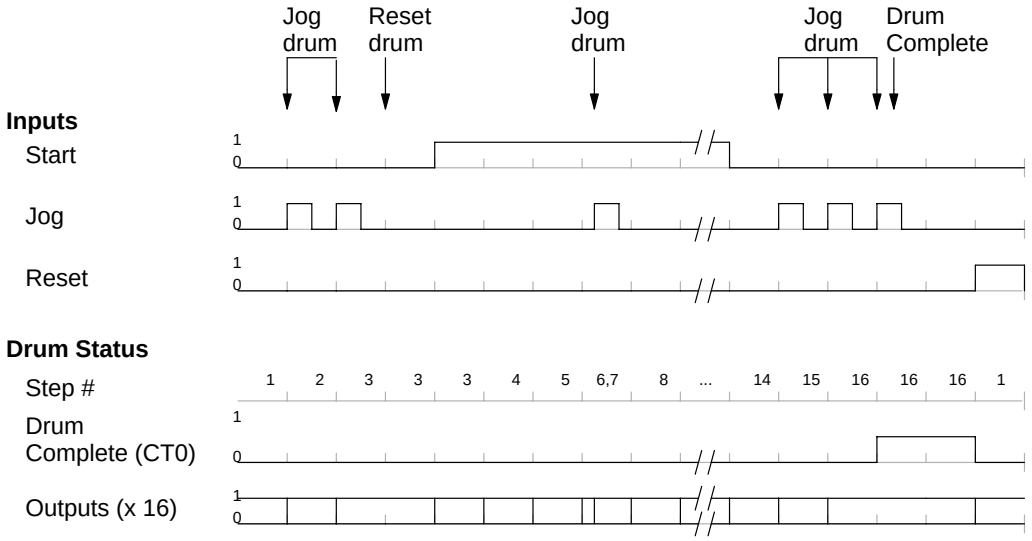


When the drum completes the last step (Step 16 in this example), the Drum Complete bit (CT0) turns on, and the step number remains at 16. When the Reset input turns on, it turns off the Drum Complete bit (CT0), and forces the drum to enter the preset step.

NOTE: The timing diagram shows all steps using equal time durations. Step times can vary greatly, depending on the counts/step programmed.

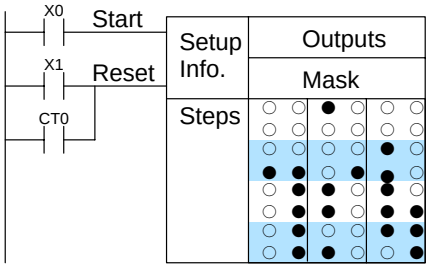
In the figure below, we focus on how the Jog input works on event drums. To the left of the diagram, note the off-to-on transitions of the Jog input increments the step. Start may be either on or off (however, Reset must be off). Two jogs takes the drum to step three. Next, the Start input turns on, and the drum begins running normally. During step 6 another Jog input signal occurs. This increments the drum to step 7, setting the timer to 0. The drum begins running immediately in step 7, because Start is already on. The drum advances to step 8 normally.

As the drum enters step 14, the Start input turns off. Two more Jog signals moves the drum to step 16. However, note that a third Jog signal is required to move the drum through step 16 to “drum complete”. Finally, a Reset input signal arrives which forces the drum into the preset step and turns off the drum complete bit.



Self-Resetting Drum

Applications often require drums that automatically start over once they complete a cycle. This is easily accomplished, using the drum complete bit. In the figure to the right, the drum instruction setup is for CT0, so we logically OR the drum complete bit (CT0) with the Reset input. When the last step is done, the drum turns on CT0 which resets itself to the preset step, also resetting CT0. Contact X1 still works as a manual reset.



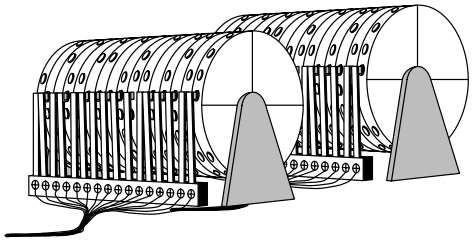
Initializing Drum Outputs

The outputs of a drum are enabled any time the CPU is in run mode. On program-to-run mode transitions, the drum goes to the preset step, and the outputs energize according to the pattern of that step. If your application requires all outputs to be off at powerup, there are two approaches:

- Make the preset step in the drum a “reset step”, with all outputs off.
- Or, use a drum with an output mask. Initialize the mask to “0000” on the first scan using contact SP0, and LD K000 and OUT Vxxx instructions, where Vxxxx is the location of the mask register.

Cascaded Drums Provide More Than 16 Steps

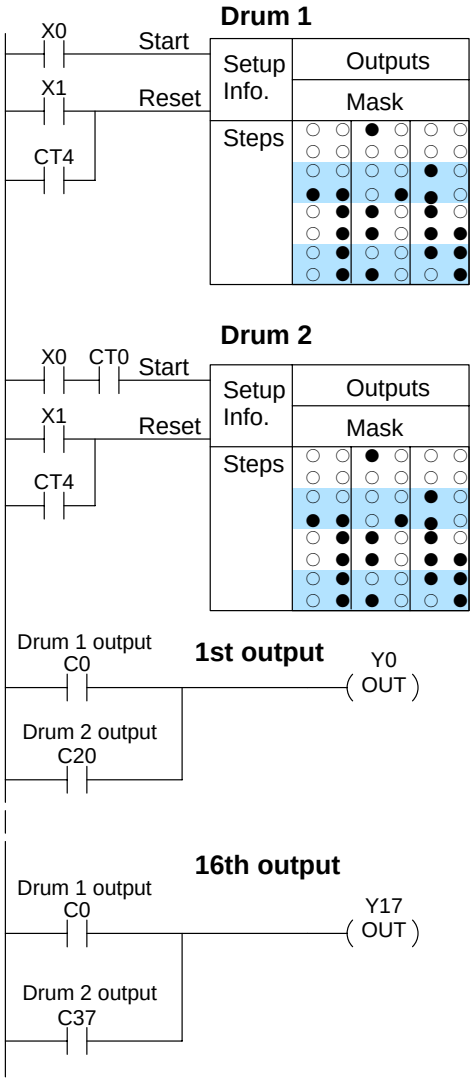
Occasionally the need arises for a drum with more than 16 steps. The solution is to use two or more drums that are logically cascaded. When the first drum finishes, the second one starts, and so on. Remember that a drum instruction writes to the outputs on every scan, even when its start input is off. So, two drums using the same output points will be in conflict. The solution for this is to use separate control relays contacts (CRs) for each drum's outputs, and logically OR them together to control the final outputs.



Refer to the figure to the right. The two drums behave as one 32–step drum. The procedure is:

- Use the drum cycle done bit of the first drum for the start input of the next drum (CT0 in the example).
- Use the last drum's cycle done bit for the reset input of all drums (CT4 in the example).
- OR a manual reset contact with the reset contact above, if needed (is X1 in the example).
- Use the same V-memory address for the output mask of both drums, if your drum application requires a mask.
- Use different control relay (CR) output coils for each drum, but OR them together in ladder logic as shown.

Now, Y0 is the final output from the combined drums. Note each drum must have an "idle" step in which its CR outputs are off, while the other drum(s) operate (will typically be step 1).

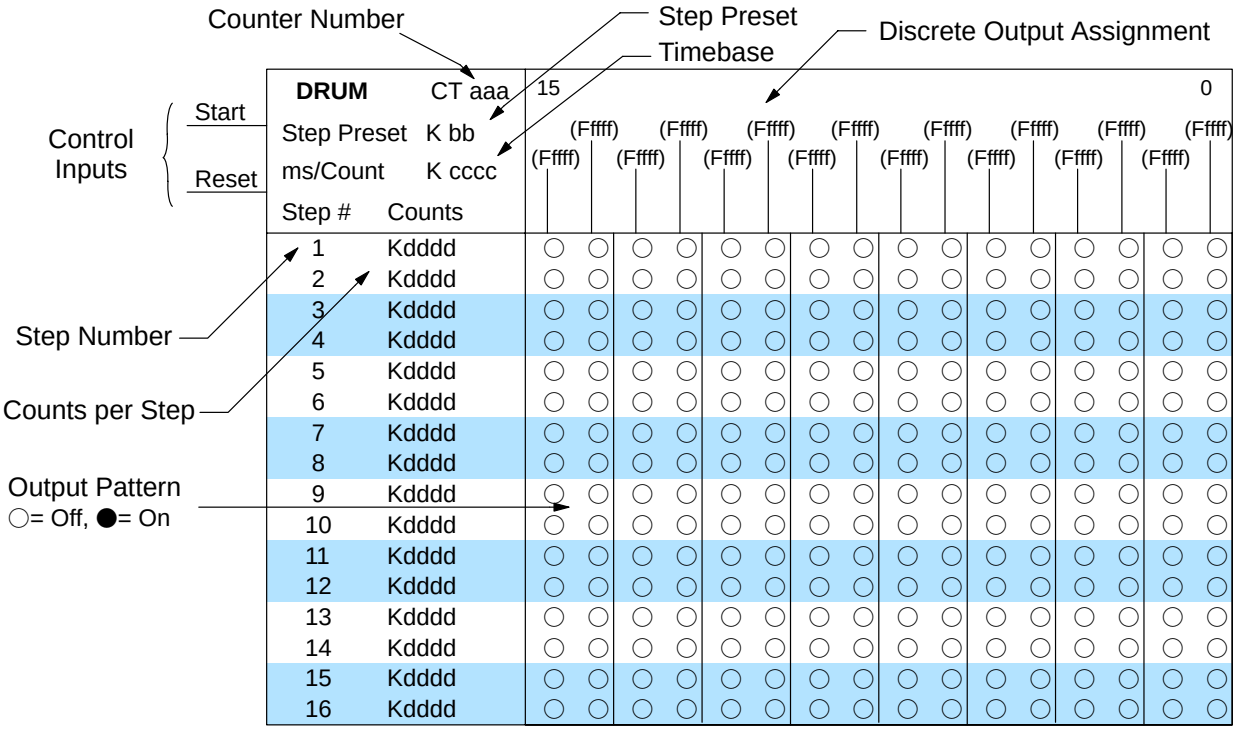


Drum Instructions

The DL350 drum instructions may be programmed using ***Direct***SOFT or for the EDRUM instruction only you can use a handheld programmer (firmware version v1.8 or later. This section covers entry using ***Direct***SOFT for all instructions plus the handheld mnemonics for the EDRUM instruction.

Timed Drum with Discrete Outputs (DRUM)

The Timed Drum with Discrete Outputs is the most basic of the DL350's drum instructions. It operates according to the principles covered on the previous pages. Below is the instruction in chart form as displayed by ***Direct***SOFT.



The Timed Drum features 16 steps and 16 outputs. Step transitions occur only on a timed basis, specified in counts per step. Unused steps must be programmed with "counts per step" = 0 (this is the default entry). The discrete output points may be individually assigned as X, Y, or C types, or may be left unused. The output pattern may be edited graphically with ***Direct***SOFT.

Whenever the Start input is energized, the drum's timer is enabled. It stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

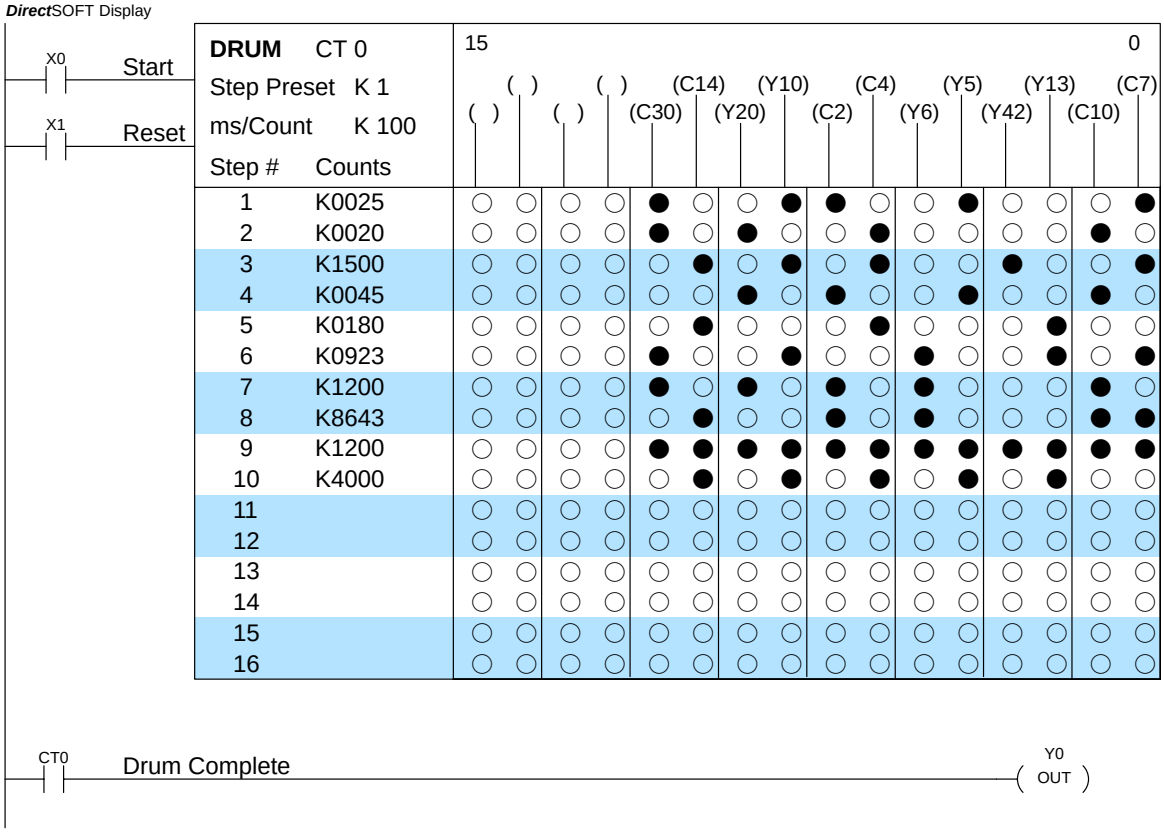
Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa	–	0 – 177
Preset Step	bb	K	1 – 16
Timer base	cccc	K	0 – 99.99 seconds
Counts per step	dddd	K	0 – 9999
Discrete Outputs	Fffff	X, Y, C	see page 3–29

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CT(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CT(n)	0 – 124	Counts in step	CTn = Drum Complete
CT(n+1)	1 – 125	Timer value	CT(n+1) = (not used)
CT(n+2)	2 –126	Preset Step	CT(n+2) = (not used)
CT(n+3)	3 –127	Current Step	CT(n+1) = (not used)

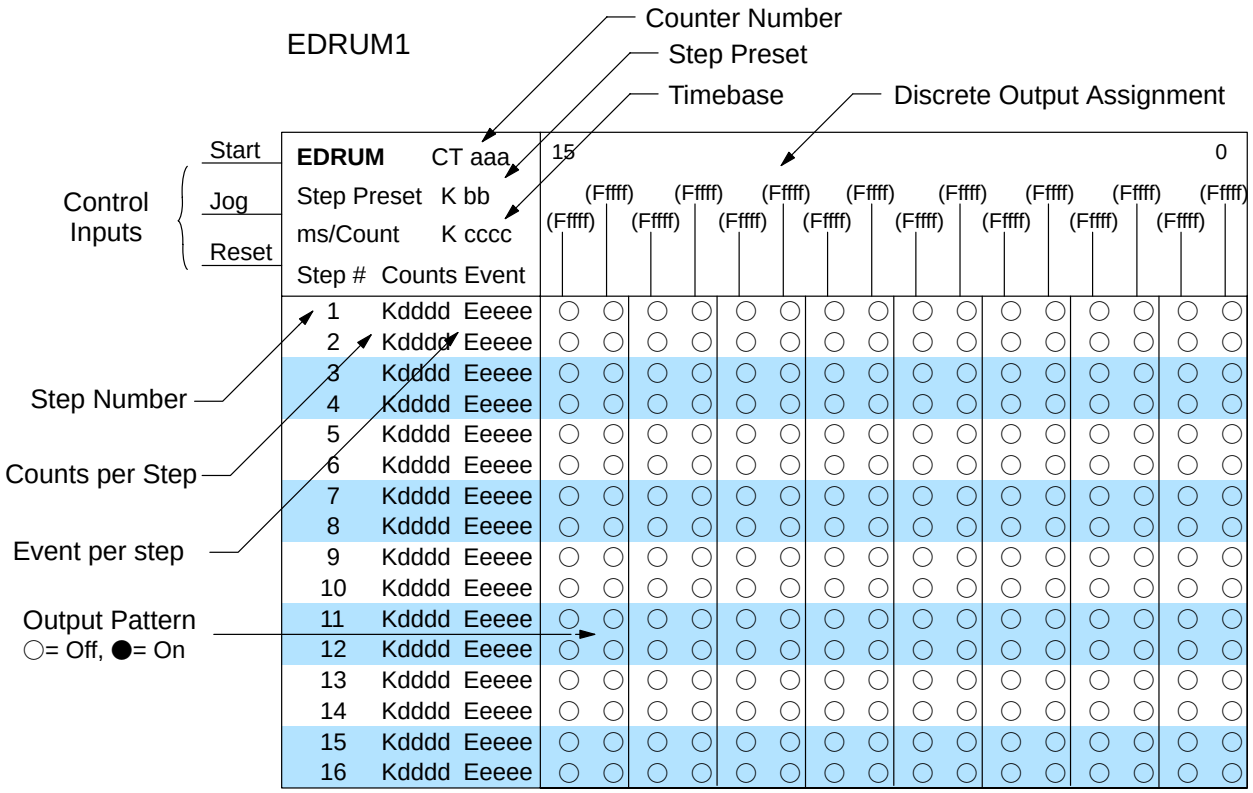
The following ladder program shows the DRUM instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 10 are used, and twelve of the sixteen output points are used. The preset step is step 1. The timebase runs at 100 mS per count. Therefore, the duration of step 1 is (25 x 0.1) = 2.5 seconds. In the last rung, the Drum Complete bit (CT0) turns on output Y0 upon completion of the last step (step 10). A drum reset also resets CT0.

DirectSOFT Display



Event Drum with Discrete Outputs (EDRUM)

The Event Drum with Discrete Outputs has all the features of the Timed Drum, plus event-based step transitions. It operates according to the general principles of drum operation covered in the beginning of this section. Below is the instruction in chart form as displayed by ***Direct***SOFT.



The Event Drum with Discrete Outputs features 16 steps and 16 outputs. Step transitions occur on timed and/or event basis. The jog input also advances the step on each off-to-on transition. Time is specified in counts per step, and events are specified as discrete contacts. Unused steps must be programmed with “counts per step” = 0, and event = “0000”. The discrete output points may be individually assigned. The output pattern may be edited graphically with ***Direct***SOFT.

Whenever the Start input is energized, the drum’s timer is enabled. As long as the event is true for the current step, the timer runs during that step. When the step count equals the counts per step, the drum transitions to the next step. This process stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

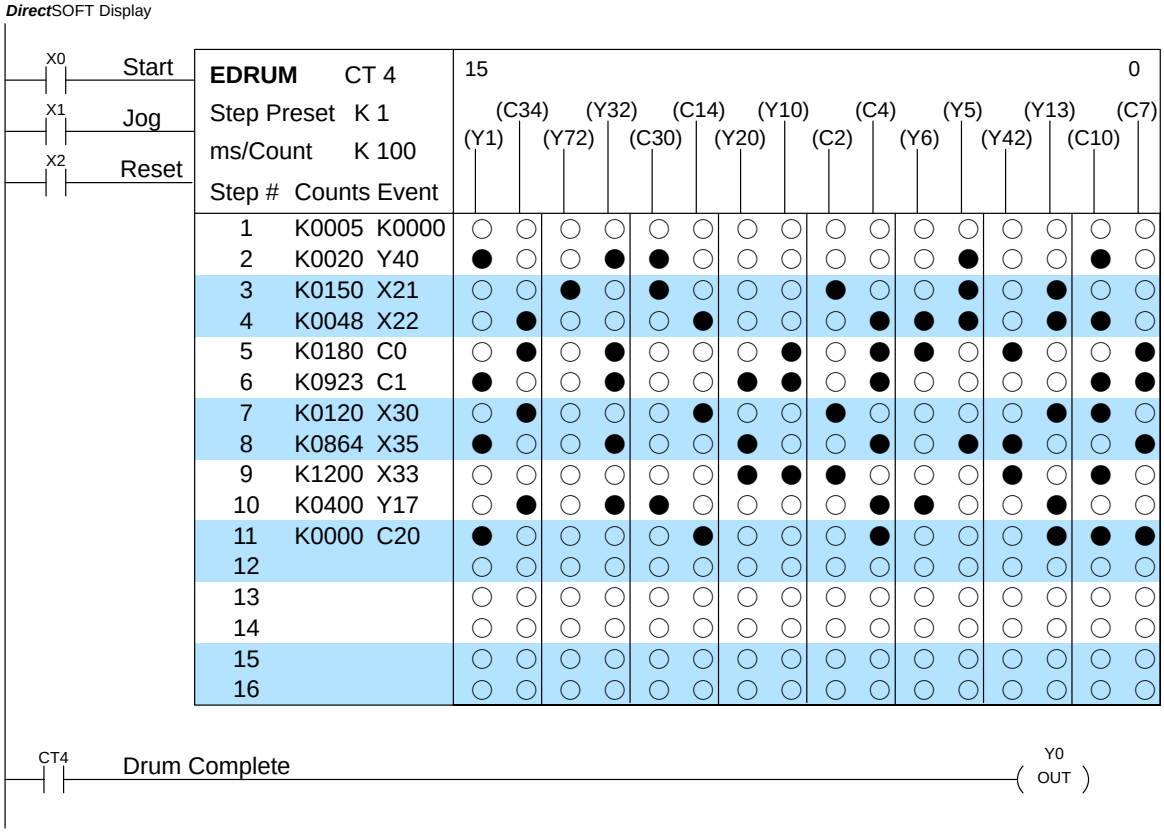
Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa	–	0 – 177
Preset Step	bb	K	1 – 16
Timer base	cccc	K	0 – 99.99 seconds
Counts per step	dddd	K	0 – 9999
Event	eeee	X, Y, C, S, T, ST	
Discrete Outputs	Fffff	X, Y, C ,	

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CT(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CT(n)	0 – 124	Counts in step	CTn = Drum Complete
CT(n+1)	1 – 125	Timer value	CT(n+1) = (not used)
CT(n+2)	2 –126	Preset Step	CT(n+2) = (not used)
CT(n+3)	3 –127	Current Step	CT(n+1) = (not used)

The following ladder program shows the EDRUM instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 11 are used, and all sixteen output points are used. The preset step is step 1. The timebase runs at 100 mS per count. Therefore, the duration of step 1 is (5 x 0.1) = 0.5 seconds. Note that step 1 is time-based only (event = "K0000"). And, the output pattern for step 1 programs all outputs off, which is a typically desirable powerup condition. In the last rung, the Drum Complete bit (CT4) turns on output Y0 upon completion of the last step (step 10). A drum reset also resets CT4.

DirectSOFT Display



The handheld programmer can also enter or edit drum instructions. The diagram below lists the keystrokes for entering the drum example on the previous page. **NOTE**: Drum editing requires Handheld Programmer firmware version 1.8 or later.

Handheld Programmer Keystrokes

Start		$ STR	→	A 0	ENT					
Jog		$ STR	→	B 1	ENT					
Reset		$ STR	→	C 2	ENT					
Drum Inst.		SHFT	E 4	D 3	R ORN	U ISG	M ORST	→	A 0	ENT
Preset Step	(DEF K0001)	NEXT								
Time Base	(DEF K0000)	G 6	E 4	NEXT						
Outputs 1	(DEF 0000)	SHFT	C 2	H 7	NEXT					
	(DEF 0000)	SHFT	C 2	B 1	A 0	NEXT				
	(DEF 0000)	SHFT	Y MLS	B 1	NEXT					
	(DEF 0000)	SHFT	Y MLS	E 4	NEXT					
	(DEF 0000)	SHFT	Y MLS	F 5	NEXT					
	(DEF 0000)	SHFT	Y MLS	G 6	NEXT					
	(DEF 0000)	SHFT	C 2	E 4	NEXT					
	(DEF 0000)	SHFT	C 2	C 2	NEXT					
	(DEF 0000)	SHFT	Y MLS	A 0	NEXT					
	(DEF 0000)	SHFT	Y MLS	C 2	NEXT					
	(DEF 0000)	SHFT	C 2	B 1	E 4	NEXT				
	(DEF 0000)	SHFT	C 2	D 3	A 0	NEXT				
	(DEF 0000)	SHFT	Y MLS	G 6	NEXT					
	(DEF 0000)	SHFT	Y MLS	H 7	NEXT					
	(DEF 0000)	SHFT	C 2	D 3	E 4	NEXT				
16	(DEF 0000)	SHFT	Y MLS	B 1	NEXT					

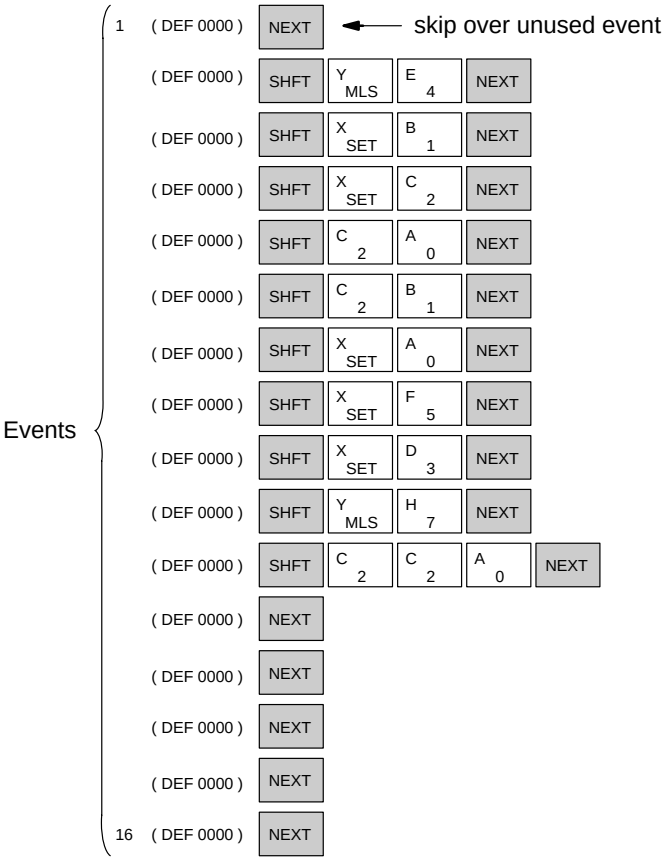
NOTE: You may use the NXT and PREV keys to skip past entries for unused outputs or steps.

Handheld Programmer Keystrokes cont'd

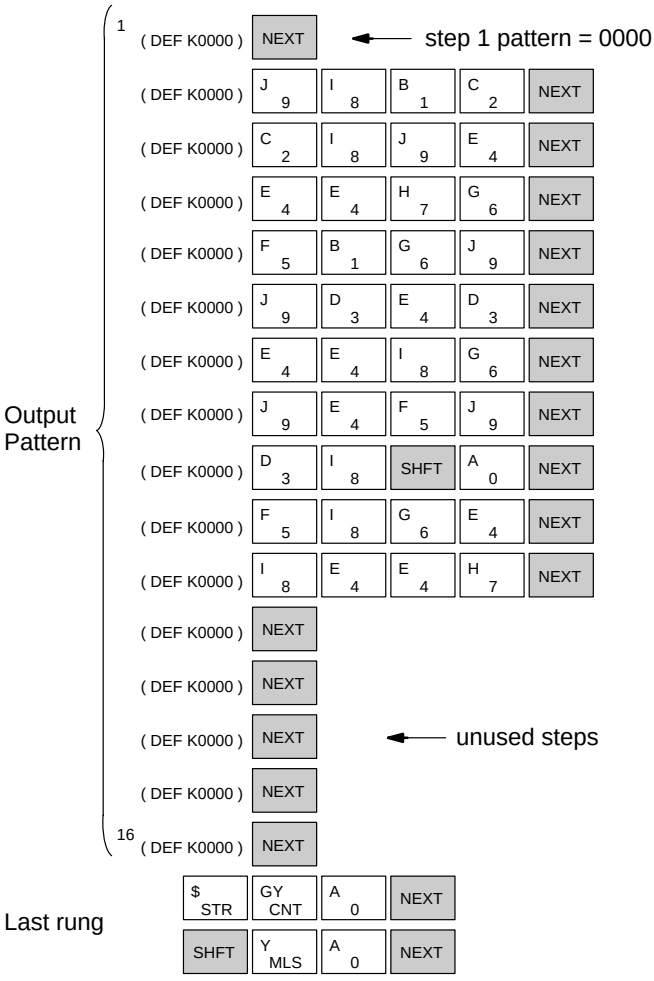
Counts/Step 1	(DEF K0000)	F 5	NEXT			
	(DEF K0000)	C 2	A 0	NEXT		
	(DEF K0000)	B 1	F 5	A 0	NEXT	
	(DEF K0000)	E 4	F 5	NEXT		
	(DEF K0000)	B 1	I 8	A 0	NEXT	
	(DEF K0000)	J 9	C 2	D 3	NEXT	
	(DEF K0000)	B 1	C 2	A 0	NEXT	
	(DEF K0000)	I 8	G 6	E 4	NEXT	
	(DEF K0000)	B 1	C 2	A 0	A 0	NEXT
	(DEF K0000)	E 4	A 0	A 0	NEXT	
	(DEF K0000)	NEXT				
	(DEF K0000)	NEXT				
	(DEF K0000)	NEXT	← skip over unused steps			
	(DEF K0000)	NEXT				
	(DEF K0000)	NEXT				
16	(DEF K0000)	NEXT				

(Continued on next page)

Handheld Programmer Keystrokes cont'd



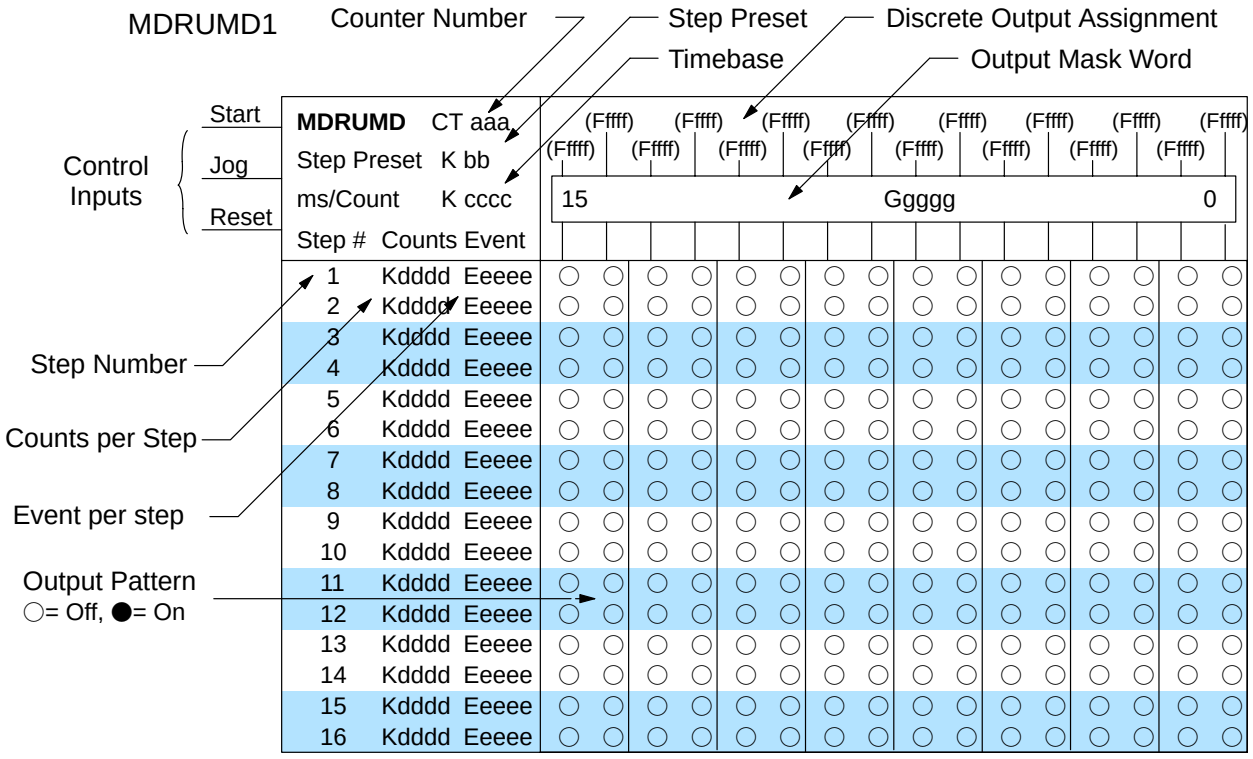
Handheld Programmer Keystrokes cont'd



NOTE: You may use the NXT and PREV keys to skip past entries for unused outputs or steps.

Masked Event Drum with Discrete Outputs (MDRUMD)

The Masked Event Drum with Discrete Outputs has all the features of the basic Event Drum plus final output control for each step. It operates according to the general principles of drum operation covered in the beginning of this section. Below is the instruction in chart form as displayed by ***Direct***SOFT.



The Masked Event Drum with Discrete Outputs features sixteen steps and sixteen outputs. Drum outputs are logically ANDed bit-by-bit with an output mask word for each step. The Ggggg field specifies the beginning location of the 16 mask words. Step transitions occur on timed and/or event basis. The jog input also advances the step on each off-to-on transition. Time is specified in counts per step, and events are specified as discrete contacts. Unused steps must be programmed with “counts per step” = 0, and event = “0000”.

Whenever the Start input is energized, the drum’s timer is enabled. As long as the event is true for the current step, the timer runs during that step. When the step count equals the counts per step, the drum transitions to the next step. This process stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

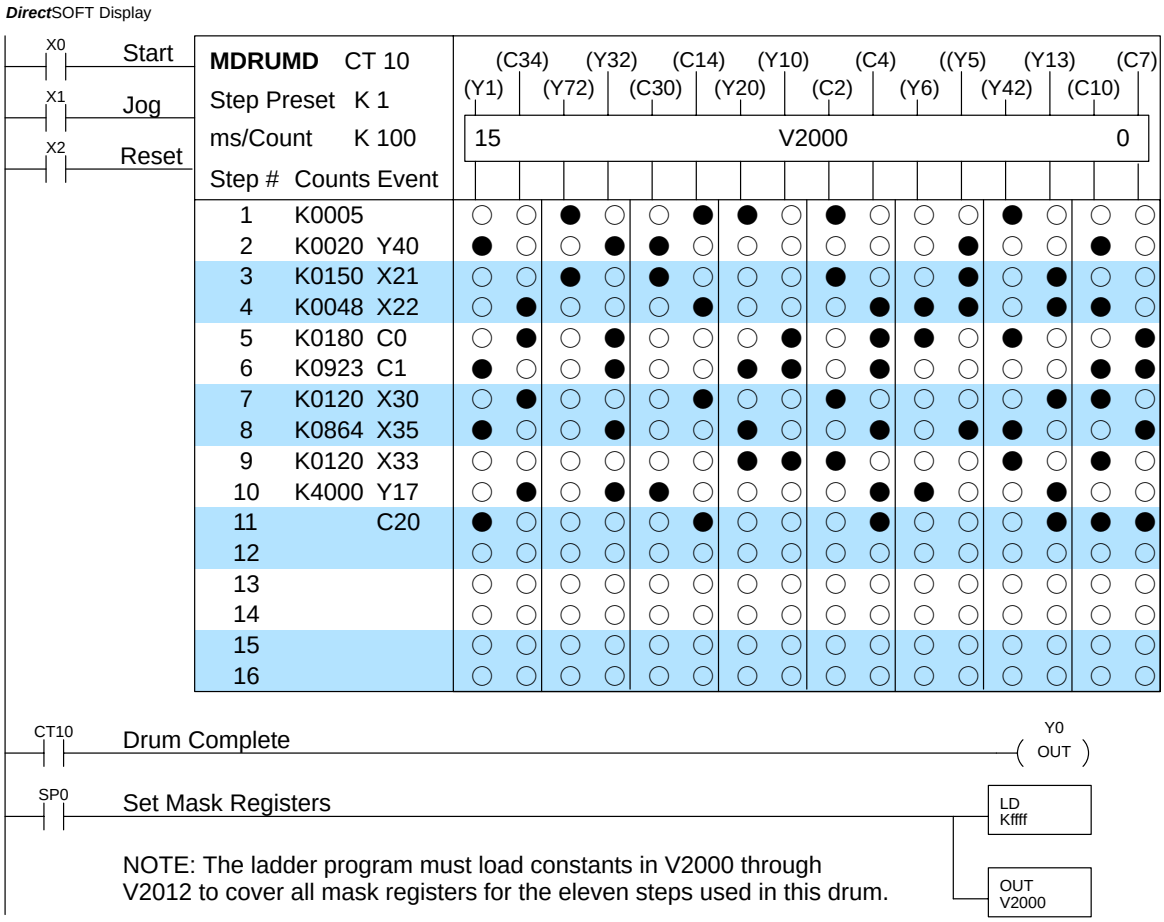
Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa	–	0 – 177
Preset Step	bb	K	1 – 16
Timer base	cccc	K	0 – 99.99 seconds
Counts per step	dddd	K	0 – 9999
Event	eeee	X, Y, C, S, T, ST	
Discrete Outputs	Fffff	X, Y, C	
Output Mask	Ggggg	V	

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CT(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CT(n)	0 – 124	Counts in step	CTn = Drum Complete
CT(n+1)	1 – 125	Timer value	CT(n+1) = (not used)
CT(n+2)	2 –126	Preset Step	CT(n+2) = (not used)
CT(n+3)	3 –127	Current Step	CT(n+1) = (not used)

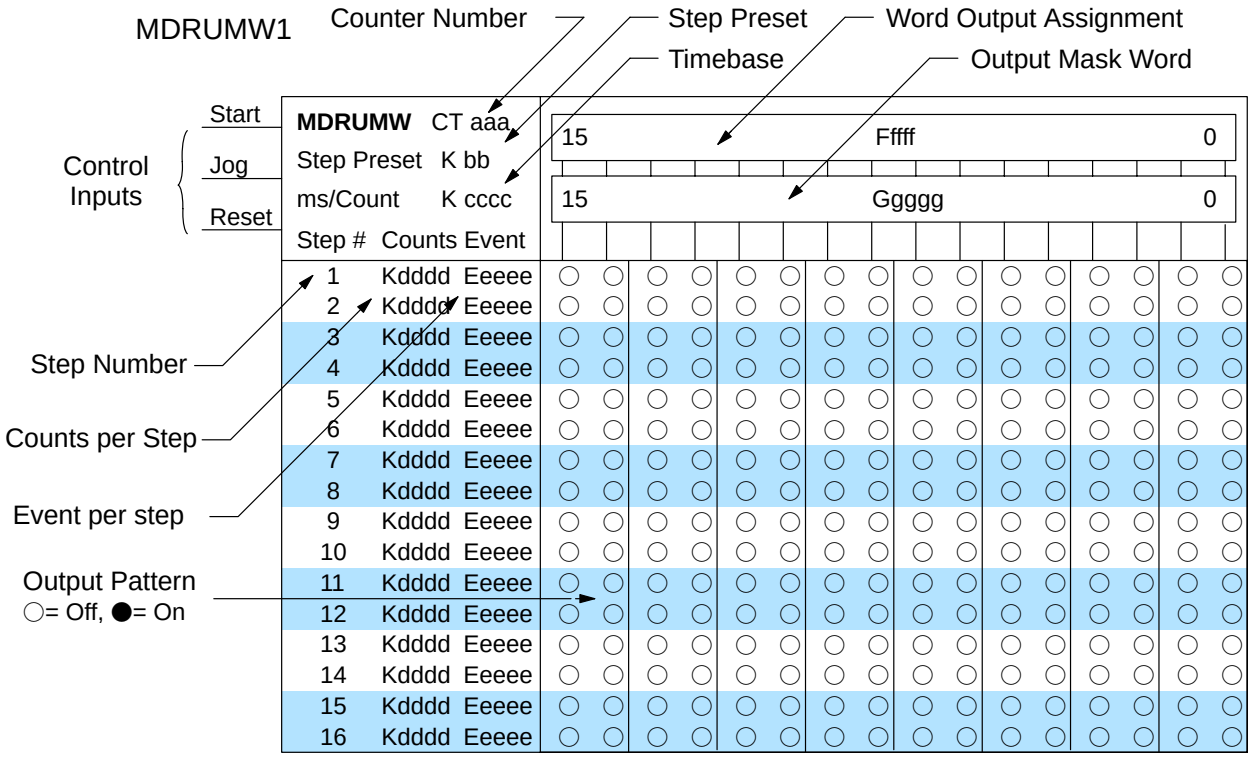
The following ladder program shows the MDRUMD instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 11 are used, and all 16 output points are used. The output mask word is at V2000. The final drum outputs are shown above the mask word as individual bits. The data bits in V2000 are logically ANDed with the output pattern of the current step in the drum. If you want all drum outputs to be off after powerup, write zeros to V2000 on the first scan. Ladder logic may update the output mask at any time to enable or disable the drum outputs. The preset step is step 1. The timebase runs at 100 mS per count. Therefore, the duration of step 1 is (5 x 0.1) = 0.5 seconds. Note that step 1 is time-based only (event – "K0000"). In the last rung, the Drum Complete bit (CT10) turns on output Y0 upon completion of the last step (step 10). A drum reset also resets CT10.

DirectSOFT Display



Masked Event Drum with Word Output (MDRUMW)

The Masked Event Drum with Word Output features outputs organized as bits of a single word, rather than discrete points. It operates according to the general principles of drum operation covered in the beginning of this section. Below is the instruction in chart form as displayed by ***Direct***SOFT.



The Masked Event Drum with Word Output features sixteen steps and sixteen outputs. Drum outputs are logically ANDed bit-by-bit with an output mask word for each step. The Ggggg field specifies the beginning location of the 16 mask words, creating the final output (Fffff field). Step transitions occur on timed and/or event basis. The jog input also advances the step on each off-to-on transition. Time is specified in counts per step, and events are specified as discrete contacts. Unused steps must be programmed with “counts per step” = 0, and event = “0000”.

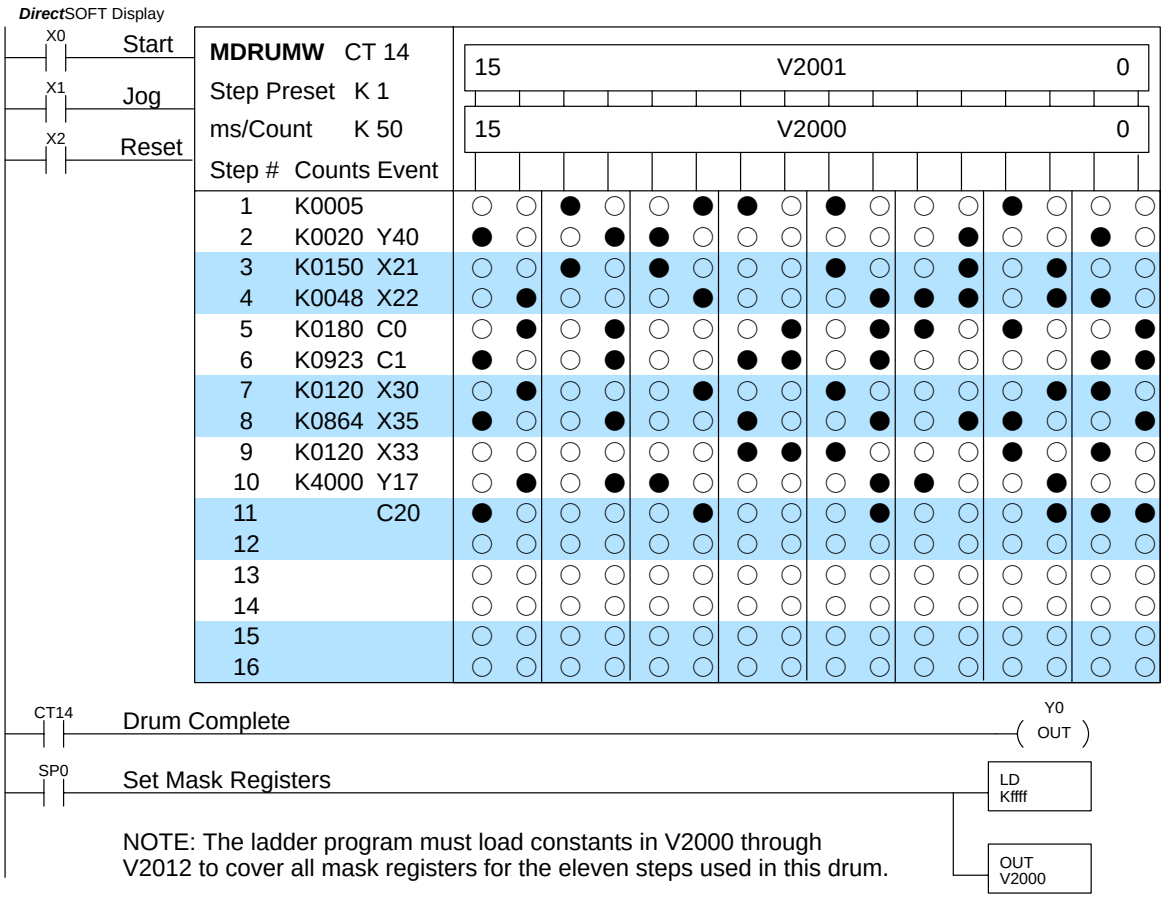
Whenever the Start input is energized, the drum’s timer is enabled. As long as the event is true for the current step, the timer runs during that step. When the step count equals the counts per step, the drum transitions to the next step. This process stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa	–	0 – 177
Preset Step	bb	K	1 – 16
Timer base	cccc	K	0 – 99.99 seconds
Counts per step	dddd	K	0 – 9999
Event	eeee	X, Y, C, S, T, ST	see page 3–29
Word Output	Fffff	V	see page 3–29
Output Mask	Ggggg	V	see page 3–29

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CT(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CT(n)	0 – 124	Counts in step	CTn = Drum Complete
CT(n+1)	1 – 125	Timer value	CT(n+1) = (not used)
CT(n+2)	2 –126	Preset Step	CT(n+2) = (not used)
CT(n+3)	3 –127	Current Step	CT(n+1) = (not used)

The following ladder program shows the MDRUMD instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 11 are used, and all sixteen output points are used. The output mask word is at V2000. The final drum outputs are shown above the mask word as a word at V2001. The data bits in V2000 are logically ANDed with the output pattern of the current step in the drum, generating the contents of V2001. If you want all drum outputs to be off after powerup, write zeros to V2000 on the first scan. Ladder logic may update the output mask at any time to enable or disable the drum outputs. The preset step is step 1. The timebase runs at 50 mS per count. Therefore, the duration of step 1 is (5 x 0.1) = 0.5 seconds. Note that step 1 is time-based only (event – "K0000"). In the last rung, the Drum Complete bit (CT14) turns on output Y0 upon completion of the last step (step 10). A drum reset also resets CT14.



7

RLL*PLUS* Stage Programming

In This Chapter. . . .

- Introduction to Stage Programming
- Learning to Draw State Transition Diagrams
- Using the Stage Jump Instruction for State Transitions
- Stage Program Example: Toggle On/Off Lamp Controller
- Four Steps to Writing a Stage Program
- Stage Program Example: A Garage Door Opener
- Stage Program Design Considerations
- Parallel Processing Concepts
- Managing Large Programs
- RLL*PLUS* Instructions
- Questions and Answers About Stage Programming

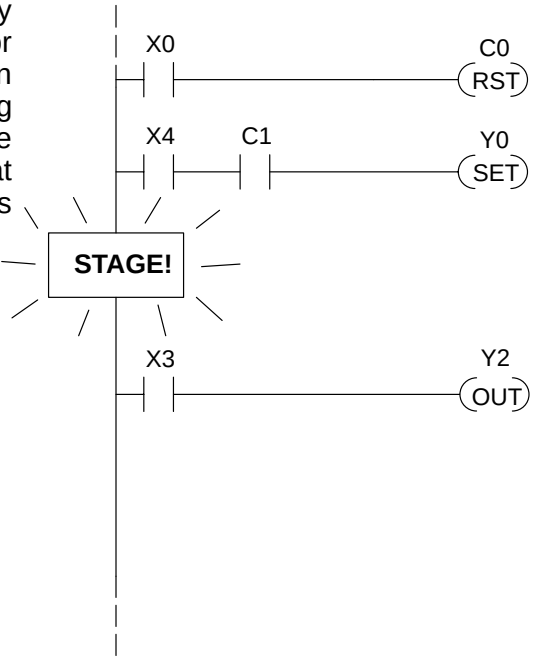
Introduction to Stage Programming

Stage Programming provides a way to organize and program complex applications with relative ease, when compared to purely relay ladder logic (RLL) solutions. Stage programming does not replace or negate the use of traditional boolean ladder programming. This is why Stage Programming is also called RLL*PLUS*. You will not have to discard any training or experience you already have. Stage programming simply allows you to divide and organize a RLL program into groups of ladder instructions called stages. This allows quicker and more intuitive ladder program development than traditional RLL alone provides.

Overcoming "Stage Fright"

Many PLC programmers in the industry have become comfortable using RLL for every PLC program they write... but often remain skeptical or even fearful of learning new techniques such as stage programming. While RLL is great at solving boolean logic relationships, it has disadvantages as well:

- Large programs can become almost unmanageable, because of a lack of structure.
- In RLL, latches must be tediously created from self-latching relays.
- When a process gets stuck, it is difficult to find the rung where the error occurred.
- Programs become difficult to modify later, because they do not intuitively resemble the application problem they are solving.



It's easy to see that these inefficiencies consume a lot of additional time, and time is money. *Stage programming overcomes these obstacles!* We believe a few moments of studying the stage concept is one of the greatest investments in programming speed and efficiency a PLC programmer can make!

So, we encourage you to study stage programming and add it to your "toolbox" of programming techniques. This chapter is designed as a self-paced tutorial on stage programming. For best results:

- Start at the beginning and do not skip over any sections.
- Study each stage programing concept by working through each example. The examples build progressively on each other.
- Read the Stage Questions and Answers at the end of the chapter for a quick review.

Learning to Draw State Transition Diagrams

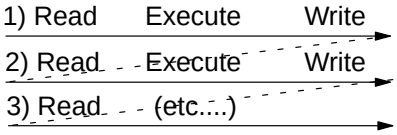
Introduction to Process States

Those familiar with ladder program execution know the CPU must scan the ladder program repeatedly, over and over. Its three basic steps are:

1. Read the inputs
2. Execute the ladder program
3. Write the outputs

The benefit is that a change at the inputs can affect the outputs in a few milliseconds.

PLC Scan



Most manufacturing processes consist of a series of activities or conditions , each lasting for several seconds. minutes, or even hours. We might call these “process states”, which are either active or inactive at any particular time. A challenge for RLL programs is that a particular input event may last for a brief instant. We typically create latching relays in RLL to preserve the input event in order to maintain a process state for the required duration.

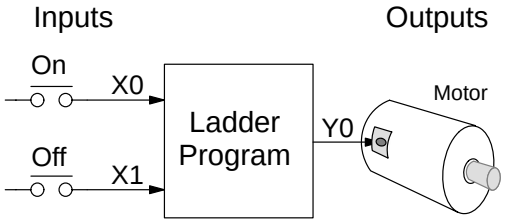
We can organize and divide ladder logic into sections called “stages”, representing process states. But before we describe stages in detail, we will reveal **the secret to understanding stage programming:** state transition diagrams.

The Need for State Diagrams

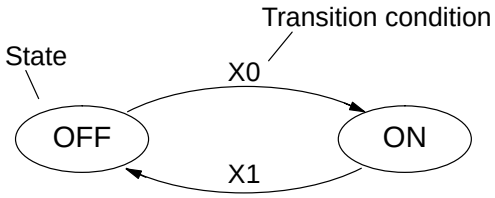
Sometimes we need to forget about the scan nature of PLCs, and focus our thinking toward the states of the process we need to identify. Clear thinking and concise analysis of an application gives us the best chance at writing efficient, bug-free programs. *State diagrams are tools to help us draw a picture of our process!* You will discover that if we can get the picture right, **our program will also be right!**

A 2–State Process

Consider the simple process shown to the right, which controls an industrial motor. We will use a green momentary SPST pushbutton to turn the motor on, and a red one to turn it off. The machine operator will press the appropriate pushbutton for a second or so. The two states of our process are ON and OFF.



The next step is to draw a *state transition diagram*, as shown to the right. It shows the two states OFF and ON, with two transition lines in-between. When the event X0 is true, we transition from OFF to ON. When X1 is true, we transition from ON to OFF.

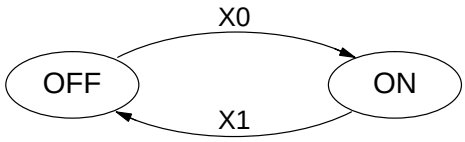


Output equation: Y0 = ON

If you’re following along, you are very close to grasping the concept and the problem-solving power of state transition diagrams. The output of our controller is Y0, which is true any time we are in the ON state. In a boolean sense, Y0=ON state.

Next, we will implement the state diagram first as RLL, then as a stage program. This will help you see the relationship between the two methods in problem solving.

The state transition diagram to the right is a picture of the solution we need to create. It expresses the problem independently of the programming language we may use to realize it. In other words, *by drawing the diagram we have already solved the control problem!*



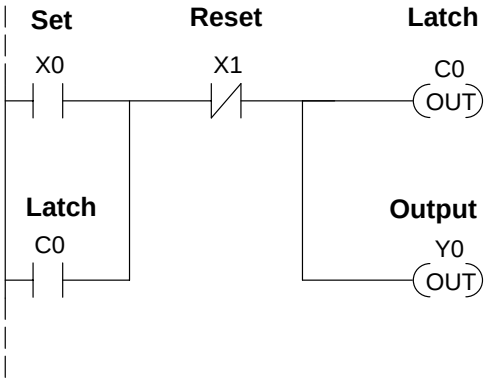
Output equation: Y0 = ON

First, we will translate the state diagram to traditional RLL. Then we will show how easy it is to translate the diagram into a stage programming solution.

RLL Equivalent

The RLL solution is shown to the right. It consists of a self-latching control relay, C0. When the On momentary pushbutton (X0) is pressed, output coil C0 turns on and the C0 contact on the second row latches itself on. So, X0 **sets the latch** C0 on, and it remains on after the X0 contact opens. The motor output Y0 also has power flow, so the motor is now on.

When the Off pushbutton (X1) is pressed, it opens the normally-closed X1 contact, which **resets the latch**. Motor output Y0 turns off when the latch coil C0 goes off.

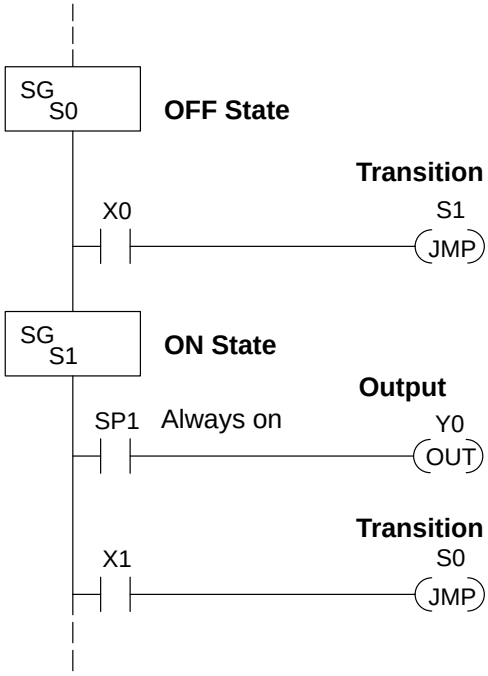


Stage Equivalent

The stage program solution is shown to the right. The two inline stage boxes S0 and S1 correspond to the two states OFF and ON. The ladder rung(s) below each stage box belong to each respective stage. This means *the PLC only has to scan those rungs when the corresponding stage is active!*

For now, let's assume we begin in the OFF State, so stage S0 is active. When the On pushbutton (X0) is pressed, a stage transition occurs. The JMP S1 instruction executes, which simply turns off the Stage bit S0 and turns on Stage bit S1. So on the next PLC scan, the CPU will not execute Stage S0, but will execute stage S1!

In the On State (Stage S1), we want the motor to always be on. The special relay contact SP1 is defined as always on, so Y0 turns the motor on.

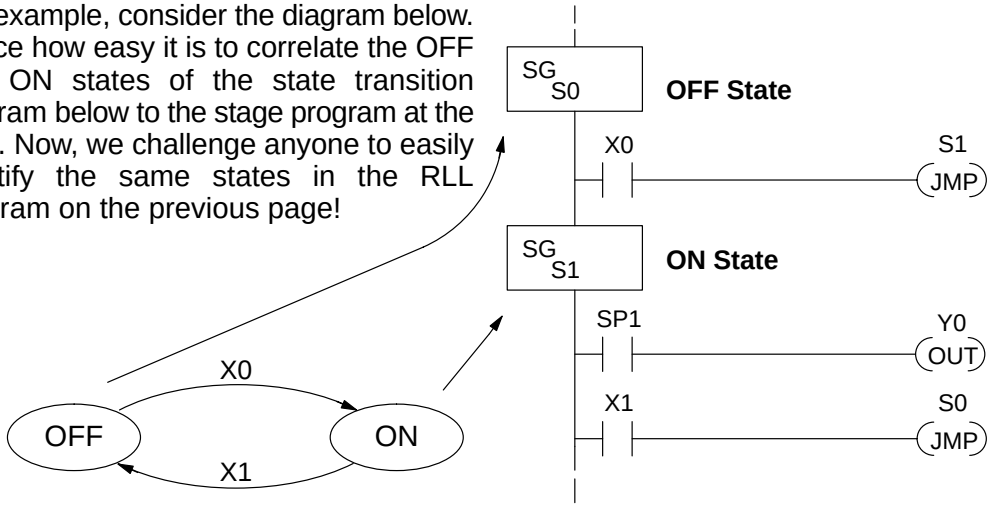


When the Off pushbutton (X1) is pressed, a transition back to the Off State occurs. The JMP S0 instruction executes, which simply turns off the Stage bit S1 and turns on Stage bit S0. On the next PLC scan, the CPU will not execute Stage S1, so the motor output Y0 will turn off. The Off state (Stage 0) will be ready for the next cycle.

Let's Compare

You may be thinking "I don't see the big advantage to Stage Programming... in fact, the stage program is longer than the standard RLL program". As control problems grow in complexity, stage programming quickly out-performs RLL in simplicity, program size, etc.

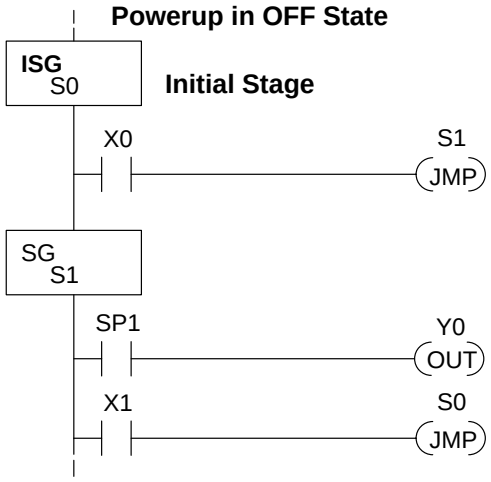
For example, consider the diagram below. Notice how easy it is to correlate the OFF and ON states of the state transition diagram below to the stage program at the right. Now, we challenge anyone to easily identify the same states in the RLL program on the previous page!



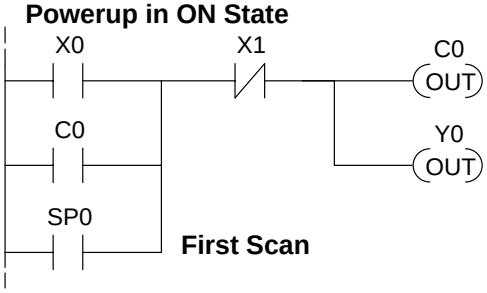
Initial Stages

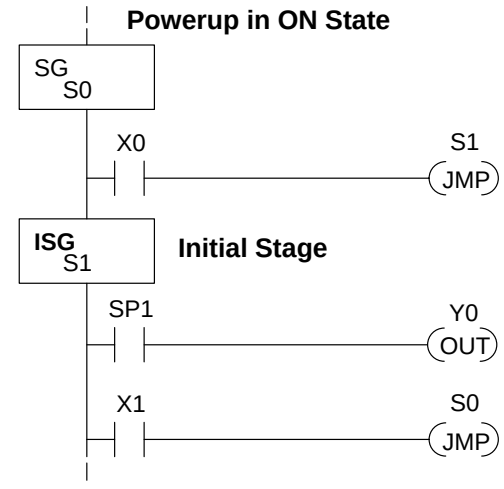
At powerup and Program-to-Run Mode transitions, the PLC always begins with all normal stages (SG) off. So, the stage programs shown so far have actually had no way to get started (because rungs are not scanned unless their stage is active).

Assume that we want to always begin in the Off state (motor off), which is how the RLL program works. The Initial Stage (ISG) is defined to be active at powerup. In the modified program to the right, we have changed stage S0 to the ISG type. This ensures the PLC will scan contact X0 after powerup, because Stage S0 is active. **After powerup, an Initial Stage (ISG) works like any other stage!**

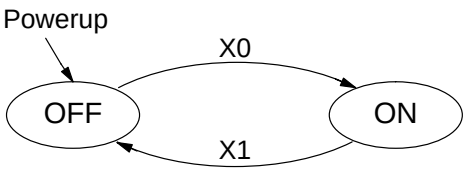


We can change both programs so the motor is ON at powerup. In the RLL below, we must add a first scan relay SP0, latching C0 on. In the stage example to the right, we simply make Stage S1 an initial stage (ISG) instead of S0.





Mark the desired powerup state as shown to the right, which helps us remember to use the appropriate Initial Stages when creating a stage program. It is permissible to have as many initial stages as the process requires.



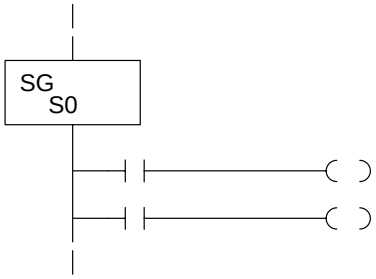
What Stage Bits Do

You may recall that a stage is a section of ladder program which is either active or inactive at a given moment. All stage bits (S0 – Sxxx) reside in the PLCs image register as individual status bits. Each stage bit is either a boolean 0 or 1 at any time.

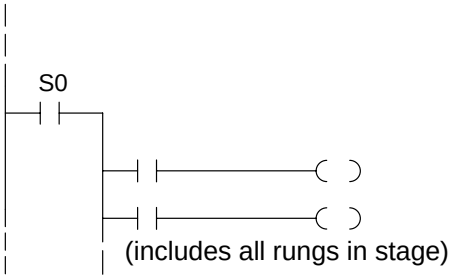
Program execution always reads ladder rungs from top to bottom, and from left to right. The drawing below shows the effect of stage bit status. The ladder rungs below the stage instruction continuing until the next stage instruction or the end of program belong to stage 0. Its equivalent operation is shown on the right. When S0 is true, the two rungs have power flow.

- If Stage bit S0 = 0, its ladder rungs *are not* scanned (executed).
- If Stage bit S0 = 1, its ladder rungs *are* scanned (executed).

Actual Program Appearance



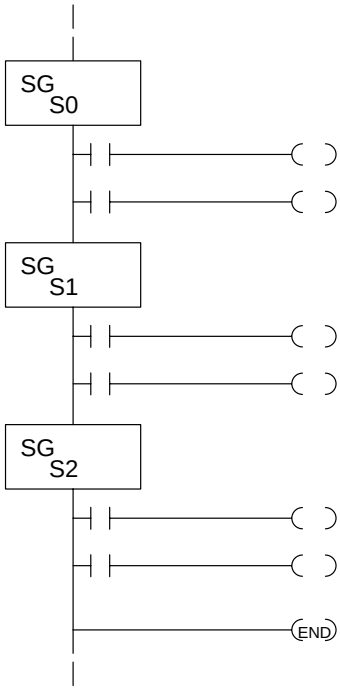
Functionally Equivalent Ladder



Stage Instruction Characteristics

The inline stage boxes on the left power rail divide the ladder program rungs into stages. Some stage rules are:

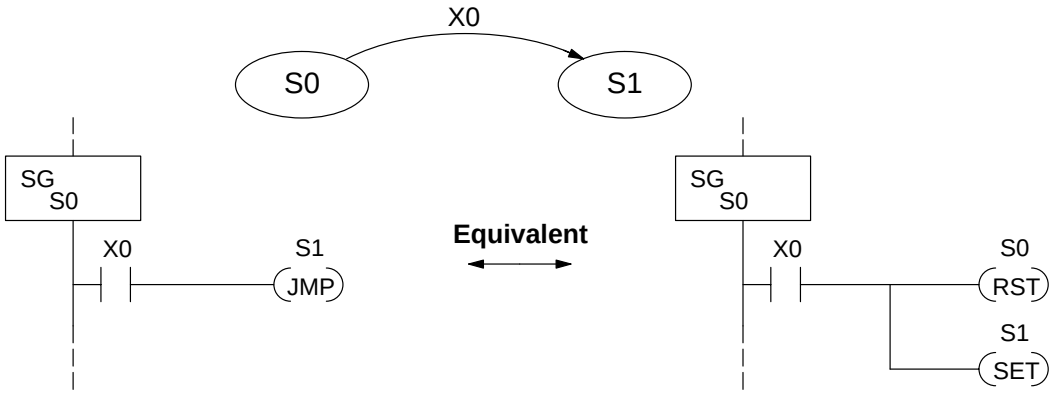
- **Execution** – Only logic in active stages are executed on any scan.
- **Transitions** – Stage transition instructions take effect on the next occurrence of the stages involved.
- **Octal numbering** – Stages are numbered in octal, like I/O points, etc. So “S8” is not valid.
- **Total Stages** – The maximum number of stages is CPU-dependent.
- **No duplicates** – Each stage number is unique and can be used once.
- **Any order** – You can skip numbers and sequence the stage numbers in any order.
- **Last Stage** – the last stage in the ladder program includes all rungs from its stage box until the end coil.



Using the Stage Jump Instruction for State Transitions

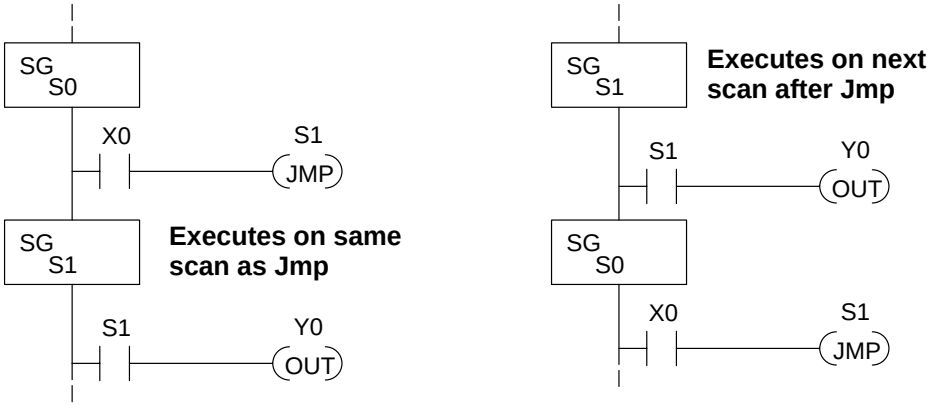
Stage Jump, Set, and Reset Instructions

The Stage JMP instruction deactivates the stage in which the instruction occurs, while activating the stage in the JMP instruction. Refer to the state transition shown below. When contact X0 energizes, the state transition from S0 to S1 occurs. The two stage examples shown below are equivalent. The Stage Jump instruction is equal to a Stage Reset of the current stage, plus a Stage Set instruction for the stage you want to transition.



Please Read Carefully – The jump instruction is easily misunderstood. The "jump" does not occur immediately like a GOTO or GOSUB program control instruction when executed. Here's how it works:

- The jump instruction resets the stage bit of the stage in which it occurs. All rungs in the stage still finish executing during the current scan, *even if there are other rungs in the stage below the jump instruction!*
- The reset will be in effect on the following scan, so the stage that executed the jump instruction previously will be inactive and bypassed.
- The stage bit of the stage named in the Jump instruction will be set immediately, so the stage will be executed on its next occurrence. In the left program shown below, stage S1 executes during the *same scan* as the JMP S1 occurs in S0. In the example on the right, Stage S1 executes on the *next scan* after the JMP S1 executes, because stage S1 is located *above* stage S0.

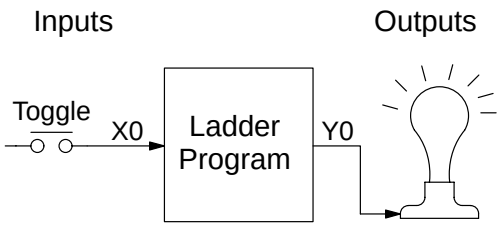


Note: Assume we start with Stage 0 active and stage 1 inactive for both examples.

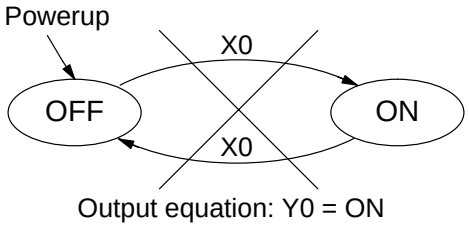
Stage Program Example: Toggle On/Off Lamp Controller

A 4–State Process

In the process shown to the right, an ordinary momentary pushbutton is used to control a light bulb. The ladder program will latch the switch input. Push and release to turn on the light, push and release again to turn it off (sometimes called toggle function). You could buy a mechanical switch with the alternate on/off action built in... However, this example is educational and also fun!

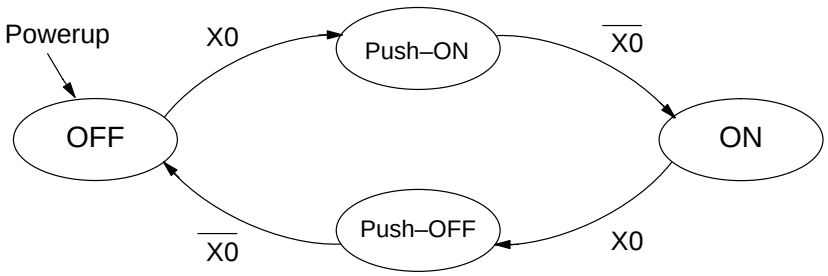


Next draw the state transition diagram. A typical first approach is to use X0 for both transitions (like the example shown to the right). However, *this is incorrect* (please keep reading).



This example differs from the motor example, because there is only one pushbutton. When the pushbutton is pressed, both transition conditions are met. If implemented in Stage, this solution would flash the light on or off each scan (obviously undesirable)!

The solution is to make the the push and the release of the pushbutton separate events. Refer to the new state transition diagram below. At powerup enter the OFF state. When switch X0 is pressed, enter the Press-ON state. When it is released, enter the ON state. Note that X0 with the bar above it denotes X0 NOT.

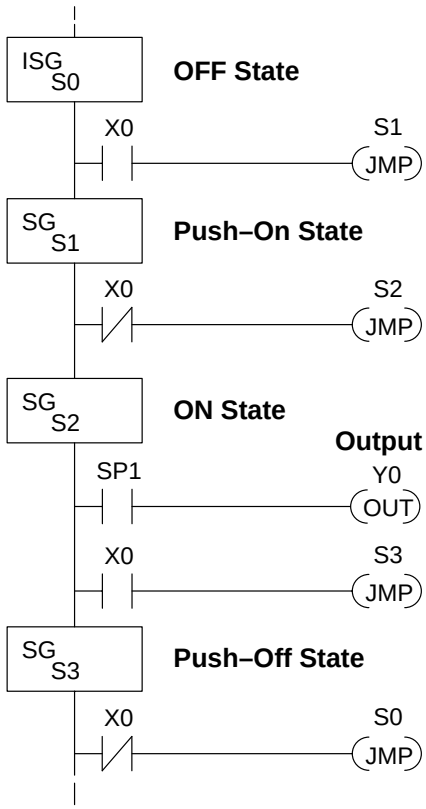


Output equation: Y0 = ON

When in the ON state, another push and release cycle similarly takes us back to the OFF state. Now there are two unique states (OFF and ON) used when the pushbutton is released, which is what was required to solve the control problem.

The equivalent stage program is shown to the right. The desired powerup state is OFF, therefore, make S0 an initial stage (ISG). In the ON state, add special relay contact SP1, which is always on.

Note that even as the programs grow more complex, it is still easy to correlate the state transition diagram with the stage program!



Four Steps to Writing a Stage Program

By now, you've probably noticed that the same steps are followed to solve each example problem. The steps will probably come to you automatically if you work through all the examples in this chapter. It's helpful to have a checklist to guide through the problem solving. The following steps summarize the stage program design procedure:

1. Write a Word Description of the application.

Describe all functions of the process in your own words. Start by listing what happens first, then next, etc. If you find there are too many things happening at once, try dividing the problem into more than one process. Remember, you can still have the processes communicate with each other to coordinate their overall activity.

2. Draw the Block Diagram.

Inputs represent all the information the process needs for decisions, and outputs connect to all devices controlled by the process.

- Make lists of inputs and outputs for the process.
- Assign I/O point numbers (X and Y) to physical inputs and outputs.

3. Draw the State Transition Diagram.

The state transition diagram describes the central function of the block diagram, reading inputs and generating outputs.

- Identify and name the states of the process.
- Identify the event(s) required for each transition between states.
- Ensure the process has a way to re-start itself, or is cyclical.
- Choose the powerup state for your process.
- Write the output equations.

4. Write the Stage Program.

Translate the state transition diagram into a stage program.

- Make each state a stage. Remember to number stages in octal. Up to 1024 total stages are available in the DL350 CPUs.
- Put transition logic inside the stage which originates each transition (the stage each arrow points *away* from).
- Use an initial stage (ISG) for any states that must be active at powerup.
- Place the outputs or actions in the appropriate stages.

You will notice that Steps 1 through 3 *prepare* us to write the stage program in Step 4. However, the program virtually writes itself because of the preparation beforehand. Soon you will be able to start with a word description of an application and create a stage program in one easy session!

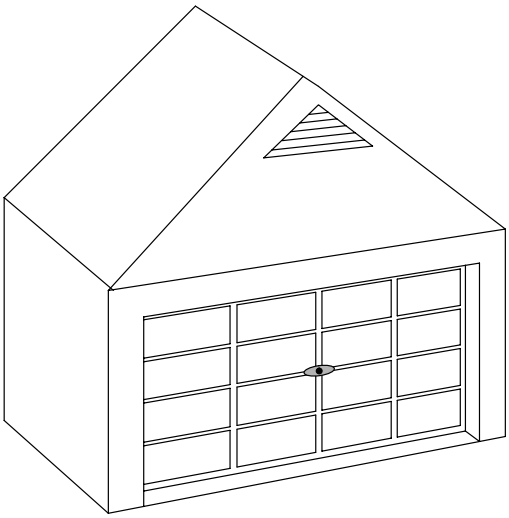
Stage Program Example: A Garage Door Opener

Garage Door Opener Example

In this next stage programming example we will create a garage door opener controller. Hopefully most readers are familiar with this application, and we can have fun besides!

The first step we must take is to describe how the door opener works. We will start by achieving the basic operation, waiting to add extra features later (stage programs are very easy to modify).

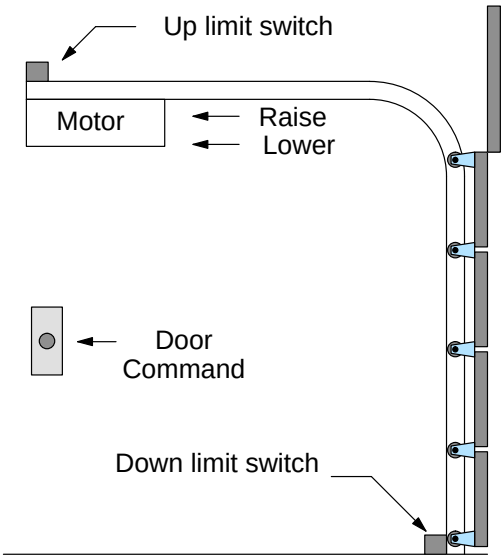
Our garage door controller has a motor which raises or lowers the door on command. The garage owner pushes and releases a momentary pushbutton once to raise the door. After the door is up, another push-release cycle will lower the door.



In order to identify the inputs and outputs of the system, it's sometimes helpful to sketch its main components, as shown in the door side view to the right. The door has an up limit and a down limit switch. Each limit switch closes only when the door has reached the end of travel in the corresponding direction. In the middle of travel, neither limit switch is closed.

The motor has two command inputs: raise and lower. When neither input is active, the motor is stopped.

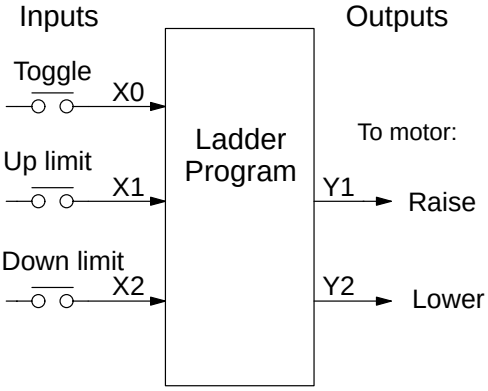
The door command is a simple pushbutton. Whether wall-mounted as shown, or a radio-remote control, all door control commands logically OR together as one pair of switch contacts.



Draw the Block Diagram

The block diagram of the controller is shown to the right. Input X0 is from the pushbutton door control. Input X1 energizes when the door reaches the full up position. Input X2 energizes when the door reaches the full down position. When the door is positioned between fully up or down, both limit switches are open.

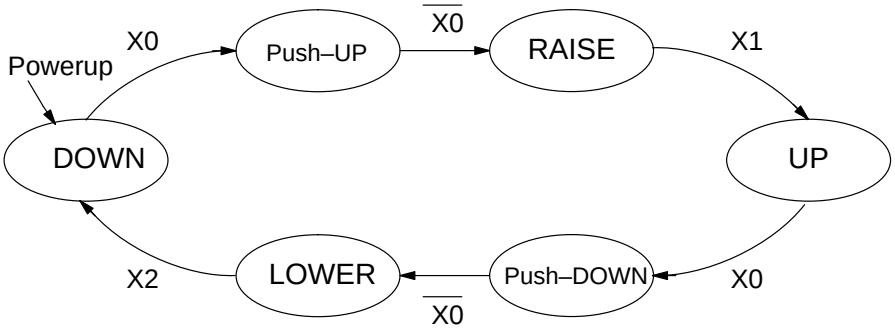
The controller has two outputs to drive the motor. Y1 is the up (raise the door) command, and Y2 is the down (lower the door) command.



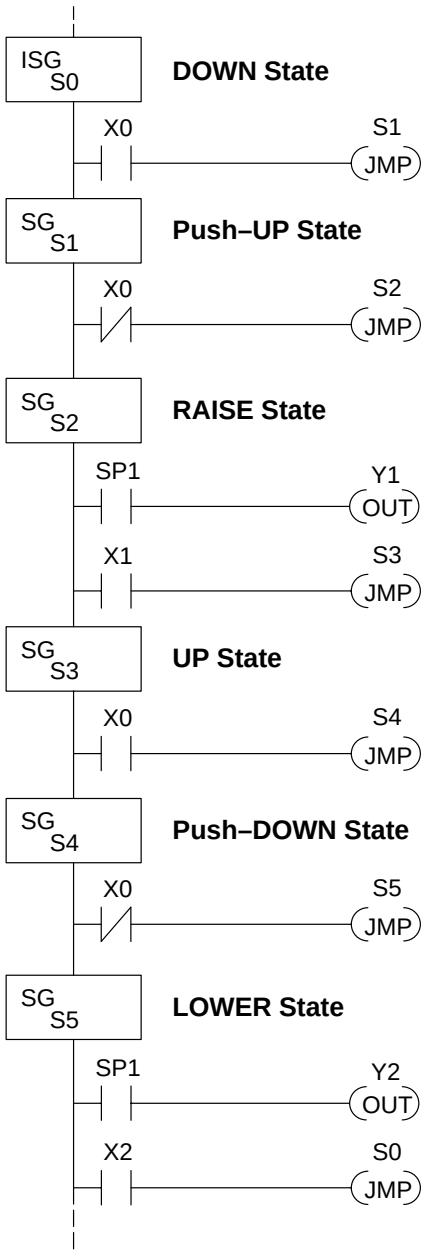
Draw the State Diagram

Now we are ready to draw the state transition diagram. Like the previous light bulb controller example, this application also has only one switch for the command input. Refer to the figure below.

- When the door is down (DOWN state), nothing happens until X0 energizes. Its push and release brings us to the RAISE state, where output Y1 turns on and causes the motor to raise the door.
- We transition to the UP state when the up limit switch (X1) energizes, and turns off the motor.
- Then nothing happens until another X0 press-release cycle occurs. That takes us to the LOWER state, turning on output Y2 to command the motor to lower the door. We transition back to the DOWN state when the down limit switch (X2) energizes.



Output equations: Y1 = RAISE Y2 = LOWER



The equivalent stage program is shown to the right. For now, we will assume the door is down at powerup, so the desired powerup state is DOWN. We make S0 an initial stage (ISG). Stage S0 remains active until the door control pushbutton activates. Then we transition (JMP) to Push-UP stage, S1.

A push-release cycle of the pushbutton takes us through stage S1 to the RAISE stage, S2. We use the always-on contact SP1 to energize the motor's raise command, Y1. When the door reaches the fully-raised position, the up limit switch X1 activates. This takes us to the UP Stage S3, where we wait until another door control command occurs.

In the UP Stage S3, a push-release cycle of the pushbutton will take us to the LOWER Stage S5, where we activate Y2 to command the motor to lower the door. This continues until the door reaches the down limit switch, X2. When X2 closes, we transition from Stage S5 to the DOWN stage S0, where we began.

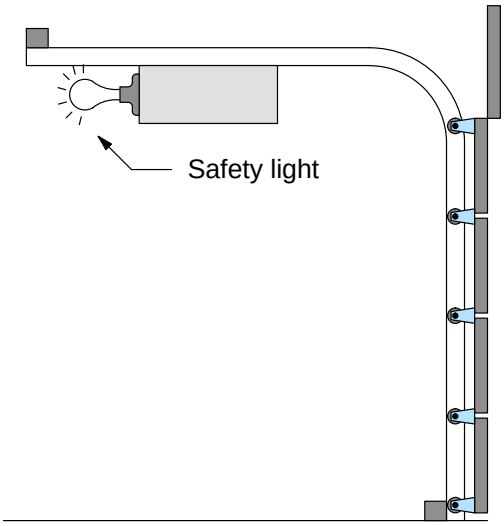
NOTE: The only special thing about an initial stage (ISG) is that it is automatically active at powerup. Afterwards, it is like any other.

Add Safety Light Feature

Next we will add a safety light feature to the door opener system. It's best to get the main function working first as we have done, then adding the secondary features.

The safety light is standard on many commercially-available garage door openers. It is shown to the right, mounted on the motor housing. The light turns on upon any door activity, remaining on for approximately 3 minutes afterwards.

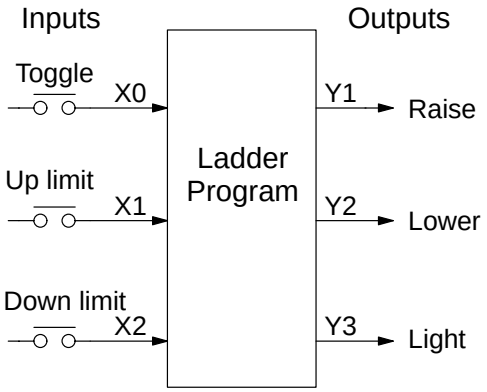
This part of the exercise will demonstrate the use of parallel states in our state diagram. Instead of using the JMP instruction, we will use the set and reset commands.



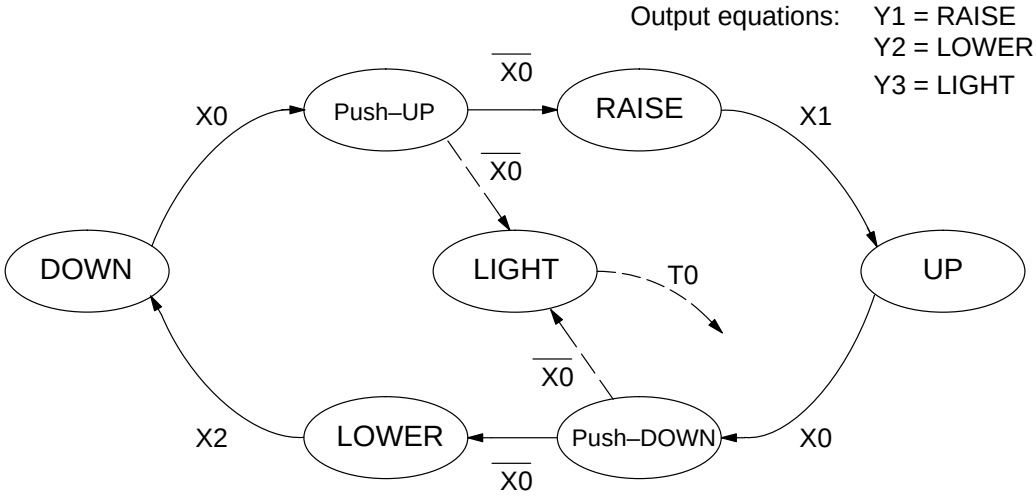
Modify the Block Diagram and State Diagram

To control the light bulb, we add an output to our controller block diagram, shown to the right, Y3 is the light control output.

In the diagram below, we add an additional state called "LIGHT". Whenever the garage owner presses the door control switch and releases, the RAISE or LOWER state is active *and the LIGHT state is simultaneously active.* The line to the Light state is dashed, because it is not the primary path.



We can think of the Light state as a parallel process to the raise and lower state. The paths to the Light state are not a transition (Stage JMP), but a State Set command. In the logic of the Light stage, we will place a three-minute timer. When it expires, timer bit T0 turns on and resets the Light stage. The path out of the Light stage goes nowhere, indicating the Light stage becomes inactive, and the light goes out!



Using a Timer Inside a Stage

The finished modified program is shown to the right. The shaded areas indicate the program additions.

In the Push-UP stage S1, we add the Set Stage Bit S6 instruction. When contact X0 opens, we transition from S1 and go to two new active states: S2 and S6. In the Push-DOWN state S4, we make the same additions. So, any time someone presses the door control pushbutton, the light turns on.

Most new stage programmers would be concerned about where to place the Light Stage in the ladder, and how to number it. The good news is that it doesn't matter!

- Choose an unused Stage number, and use it for the new stage and as the reference from other stages.
- Placement in the program is not critical, so we place it at the end.

You might think that each stage has to be directly under the stage that transitions to it. While it is good practice, it is not required (that's good, because our two locations for the Set S6 instruction make that impossible). Stage numbers and how they are used determines the transition paths.

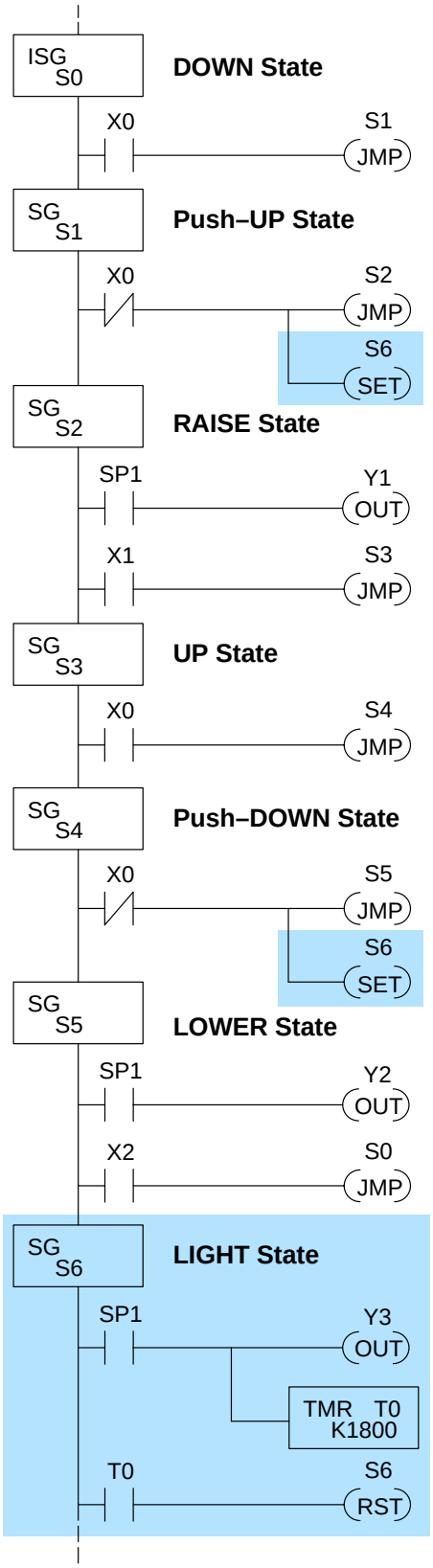
In stage S6, we turn on the safety light by energizing Y3. Special relay contact SP1 is always on. Timer T0 times at 0.1 second per count. To achieve 3 minutes time period, we calculate:

$$K = \frac{3 \text{ min.} \times 60 \text{ sec/min}}{0.1 \text{ sec/count}}$$

$$K = 1800 \text{ counts}$$

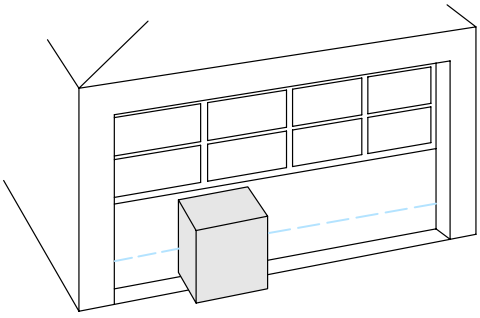
The timer has power flow whenever stage S6 is active. The corresponding timer bit T0 is set when the timer expires. So three minutes later, T0=1 and the instruction Reset S6 causes the stage to be inactive.

While Stage S6 is active and the light is on, stage transitions in the primary path continue normally and independently of Stage 6. That is, the door can go up, down, or whatever, but the light will be on for precisely 3 minutes.

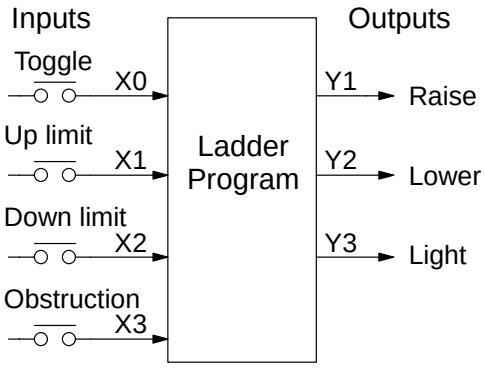


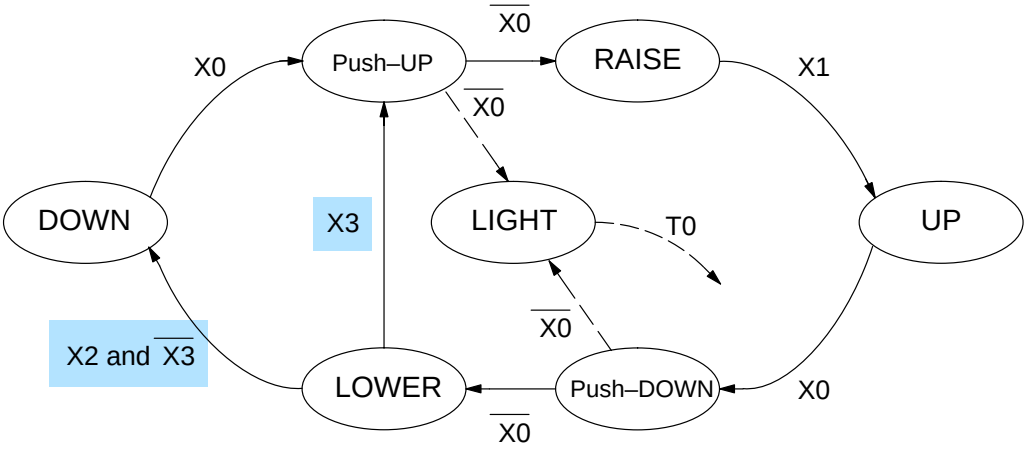
Add Emergency Stop Feature

Some garage door openers today will detect an object under the door. This halts further lowering of the door. Usually implemented with a photocell ("electric-eye"), a door in the process of being lowered will halt and begin raising. We will define our safety feature to work in this way, adding the input from the photocell to the block diagram as shown to the right. X3 will be on if an object is in the path of the door.



Next, we make a simple addition to the state transition diagram, shown in shaded areas in the figure below. Note the new transition path at the top of the LOWER state. If we are lowering the door and detect an obstruction (X3), we then jump to the Push-UP State. We do this instead of jumping directly to the RAISE state, to give the Lower output Y2 one scan to turn off, before the Raise output Y1 energizes.

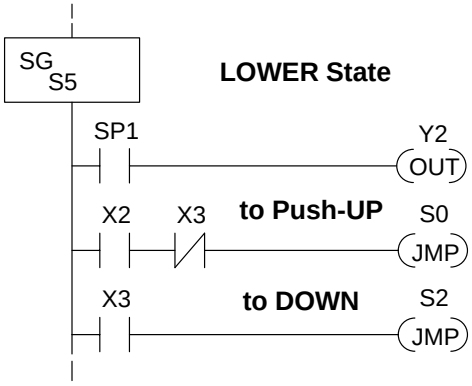




Exclusive Transitions

It is theoretically possible the down limit (X2) and the obstruction input (X3) could energize at the same moment. In that case, we would "jump" to the Push-UP and DOWN states simultaneously, which does not make sense.

Instead, we give priority to the obstruction by changing the transition condition to the DOWN state to [X2 AND NOT X3]. This ensures the obstruction event has the priority. The modifications we must make to the LOWER Stage (S5) logic are shown to the right. The first rung remains unchanged. The second and third rungs implement the transitions we need. Note the opposite relay contact usage for X3, which ensures the stage will execute only one of the JMP instructions.

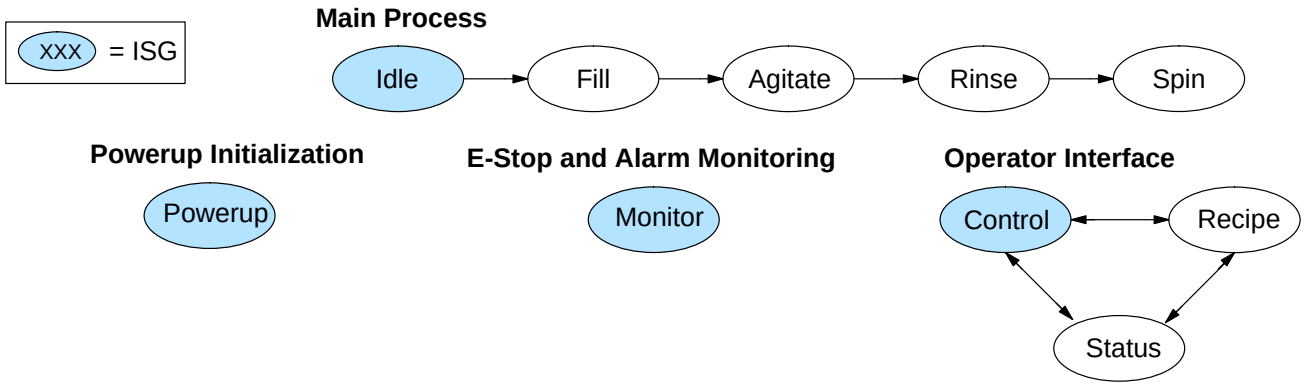


Stage Program Design Considerations

Stage Program Organization

The examples so far in this chapter used one self-contained state diagram to represent the main process. However, we can have multiple processes implemented in stages, all in the same ladder program. New stage programmers sometimes try to turn a stage on and off each scan, based on the false assumption that only one stage can be on at a time. For ladder rungs that you want to execute each scan, put them in a stage that is always on.

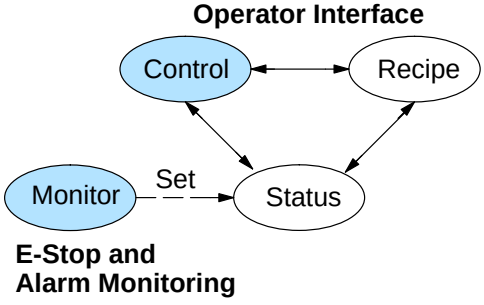
The following figure shows a typical application. During operation, the primary manufacturing activity Main Process, Powerup Initialization, E-Stop and Alarm Monitoring, and Operator Interface are all running. At powerup, four initial stages shown begin operation.



In a typical application, the separate stage sequences above operate as follows:

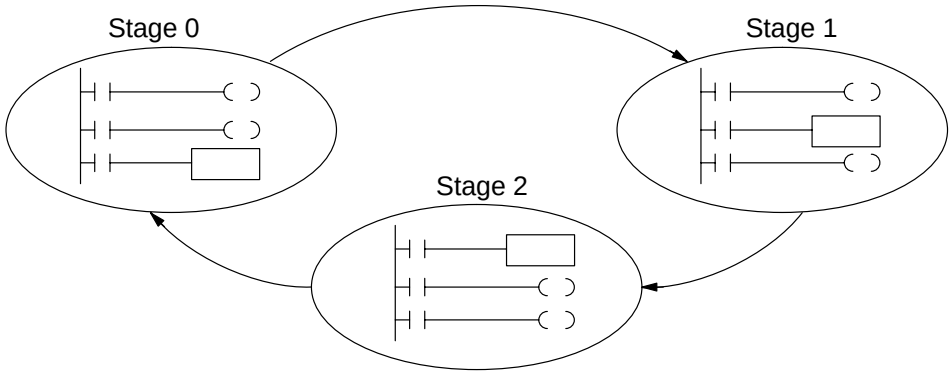
- **Powerup Initialization** – This stage contains ladder rung tasks performed once at powerup. Its last rung resets the stage, so this stage is only active for one scan (or only as many scans that are required).
- **Main Process** – This stage sequence controls the heart of the process or machine. One pass through the sequence represents one part cycle of the machine, or one batch in the process.
- **E-Stop and Alarm Monitoring** – This stage is always active because it is watching for errors that could indicate an alarm condition or require an emergency stop. It is common for this stage to reset stages in the main process or elsewhere, in order to initialize them after an error condition.
- **Operator Interface** – This is another task that must always be active and ready to respond to an operator. It allows an operator interface to change modes, etc. independently of the current main process step.

Although we have separate processes, there can be coordination among them. For example, in an error condition, the Status Stage may want to automatically switch the operator interface to the status mode to show error information as shown to the right. The monitor stage could set the stage bit for Status and Reset the stages Control and Recipe.



How Instructions Work Inside Stages

We can think of states or stages as simply dividing up our ladder program as depicted in the figure below. Each stage contains only the ladder rungs which are needed for the corresponding state of the process. The logic for transitioning out of a stage is contained within that stage. It's easy to choose which ladder rungs are active at powerup by using an "initial" stage type (ISG).



Most instructions work like they do in standard RLL. You can think of a stage like a miniature RLL program which is either active or inactive.

Output Coils – As expected, output coils in active stages will turn on or off outputs according to power flow into the coil. However, note the following:

- Outputs work as usual, provided each output reference (such as "Y3") is used in only one stage.
- Output coils automatically turn off when leaving a stage. However, Set and Reset instructions are not "undone" when leaving a stage.
- An output can be referenced from more than one stage, as long as only one of the stages is active at a time.
- If an output coil is controlled by more than one stage simultaneously, the active stage nearest the bottom of the program determines the final output status during each scan. So, use the OROUT instruction instead when you want multiple stages to have a logical OR control of an output.

One-Shot or PD coils – Use care if you must use a Positive Differential coil in a stage. Remember the input to the coil must make a 0–1 transition. If the coil is already energized on the first scan when the stage becomes active, the PD coil will not work. This is because the 0–1 transition did not occur.

PD coil alternative: If there is a task which you want to do only once (on 1 scan), it can be placed in a stage which transitions to the next stage on the same scan.

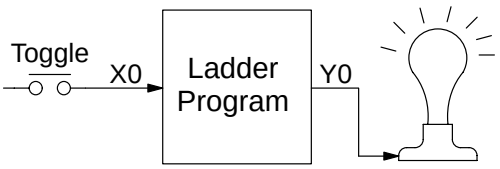
Counter – When using a counter inside a stage, the stage must be active for one scan before the input to the counter makes a 0–1 transition. Otherwise, there is no real transition and the counter will not count. The ordinary Counter instruction does have a restriction inside stages: it may not be reset from other stages using the RST instruction for the counter bit. However, the special Stage Counter provides a solution (see next paragraph).

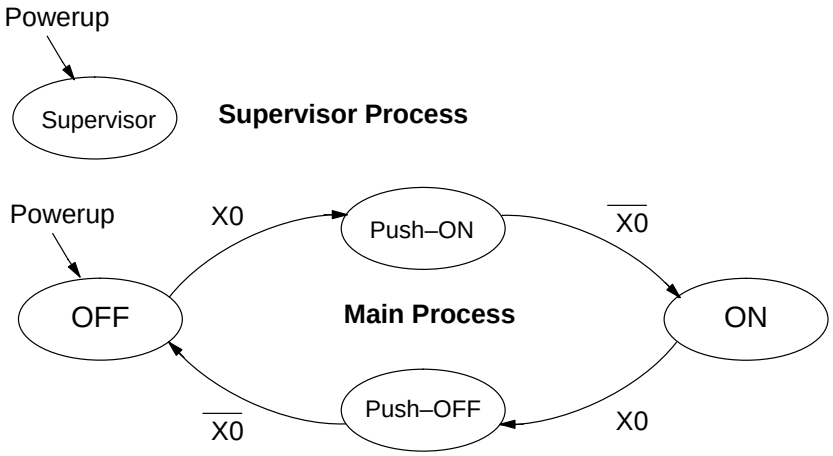
Stage Counter – The Stage Counter has the benefit that its count may be globally reset from other stages by using the RST instruction. It has a count input, but no reset input. This is the only difference from a standard counter instruction.

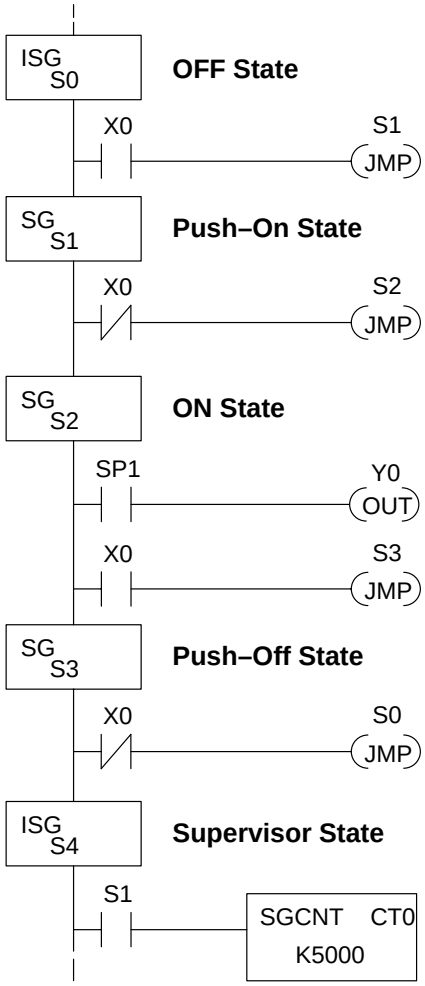
Drum – Realize the drum sequencer is its own process, and is a different programming method than stage programming. If you need to use a drum and stages, be sure to place the drum instruction in an ISG stage that is always active.

Using a Stage as a Supervisory Process

You may recall the light bulb on-off controller example from earlier in this chapter. For the purpose of illustration, suppose we want to monitor the "productivity" of the lamp process, by counting the number of on-off cycles which occurs. This application will require the addition of a simple counter, but the key decision is in where to put the counter.







New stage programming students will typically try to place the counter inside one the the stages of the process they are trying to monitor. The problem with this approach is that the stage is active only part of the time. In order for the counter to count, the count input must transition from off to on at least one scan after its stage activates. Ensuring this requires extra logic that can be tricky.

In this case, we only need to add another supervisory stage as shown above, to "watch" the main process. The counter inside the supervisor stage uses the stage bit S1 of the main process as its count input. *Stage bits used as a contact let us monitor a process!*

Note that both the Supervisor stage and the OFF stage are initial stages. The supervisor stage remains active indefinitely.

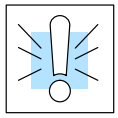
Stage Counter

The counter in the above example is a special Stage Counter. Note that it does not have a reset input. The count is reset by executing a Reset instruction, naming the counter bit (CT0 in this case). The Stage Counter has the benefit that its count may be globally reset from other stages. The standard Counter instruction does not have this global reset capability. You may still use a regular Counter instruction inside a stage... however, the reset input to the counter is the only way to reset it.

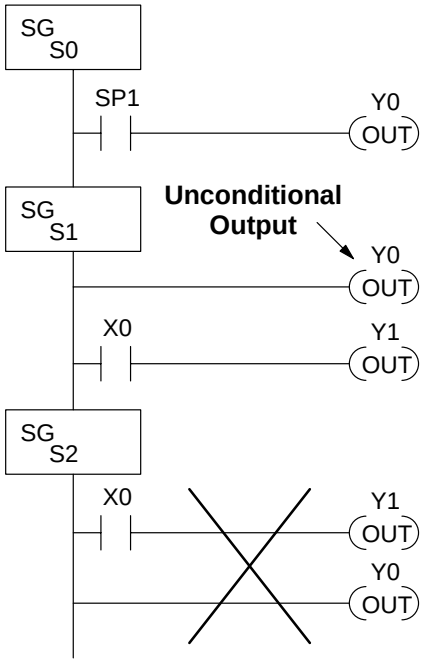
Unconditional Outputs

As in most example programs in this chapter and Stage 0 to the right, your application may require a particular output to be ON unconditionally when a particular stage is active. Until now, the examples always use the SP1 special relay contact (always on) in series with the output coils.

It's possible to omit the contact, as long as you place any unconditional outputs first (at the top) of a stage section of ladder. The first rung of Stage 1 does this.

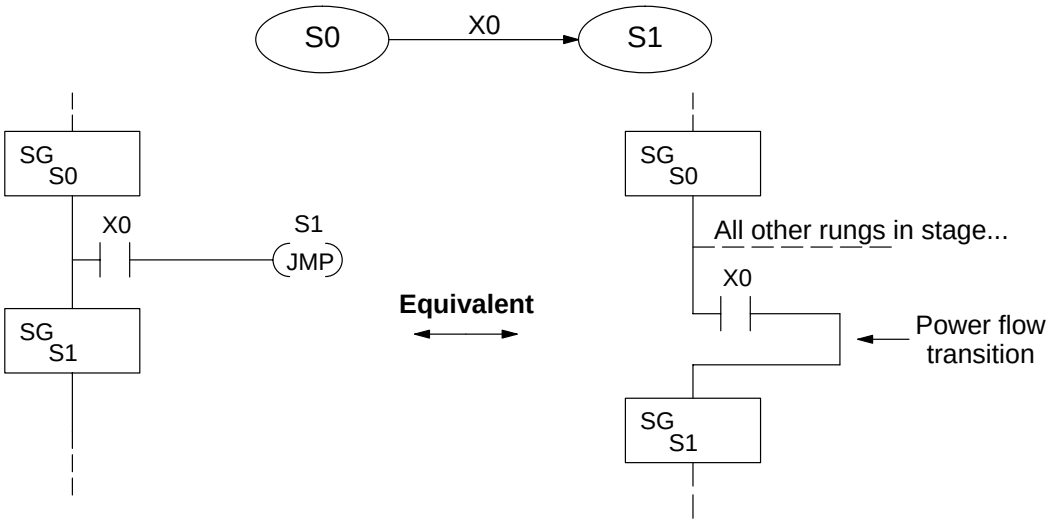


WARNING: Unconditional outputs placed elsewhere in a stage do not necessarily remain on when the stage is active. In Stage 2 to the right, Y0 is shown as an unconditional output, but its powerflow comes from the rung above. So, Y0 status will be the same as Y1 (is not correct).



Power Flow Transition Technique

Our discussion of state transitions has shown how the Stage JMP instruction makes the current stage inactive and the next stage (named in the JMP) active. As an alternative way to enter this in ***Direct***SOFT, you may use the power flow method for stage transitions. The main requirement is the current stage be located directly above the next (jump-to) stage in the ladder program. This arrangement is shown in the diagram below, by stages S0 and S1, respectively.



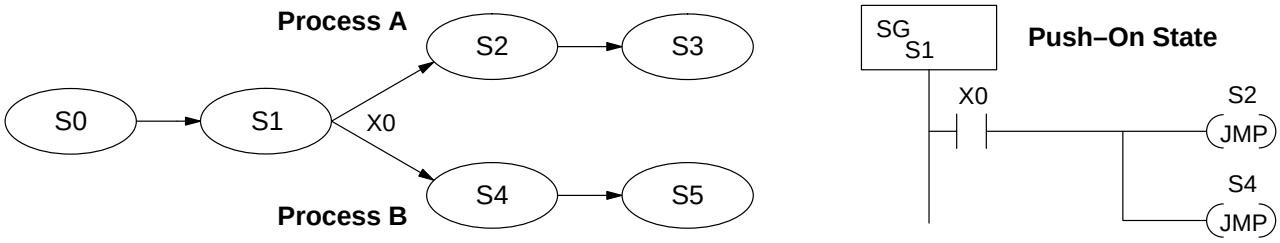
Recall the Stage JMP instruction may occur anywhere in the current stage, and the result is the same. However, power flow transitions (shown above) must occur as the last rung in a stage. All other rungs in the stage will precede it. The power flow transition method is also achievable on the handheld programmer, by simply following the transition condition with the Stage instruction for the next stage.

The power flow transition method does eliminate one Stage JMP instruction, its only advantage. However, it is not as easy to make program changes as using the Stage JMP. Therefore, we advise using Stage JMP transitions for most programs.

Parallel Processing Concepts

Parallel Processes

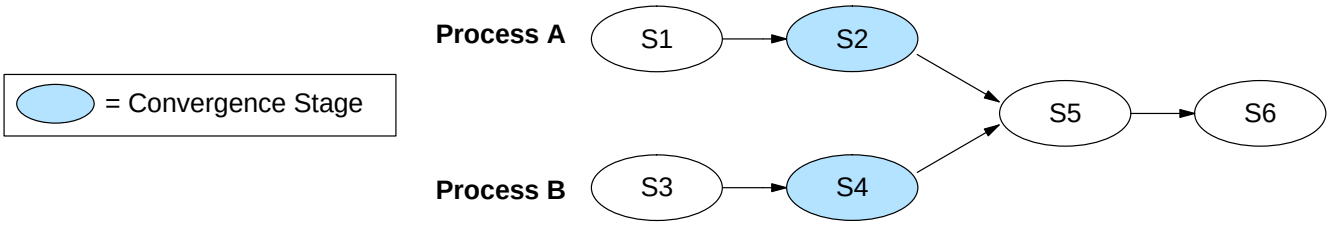
Previously in this chapter we discussed how a state may transition to either one state or another, called an *exclusive transition*. In other cases, we may need to branch simultaneously to two or more parallel processes, as shown below. It is acceptable to use all JMP instructions as shown, or we could use one JMP and a Set Stage bit instruction(s) (at least one must be a JMP, in order to leave S1). Remember that all instructions in a stage execute, even when it transitions (the JMP is not a GOTO).



Note that if we want Stages S2 and S4 to energize exactly on the same scan, both stages must be located below or above Stage S1 in the ladder program (see the explanation at the bottom of page 7–7). Overall, parallel branching is easy!

Converging Processes

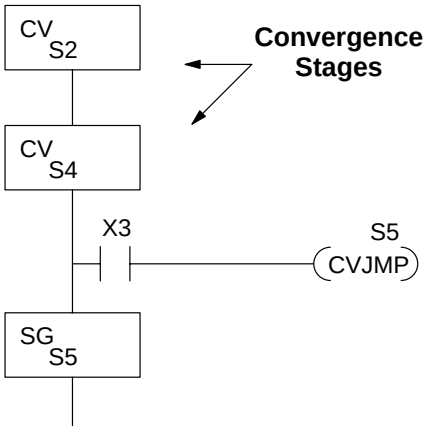
Now we consider the opposite case of parallel branching, which is *converging processes*. This simply means we stop doing multiple things and continue doing one thing at a time. In the figure below, processes A and B converge when stages S2 and S4 transition to S5 at some point in time. So, S2 and S4 are *Convergence Stages*.



Convergence Stages (CV)

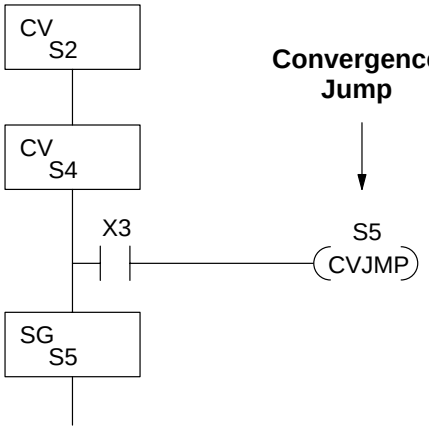
While the converging principle is simple enough, it brings a new complication. As parallel processing completes, the multiple processes almost never finish at the same time. In other words, how can we know whether Stage S2 or S4 will finish last? This is an important point, because we have to decide how to transition to Stage S5.

The solution is to coordinate the transition condition out of convergence stages. We accomplish this with a stage type designed for this purpose: the Convergence Stage (type CV). In the example to the right, convergence stages S2 and S4 are required to be grouped together as shown. **No logic is permitted between CV stages!** The transition condition (X3 in this case) must be located in the last convergence stage. The transition condition only has power flow when all convergence stages in the group are active.



Convergence Jump (CVJMP)

Recall the last convergence stage only has power flow when all CV stages in the group are active. To complement the convergence stage, we need a new jump instruction. The Convergence Jump (CVJMP) shown to the right will transition to Stage S5 when X3 is active (as one might expect), but it also *automatically resets all convergence stages in the group*. This makes the CVJMP jump a very powerful instruction. Note that this instruction may only be used with convergence stages.



Convergence Stage Guidelines

The following summarizes the requirements in the use of convergence stages, including some tips for their effective application:

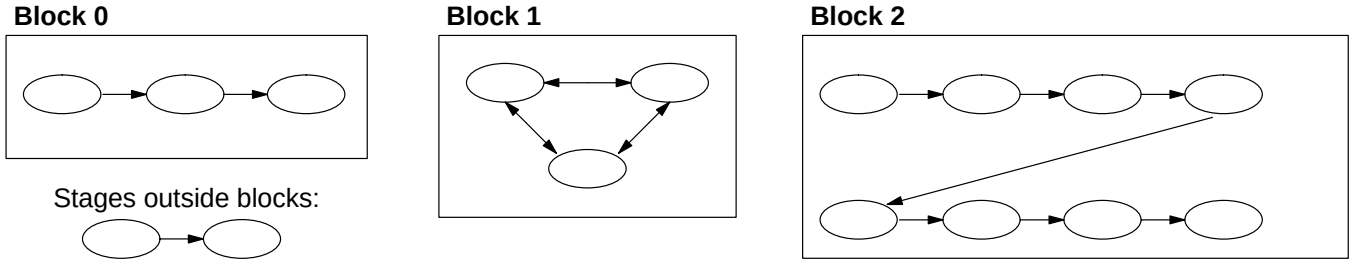
- A convergence stage is to be used as the last stage of a process which is running in parallel to another process or processes. A transition to the convergence stage means that a particular process is through, and represents a waiting point until all other parallel processes also finish.
- The maximum number of convergence stages which make up one group is 17. In other words, a maximum of 17 stages can converge into one stage.
- Convergence stages of the same group must be placed together in the program, connected on the power rail without any other logic in between.
- Within a convergence group, the stages may occur in any order, top to bottom. It does not matter which stage is last in the group, because all convergence stages have to be active before the last stage has power flow.
- The last convergence stage of a group may have ladder logic within the stage. However, this logic will not execute until all convergence stages of the group are active.
- The convergence jump (CVJMP) is the intended method to be used to transition from the convergence group of stages to the next stage. The CVJMP resets all convergence stages of the group, and energizes the stage named in the jump.
- The CVJMP instruction must only be used in a convergence stage, as it is invalid in regular or initial stages.
- Convergence Stages or CVJMP instructions may not be used in subroutines or interrupt routines.

Managing Large Programs

A stage may contain a lot of ladder rungs, or only one or two program rungs. For most applications, good program design will ensure the average number of rungs per stage will be small. However, large application programs will still create a large number of *stages*. We introduce a new construct which will help us organize related stages into groups called *blocks*. So, program organization is the main benefit of the use of stage blocks.

Stage Blocks (BLK, BEND)

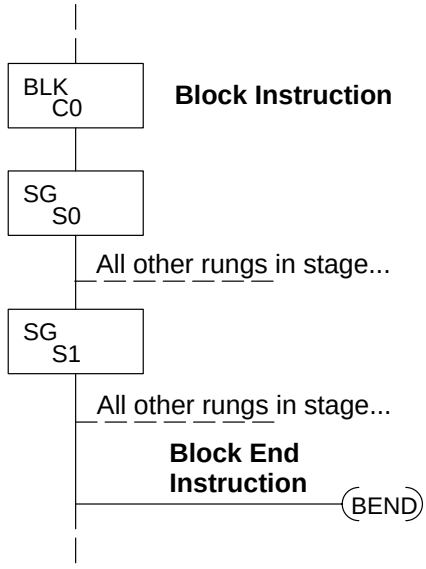
A block is a section of ladder program which contains stages. In the figure below, each block has its own reference number. Like stages, a stage block may be active or inactive. Stages inside a block are not limited in how they may transition from one to another. Note the use of stage blocks does not require each stage in a program to reside inside a block, shown below by the “stages outside blocks”.



A program with 20 or more stages may be considered large enough to use block grouping (however, their use is not mandatory). When used, the number of stage blocks should probably be two or higher, because the use of one block provides a negligible advantage.

A block of stages is separated from other ladder logic with special beginning and ending instructions. In the figure to the right, the BLK instruction at the top marks the start of the stage block. At the bottom, the Block End (BEND) marks the end of the block. The stages in between these boundary markers (S0 and S1 in this case) and their associated rungs make up the block.

Note the block instruction has a reference value field (set to “C0” in the example). The block instruction borrows or uses a control relay contact number, so that other parts of the program can control the block. *Any control relay number (such as C0) used in a BLK instruction is not available for use as a control relay.*

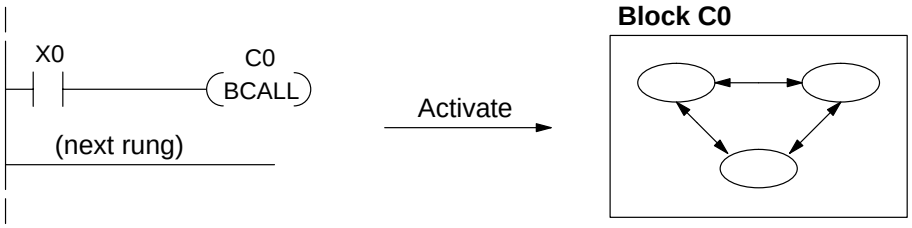


Note the stages within a block must be regular stages (SG) or convergence stages (CV). So, they cannot be initial stages. The numbering of stages inside stage blocks can be in any order, and is completely independent from the numbering of the blocks.

Block Call (BCALL)

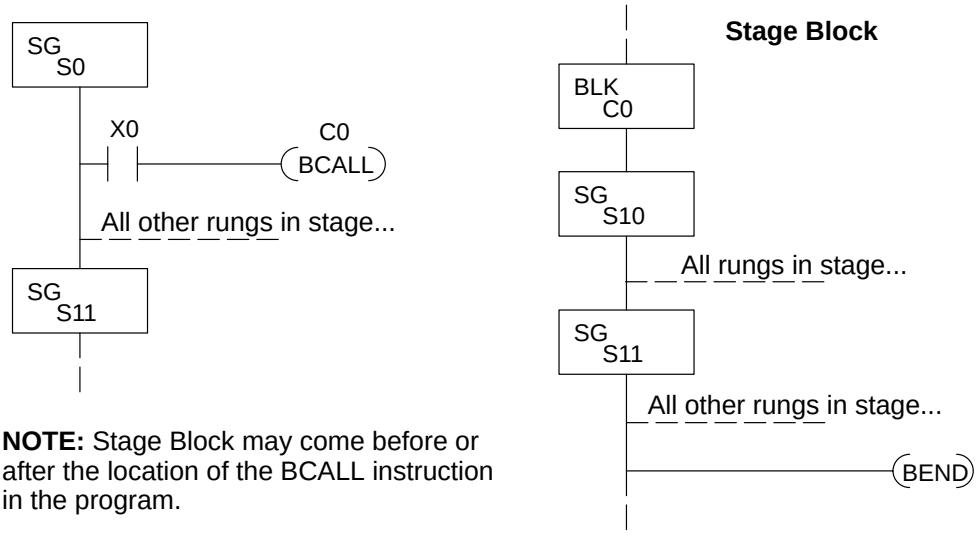
The purpose of the Block Call instruction is to activate a stage block. At powerup or upon Program-to-Run mode transitions, all stage blocks and the stages within them are inactive. Shown in the figure below, the Block Call instruction is a type of output coil. When the X0 contact is closed, the BCALL will cause the stage block referenced in the instruction (C0) to become active. When the BCALL is turned off, the corresponding stage block and the stages within it become inactive.

We must avoid confusing block call operation with how a “subroutine call” works. After a BCALL coil executes, program execution continues with the next program rung. Whenever program execution arrives at the ladder location of the stage block named in the BCALL, then logic within the block executes because the block is now active. Similarly, do not classify the BCALL as type of state transition (is not a JMP).



When a stage block becomes active, the first stage in the block automatically becomes active on the same scan. The “first” stage in a block is the one located immediately under the block (BLK) instruction in the ladder program. So, that stage plays a similar role to the initial type stage we discussed earlier.

The Block Call instruction may be used in several contexts. Obviously, the first execution of a BCALL must occur outside a stage block, since stage blocks are initially inactive. Still, the BCALL may occur on an ordinary ladder rung, or it may occur within an active stage as shown below. Note that either turning off the BCALL or turning off the stage containing the BCALL will deactivate the corresponding stage block. You may also control a stage block with a BCALL in another stage block.



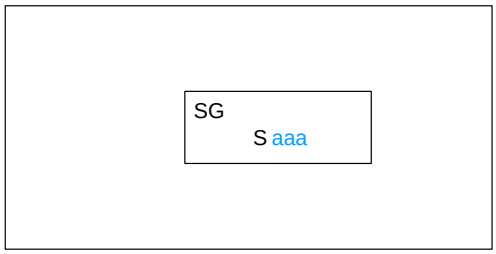
NOTE: Stage Block may come before or after the location of the BCALL instruction in the program.

The BCALL may be used in many ways or contexts, so it can be difficult to find the best usage. Remember the purpose of stage blocks is to help you organize the application problem by grouping related stages together. Remember that initial stages must exist *outside* stage blocks.

RLL*PLUS* Instructions

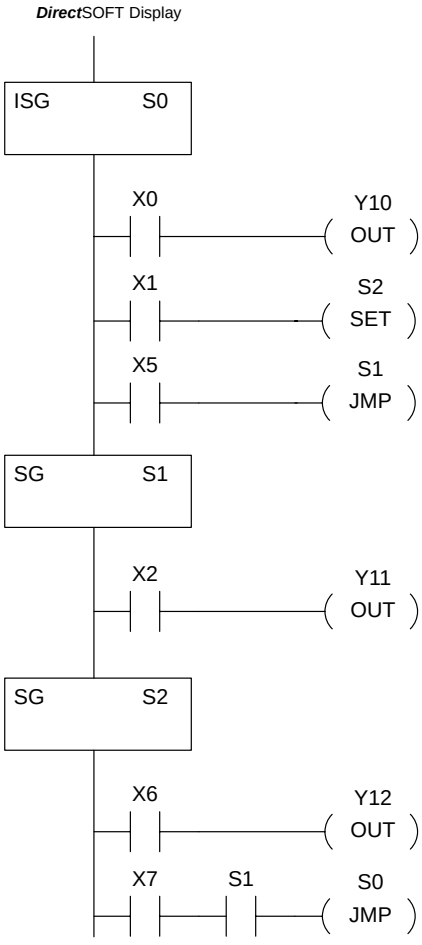
Stage (SG)

The Stage instructions are used to create structured RLL*PLUS* programs. Stages are program segments which can be activated by transitional logic, a jump or a set stage that is executed from an active stage. Stages are deactivated one scan after transitional logic, a jump, or a reset stage instruction is executed.



Operand Data Type		DL350 Range
		aaa
Stage	S	0–1777

The following example is a simple RLL*PLUS* program. This program utilizes the initial stage, stage, and jump instruction to create a structured program.

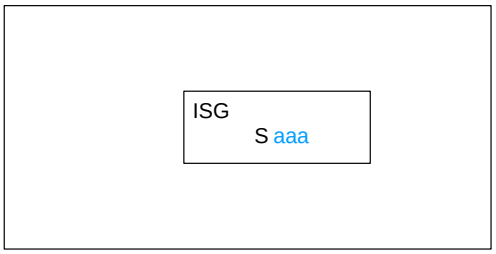


Handheld Programmer Keystrokes

ISG	→	S(SG)	0	ENT	
STR	→	X(IN)	0	ENT	
OUT	→	Y(OUT)	1	0	ENT
STR	→	X(IN)	1	ENT	
SET	→	S(SG)	2	ENT	
STR	→	X(IN)	5	ENT	
JMP	→	S(SG)	1	ENT	
SG	→	S(SG)	1	ENT	
STR	→	X(IN)	2	ENT	
OUT	→	Y(OUT)	1	1	
SG	→	S(SG)	2	ENT	
STR	→	X(IN)	6	ENT	
OUT	→	Y(OUT)	1	2	ENT
STR	→	X(IN)	7	ENT	
AND	→	S(SG)	1	ENT	
JMP	→	S(SG)	0	ENT	

Initial Stage (ISG)

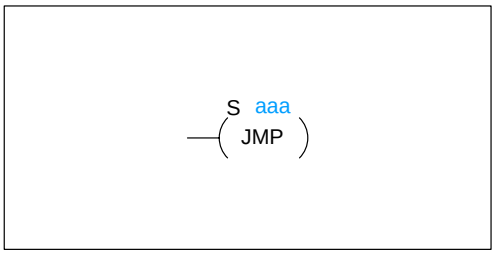
The Initial Stage instruction is normally used as the first segment of an RLL*PLUS* program. Initial stages will be active when the CPU enters the run mode allowing for a starting point in the program. Initial Stages are also activated by transitional logic, a jump or a set stage executed from an active stage. Initial Stages are deactivated one scan after transitional logic, a jump, or a reset stage instruction is executed. Multiple Initial Stages are allowed in a program.



Operand Data Type		DL350 Range
		aaa
Stage	S	0–1777

Jump (JMP)

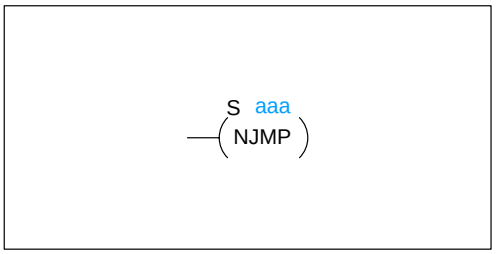
The Jump instruction allows the program to transition from an active stage which contains the jump instruction to another which stage is specified in the instruction. The jump will occur when the input logic is true. The active stage that contains the Jump will be deactivated 1 scan after the Jump instruction is executed.



Operand Data Type		DL350 Range
		aaa
Stage	S	0–1777

Not Jump (NJMP)

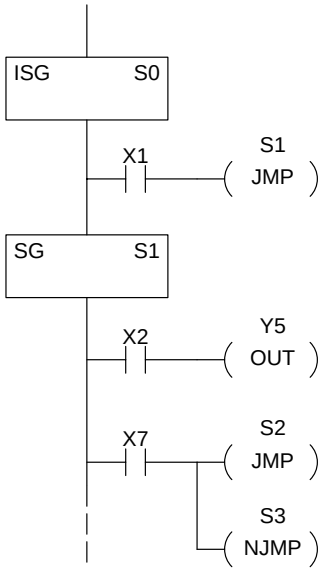
The Not Jump instruction allows the program to transition from an active stage which contains the jump instruction to another which is specified in the instruction. The jump will occur when the input logic is off. The active stage that contains the Not Jump will be deactivated 1 scan after the Not Jump instruction is executed.



Operand Data Type		DL350 Range
		aaa
Stage	S	0–1777

In the following example, when the CPU begins program execution only ISG 0 will be active. When X1 is on, the program execution will jump from Initial Stage 0 to Stage 1. In Stage 1, if X2 is on, output Y5 will be turned on. If X7 is on, program execution will jump from Stage 1 to Stage 2. If X7 is off, program execution will jump from Stage 1 to Stage 3.

DirectSOFT Display



Handheld Programmer Keystrokes

ISG	→	S(SG)	0	ENT
STR	→	X(IN)	1	ENT
JMP	→	S(SG)	1	ENT
SG	→	S(SG)	1	ENT
STR	→	X(IN)	2	ENT
OUT	→	Y(OUT)	5	ENT
STR	→	X(IN)	7	ENT
JMP	→	S(SG)	2	ENT
SHFT	N	JMP	→	
S(SG)	3	ENT		

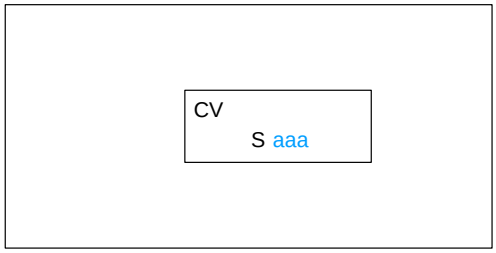
Converge Stage (CV) and Converge Jump (CVJMP)

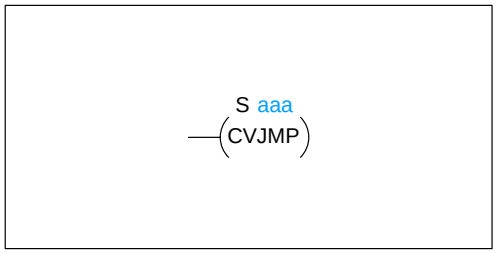
The Converge Stage instruction is used to group certain stages together by defining them as Converge Stages.

When all of the Converge Stages within a group become active, the CVJMP instruction (and any additional logic in the final CV stage) will be executed. All preceding CV stages *must* be active before the final CV stage logic can be executed. All Converge Stages are deactivated one scan after the CVJMP instruction is executed.

Additional logic instructions are only allowed following the last Converge Stage instruction and before the CVJMP instruction. Multiple CVJUMP instructions are allowed.

Converge Stages must be programmed in the main body of the application program. This means they cannot be programmed in Subroutines or Interrupt Routines.

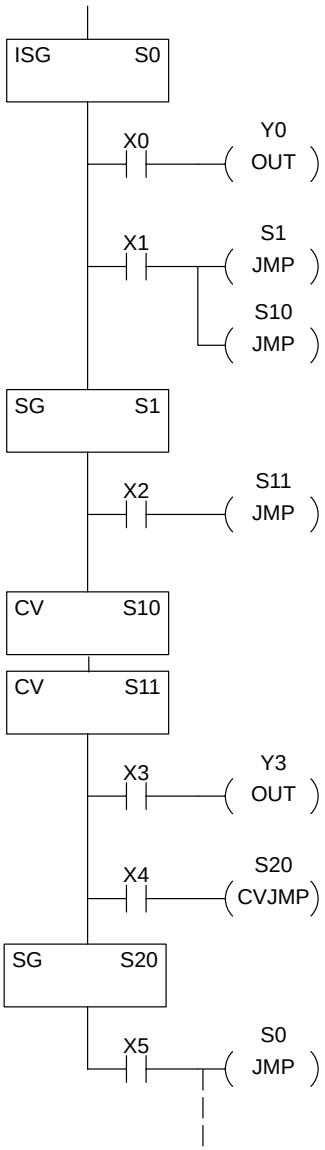




Operand Data Type		DL350 Range
		aaa
Stage	S	0–1777

In the following example, when Converge Stages S10 and S11 are *both* active the CVJMP instruction will be executed when X4 is on. The CVJMP will deactivate S10 and S11, and activate S20. Then, if X5 is on, the program execution will jump back to the initial stage, S0.

DirectSOFT Display



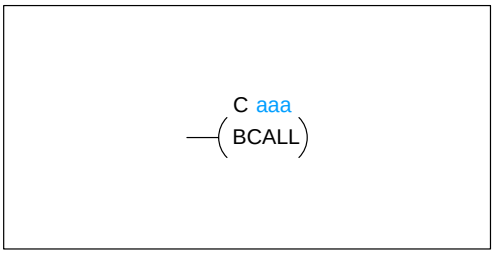
Handheld Programmer Keystrokes

ISG	→	S(SG)	0	ENT				
STR	→	X(IN)	0	ENT				
OUT	→	Y(OUT)	0	ENT				
STR	→	X(IN)	1	ENT				
JMP	→	S(SG)	1	ENT				
JMP	→	S(SG)	1	0	ENT			
SG	→	S(SG)	1	ENT				
STR	→	X(IN)	2	ENT				
JMP	→	S(SG)	1	1	ENT			
SHFT	C	V	→	S(SG)	1	0	ENT	
SHFT	C	V	→	S(SG)	1	1	ENT	
STR	→	X(IN)	3	ENT				
OUT	→	Y(OUT)	3	ENT				
STR	→	X(IN)	4	ENT				
SHFT	C	V	SHFT	JMP	S(SG)	2	0	ENT
SG	→	S(SG)	2	0	ENT			
STR	→	X(IN)	5	ENT				
JMP	→	S(SG)	0	ENT				

The stage block instructions are used to activate a block of stages. The Block Call, Block, and Block End instructions must be used together.

Block Call (BCALL)

The BCALL instruction is used to activate a stage block. There are several things you need to know about the BCALL instruction.



Uses CR Numbers — The BCALL appears as an output coil, but does not actually refer to a Stage number as you might think. Instead, the block is identified with a Control Relay (Caaa). This control relay cannot be used as an output anywhere else in the program.

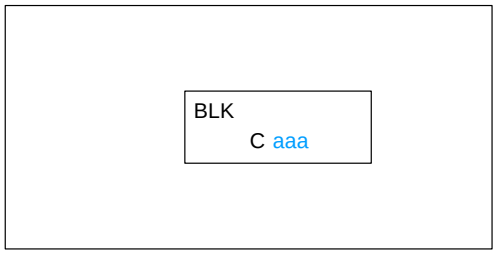
Must Remain Active — The BCALL instruction actually controls all the stages between the BLK and the BEND instructions even after the stages inside the block have started executing. The BCALL must *remain* active or all the stages in the block will automatically be turned off. *If either the BCALL instruction, or the stage that contains the BCALL instruction goes off, then the stages in the defined block will be turned off automatically.*

Activates First Block Stage — When the BCALL is executed it automatically activates the first stage following the BLK instructions.

Operand Data Type		DL350 Range
		aaa
Control Relay	C	0–1777

Block (BLK)

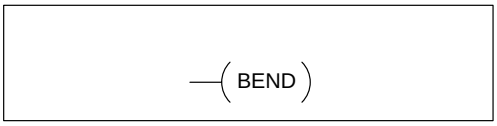
The Block instruction is a label which marks the beginning of a block of stages that can be activated as a group. A Stage instruction must immediately follow the Start Block instruction. Initial Stage instructions are not allowed in a block. The control relay (Caaa) specified in Block instruction must not be used as an output any where else in the program.



Operand Data Type		DL350 Range
		aaa
Control Relay	C	0–1777

Block End (BEND)

The Block End instruction is a label used with the Block instruction. It marks the end of a block of stages. There is no operand with this instruction. Only one Block End is allowed per Block Call.

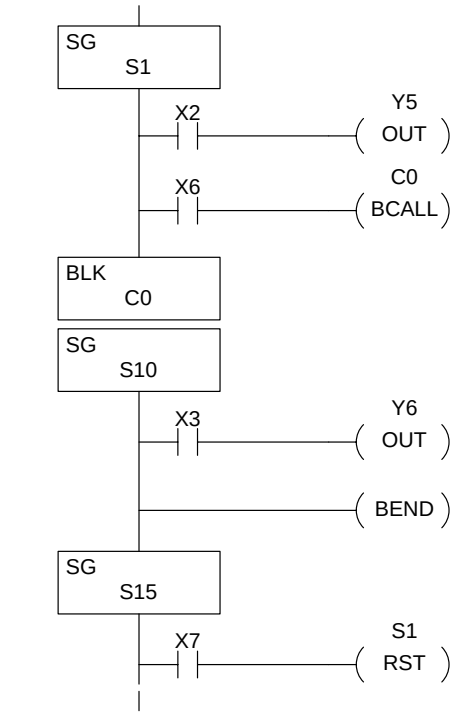


In this example, the Block Call is executed when stage 1 is active and X6 is on. The Block Call then automatically activates stage S10, which immediately follows the Block instruction.

This allows the stages between S10 and the Block End instruction to operate as programmed. If the BCALL instruction is turned off, or if the stage containing the BCALL instruction is turned off, then *all* stages between the BLK and BEND instructions are automatically turned off.

If you examine S15, you will notice that X7 could reset Stage S1, which would disable the BCALL, thus resetting all stages within the block.

DirectSOFT Display

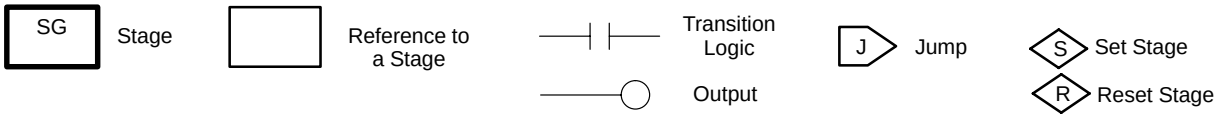


Handheld Programmer Keystrokes

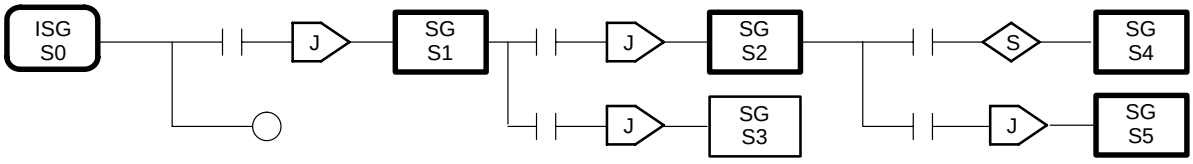
SG	→	S(SG)	1	ENT					
STR	→	X(IN)	2	ENT					
OUT	→	Y(OUT)	5	ENT					
STR	→	X(IN)	6	ENT					
SHFT	B	C	A	L	L	→	C(CR)	0	ENT
SHFT	B	L	K	→	C(CR)	0	ENT		
SG	→	S(SG)	1	0	ENT				
STR	→	X(IN)	3	ENT					
OUT	→	Y(OUT)	6	ENT					
SHFT	B	E	N	D	ENT				
SG	→	S(SG)	1	5	ENT				
STR	→	X(IN)	7	ENT					
RST	→	S(SG)	1	ENT					

Stage View in *Direct*SOFT

The Stage View option in ***Direct***SOFT will let you view the ladder program as a flow chart. The figure below shows the symbol convention used in the diagrams. You may find the stage view useful as a tool to verify that your stage program has faithfully reproduced the logic of the state transition diagram you intend to realize.



The following diagram is a typical stage view of a ladder program containing stages. Note the left-to-right direction of the flow chart.



Questions and Answers about Stage Programming

We include the following commonly-asked questions about Stage Programming as an aid to new students. All question topics are covered in more detail in this chapter.

Q. What does stage programming do that I cannot do with regular RLL programs?

A. Stages allow you to identify all the states of your process before you begin programming. This approach is more organized, because you divide up a ladder program into sections. As stages, these program sections are active only when they are actually needed by the process. Most processes can be organized into a sequence of stages, connected by event-based transitions.

Q. Isn't a stage really like a software subroutine?

A. No, it is very different. A subroutine is called by a main program when needed, and executes only once before returning to the point from which it was called. A stage, however, is part of the main program. It represents a state of the process, so an active stage executes on every scan of the CPU until it becomes inactive.

Q. What are Stage Bits?

A. A stage bit is a single bit in the CPU's image register, representing the active/inactive status of the stage in real time. For example, the bit for Stage 0 is referenced as "S0". If S0 = 0, then the ladder rungs in Stage 0 are bypassed (not executed) on each CPU scan. If S0 = 1, then the ladder rungs in Stage 0 are executed on each CPU scan. Stage bits, when used as contacts, allow one part of your program to monitor another part by detecting stage active/inactive status.

Q. How does a stage become active?

A. There are three ways:

- If the Stage is an initial stage (ISG), it is automatically active at powerup.
- Another stage can execute a Stage JMP instruction naming this stage, which makes it active upon its next occurrence in the program.
- A program rung can execute a Set Stage Bit instruction (such as SET S0).

Q. How does a stage become inactive?

A. There are three ways:

- Standard Stages (SG) are automatically inactive at powerup.
- A stage can execute a Stage JMP instruction, resetting its Stage Bit to 0.
- Any rung in the program can execute a Reset Stage Bit instruction (such as RST S0).

Q. What about the power flow technique of stage transitions?

A. The power flow method of connecting adjacent stages (directly above or below in the program) actually is the same as the Stage Jump instruction executed in the stage above, naming the stage below. Power flow transitions are more difficult to edit in ***Direct***SOFT, we list them separately from two preceding questions.

Q. Can I have a stage which is active for only one scan?

A. Yes, but this is not the intended use for a stage. Instead, make a ladder rung active for 1 scan by including a stage Jump instruction at the bottom of the rung. Then the ladder will execute on the last scan before its stage jumps to a new one.

Q. Isn't a Stage JMP like a regular GOTO instruction used in software?

A. No, it is very different. A GOTO instruction sends the program execution immediately to the code location named by the GOTO. A Stage JMP simply resets the Stage Bit of the current stage, while setting the Stage Bit of the stage named in the JMP instruction. Stage bits are 0 or 1, determining the inactive/active status of the corresponding stages. A stage JMP has the following results:

- When the JMP is executed, the remainder of the current stage's rungs are executed, even if they reside past(under) the JMP instruction. On the following scan, that stage is not executed, because it is inactive.
- The Stage named in the Stage JMP instruction will be executed upon its next occurrence. If located past (under) the current stage, it will be executed on the same scan. If located before (above) the current stage, it will be executed on the following scan.

Q. How can I know when to use stage JMP, versus a Set Stage Bit or Reset Stage Bit?

A. These instructions are used according to the state diagram topology you have derived:

- Use a Stage JMP instruction for a state transition... moving from one state to another.
- Use a Set Stage Bit instruction when the current state is spawning a new parallel state or stage sequence, or when a supervisory state is starting a state sequence under its command.
- Use a Reset Stage Bit instruction when the current state is the last state in a sequence and its task is complete, or when a supervisory state is ending a state sequence under its command.

Q. What is an initial stage, and when do I use it?

A. An initial stage (ISG) is automatically active at powerup. Afterwards, it works like any other stage. You can have multiple initial stages, if required. Use an initial stage for ladder that must always be active, or as a starting point.

Q. Can I place program ladder rungs outside of the stages, so they are always on?

A. It is possible, but it's not good software design practice. Place ladder that must always be active in an initial stage, and do not reset that stage or use a Stage JMP instruction inside it. It can start other stage sequences at the proper time by setting the appropriate Stage Bit(s).

Q. Can I have more than one active stage at a time?

A. Yes, and this is a normal occurrence for many programs. However, it is important to organize your application into separate processes, each made up of stages. And a good process design will be mostly sequential, with only one stage on at a time. However, all the processes in the program may be active simultaneously.

8

PID Loop Operation

In This Chapter. . . .

- DL350 PID Loop Features
- Loop Setup Parameters
- Loop Sample Rate and Scheduling
- Ten Steps to Successful Process Control
- Basic Loop Operation
- PID Loop Data Configuration
- PID Algorithms
- Loop Tuning Procedure
- Feedforward Control
- Time Proportioning Control
- Cascade Control
- Process Alarms
- Ramp/Soak Generator
- Troubleshooting Tips
- Bibliography
- Glossary of PID Loop Terminology

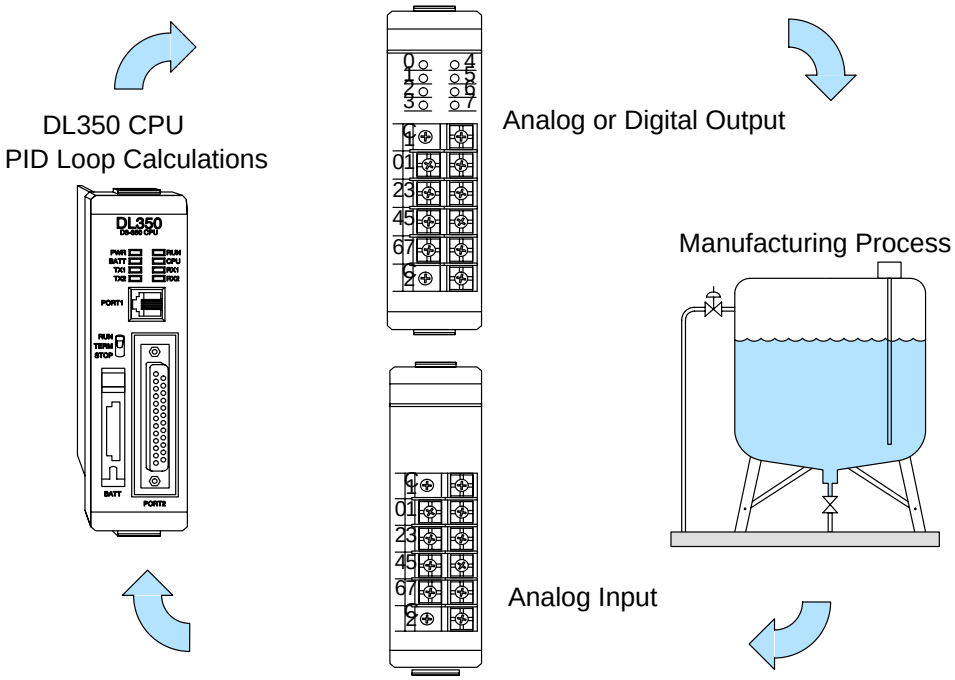
DL350 PID Loop Features

Main Features

The DL350 process loop control offers a sophisticated set of features to address many application needs. The main features are:

- Up to 4 loops, individual programmable sample rates
- Manual/Automatic/Cascaded loop capability available
- Two types of bumpless transfer available
- Full-featured alarms
- Ramp/soak generator with up to 16 segments
- Auto Tuning

The DL350 CPU has process control loop capability in addition to ladder program execution. You can select and configure up to four loops. All sensor and actuator wiring connects to standard DL305 I/O modules, as shown below. All process variables, gain values, alarm levels, etc., associated with each loop reside in a Loop Variable Table in the CPU. The DL350 CPU reads process variable (PV) inputs during each scan. Then it makes PID loop calculations during a dedicated time slice on each PLC scan, updating the control output value. The control loops use the Proportional-Integral-Derivative (PID) algorithm to generate the control output command. This chapter describes how the loops operate,and what you must do to configure and tune the loops.



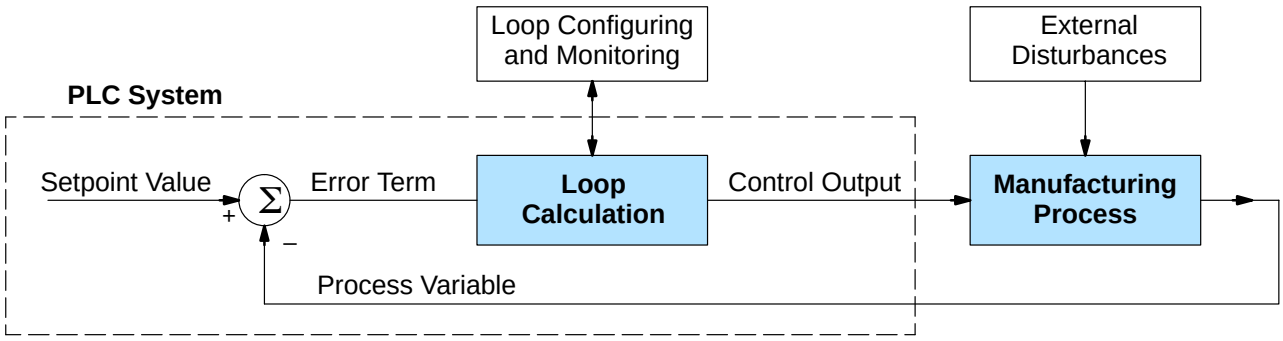
The best tool for configuring loops in the DL350 is the ***Direct***SOFT programming software, Release 2.2 or later. ***Direct***SOFT uses dialog boxes to create a forms-like editor to let you individually set up the loops. After completing the setup, you can use ***Direct***SOFT's PID Trend View to tune each loop. The loop parameters also may be saved to disk for recall later.

PID Loop Feature	Specifications
Number of loops	Selectable, 4 maximum
CPU V-memory needed	32 words (V locations) per loop selected, 64 words if using ramp/soak
PID algorithm	Position or Velocity form of the PID equation
Control Output polarity	Selectable direct-acting or reverse-acting
Error term curves	Selectable as linear, square root of error, and error squared
Loop update rate (time between PID calculation)	0.05 to 99.99 seconds, user programmable
Minimum loop update rate	0.05 seconds for 1 to 4 loops,
Loop modes	Automatic, Manual (operator control), or Cascade control
Ramp/Soak Generator	Up to 8 ramp/soak steps (16 segments) per loop with indication of ramp/soak step number
PV curves	Select standard linear, or square-root extract (for flow meter input)
Set Point Limits	Specify minimum and maximum setpoint values
Process Variable Limits	Specify minimum and maximum Process Variable values
Proportional Gain	Specify gains of 0.01 to 99.99
Integrator (Reset)	Specify reset time of 0.1 to 999.8 in units of seconds or minutes
Derivative (Rate)	Specify the derivative time from 0.01 to 99.99 seconds
Rate Limits	Specify derivative gain limiting from 1 to 20
Bumpless Transfer I	Automatically initialized bias and setpoint when control switches from manual to automatic
Bumpless Transfer II	Automatically set the bias equal to the control output when control switches from manual to automatic
Step Bias	Provides proportional bias adjustment for large setpoint changes
Anti-windup	For position form of PID, this inhibits integrator action when the control output reaches 0% or 100 % (speeds up loop recovery when output recovers from saturation)
Error Deadband	Specify a tolerance (plus and minus) for the error term (SP–PV), so that no change in control output value is made

Alarm Feature	Specifications
Deadband	Specify 0.1% to 5% alarm deadband on all alarms
PV Alarm Points	Select PV alarm settings for Low–low, Low, High, and High-high conditions
PV Deviation	Specify alarms for two ranges of PV deviation from the setpoint value
Rate of Change	Detect when PV exceeds a rate of change limit you specify

Getting Acquainted with PID Loops

As an introduction to key parts of a control loop, refer to the block diagram shown below. The closed path around the diagram is the “loop” referred to in “closed loop control”.



Manufacturing Process – the set of actions that adds value to raw materials. The process can involve physical changes and/or chemical changes to the material. The changes render the material more useful for a particular purpose, ultimately used in a final product.

Process Variable – a measurement of some physical property of the raw materials. Measurements are made using some type of sensor. For example, if the manufacturing process uses an oven, we will have a strong interest in controlling temperature. Therefore, temperature is a process variable.

Setpoint Value – the theoretically perfect quantity of the process variable, or the desired amount which yields the best product. The machine operator knows this value, and either sets it manually or programs it into the PLC for later automated use.

External Disturbances – the unpredictable sources of error which the control system attempts to cancel by offsetting their effects. For example, if the fuel input is constant an oven will run hotter during warm weather than it does during cold weather. An oven control system must counter-act this effect to maintain a constant oven temperature during any season. Thus, the weather (which is not very predictable), is one source of disturbance to this process.

Error Term – the algebraic difference between the process variable and the setpoint. This is the control loop error, and is equal to zero when the process variable is equal to the setpoint (desired) value. A well-behaved control loop is able to maintain a small error term magnitude.

Loop Calculation – the real-time application of a mathematical algorithm to the error term, generating a control output command appropriate for minimizing the error magnitude. Various control algorithms are available, and the DL350 uses the Proportional-Derivative-Integral (PID) algorithm (more on this later).

Control Output – the result of the loop calculation, which becomes a command for the process (such as the heater level in an oven).

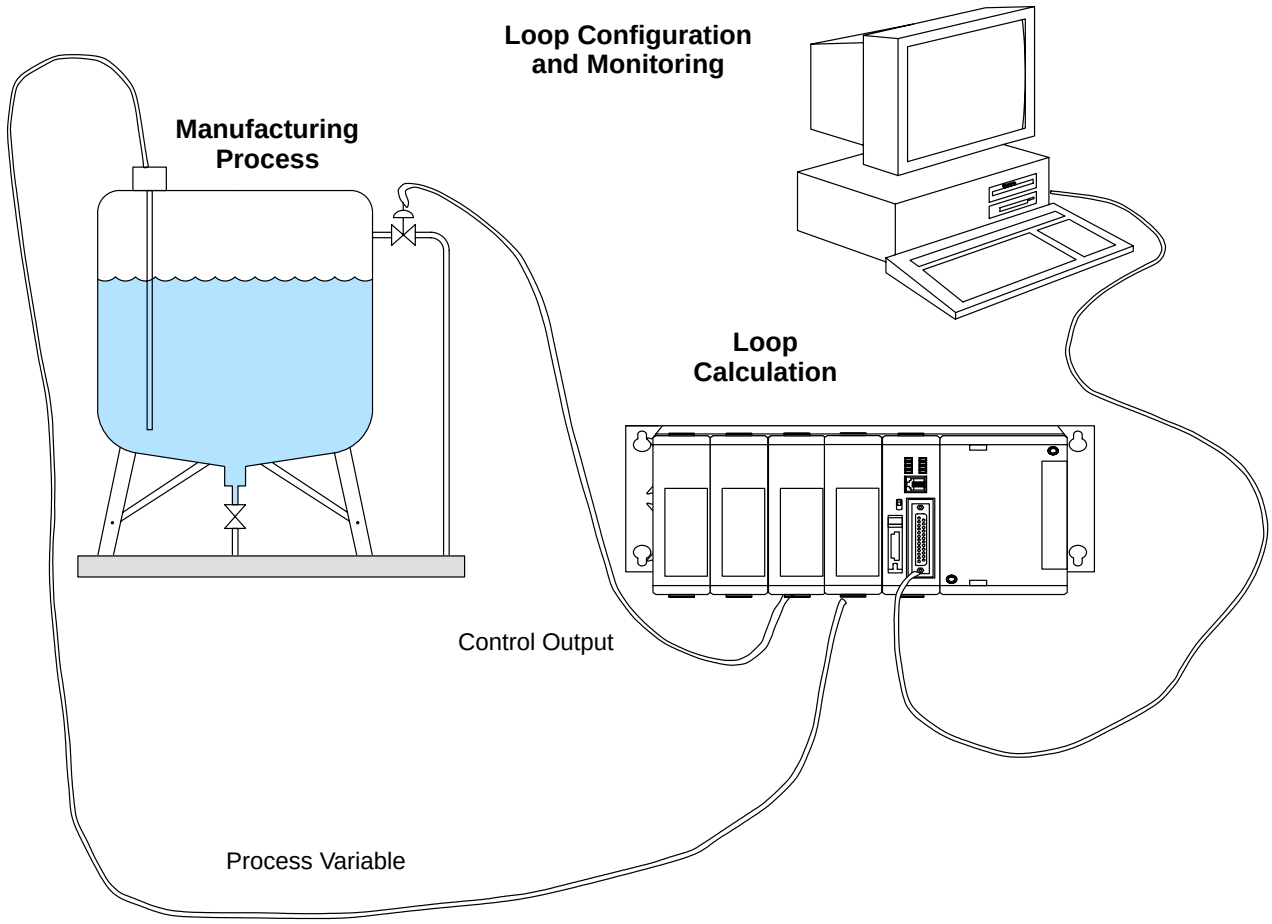
Loop Configuring – operator-initiated selections which set up and optimize the performance of a control loop. The loop calculation function uses the configuration parameters in real time to adjust gains, offsets, etc.

Loop Monitoring – the function which allows an operator to observe the status and performance of a control loop. This is used in conjunction with the loop configuring to optimize the performance of a loop (minimize the error term).

While developing an understanding of process control loops, it is important to associate each loop element with its real-world physical component. Refer to the following figure. The example manufacturing process involves some amount of liquid in a reactor vessel. A sensor probe measures a process variable which may be pressure, temperature, or another parameter. The sensor signal is amplified through a transducer, and is sent through the wire in analog form to the PLC input module.

The PLC reads the PV from an analog input. The CPU executes the loop calculation, and writes to the analog output module location.

The control output signal may be analog (proportional) or digital (on/off), depending on loop setup. This signal goes to a device in the manufacturing process which amplifies it in order to effect a physical or chemical change on the liquid in the reactor. The amplifier may drive a heater, valve, pump, etc. Over time, the liquid begins to change enough to be measured on the sensor probe. The process variable changes accordingly. The next loop calculation occurs, and the loop cycle repeats in this manner continuously.



The personal computer shown is to run ***Direct***SOFT, the PLC programming software for ***Direct***LOGIC programmable controllers. ***Direct***SOFT Release 2.2 or later can program the DL350 CPU. The software features a forms-based editor to configure loop parameters. Most importantly, it features a PID loop trending screen which will be invaluable during the loop tuning process. Details on how to use that software are in the ***Direct***SOFT Manual.

Loop Setup Parameters

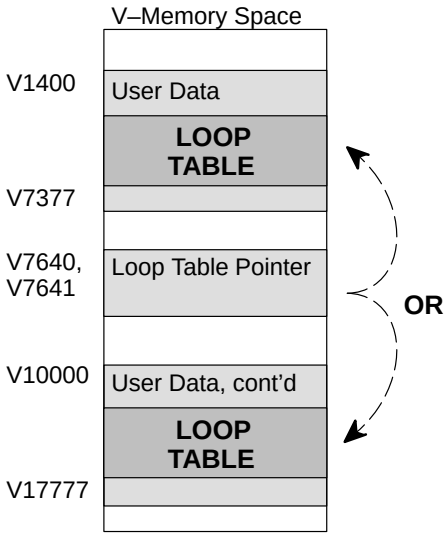
Loop Table and Number of Loops

The DL350 CPU gets its PID loop processing instructions solely from tables in V-memory. This means that a “PID instruction” type in RLL does not exist. Instead, the CPU reads setup parameters from reserved V-memory locations. Shown in the table below, you must program a value in V7640 to point to the main loop table. Then you will need to program V7641 with the number of loops you want the CPU to calculate. V7642 contains error flags which will be set if V7640 or V7641 are programmed improperly.

Address	Setup Parameter	Data type	Ranges	Read/Write
V7640	Loop Parameter Table Pointer	Octal	V1400 – V7340, V10000 – V17740	write
V7641	Number of Loops	BCD	0 – 4	write
V7642	Loop Error Flags	Binary	0 or 1	read

If the number of loops is “0”, the loop controller task is turned off during the ladder program scan. The loop controller will allow use of loops in ascending order, beginning with 1. For example, you cannot use loop 1 and 4 while skipping 2 and 3. The loop controller attempts to control the full number of loops specified in V7641.

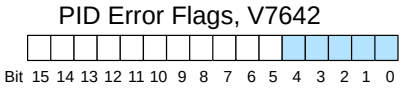
The Loop Parameter table may occupy a block of memory in the lower user data space (V1400 – V7377), or in the upper user memory data space (V10000 – V17777) as shown to the right. Be sure to choose an available space in the memory map for you application. The value in V7641 tells the CPU how big the loop table is, because there are 32 locations for each loop.



The ***Direct***SOFT PID Setup dialog box is a good way to program these parameters. It’s also possible to use ladder commands such a LDA or LD, and OUT instructions. However, these memory locations are part of the retentive system parameters, so writing them from RLL is not required.

PID Error Flags

The CPU reports any programming errors of the setup parameters in V7640 and V7641. It does this by setting the appropriate bits in V7642 on program-to-run mode transitions.



If you use the ***Direct***SOFT loop setup dialog box, its automatic range checking prohibits possible setup errors. However, the setup parameters may be written using other methods such as RLL, so the error flag register may be helpful in those cases.

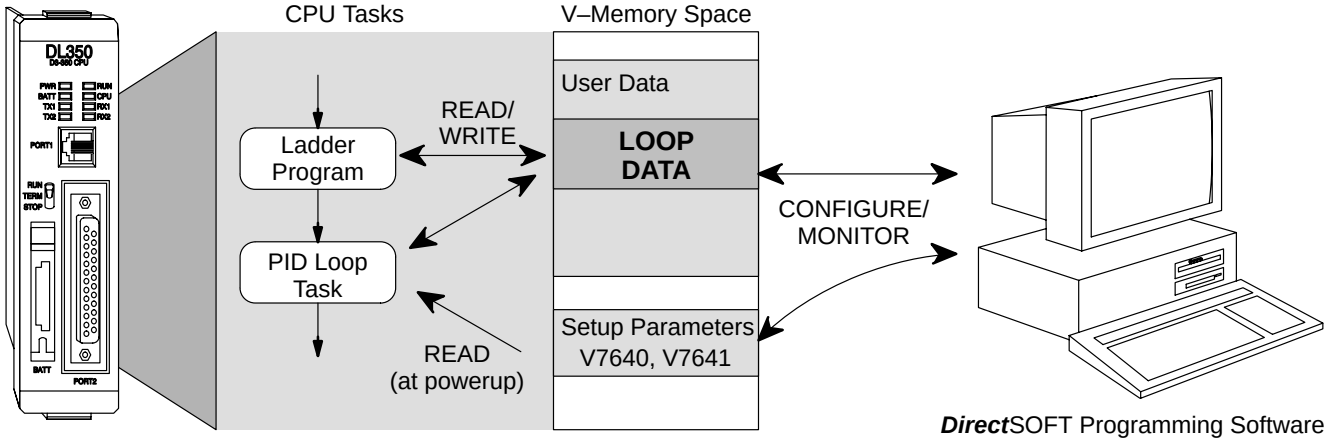
The following table lists the errors reported in V7642.

Bit	Error Description (0 = no error, 1 = error)
0	The starting address (in V7640) is out of the lower V-memory range.
1	The starting address (in V7640) is out of the upper V-memory range.
2	The number of loops selected (in V7641) is greater than 4.
3	The loop table extends past (straddles) the boundary at V7377. Use an address closer to V1400.
4	The loop table extends past (straddles) the boundary at V17777. Use an address closer to V10000.

As a quick check, if the CPU is in Run mode and V7642=0000, then we know there are no programming errors.

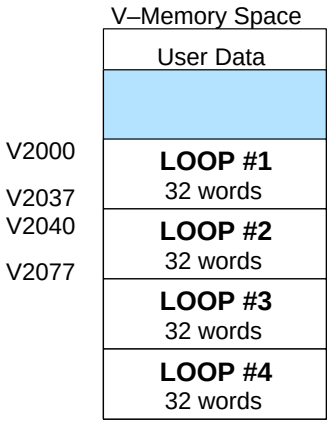
Establishing the Loop Table Size and Location

On a program-to-run mode transition, the CPU reads the loop setup parameters as pictured below. At that moment, the CPU learns the location of the loop table and the number of loops it configures. Then during the ladder program scan, the PID Loop task uses the loop data to perform calculations, generate alarms, and so on. There are some loop table parameters the CPU will read or write on every loop calculation.



The Loop Parameter table contains data for only as many loops selected by the value you have programmed in V7641. Each loop configured occupies 32 words (0 to 37 octal) in the loop table.

For example, suppose we have an application with 4 loops. Arbitrarily, we choose V2000 as the starting location. The Loop Parameter will occupy V2000 – V2037 for loop 1, V2040 – V2077 for loop 2 and so on. Loop 4 occupies V2140 – V2177.



Loop Table Word Definitions

The parameters associated with each loop are listed in the following table. The address offset is in octal, to help you locate specific parameters in a loop table. For example, if a table begins at V2000, then the location of the reset (integral) term is Addr+11, or V2011. Do not use the word# to calculate addresses.

Word #	Address+Offset	Description	Format
1	Addr + 0	PID Loop Mode Setting 1	bits
2	Addr + 1	PID Loop Mode Setting 2	bits
3	Addr + 2	Setpoint Value (SP)	word/binary
4	Addr + 3	Process Variable (PV)	word/binary
5	Addr + 4	Bias (Integrator) Value	word/binary
6	Addr + 5	Control Output Value	word/binary
7	Addr + 6	Loop Mode and Alarm Status	bits
8	Addr + 7	Sample Rate Setting	word/BCD
9	Addr + 10	Gain (Proportional) Setting	word/BCD
10	Addr + 11	Reset (Integral) Time Setting	word/BCD
11	Addr + 12	Rate (Derivative) Time Setting	word/BCD
12	Addr + 13	PV Value, Low-low Alarm	word/binary
13	Addr + 14	PV Value, Low Alarm	word/binary
14	Addr + 15	PV Value, High Alarm	word/binary
15	Addr + 16	PV Value, High-high Alarm	word/binary
16	Addr + 17	PV Value, deviation alarm (YELLOW)	word/binary
17	Addr + 20	PV Value, deviation alarm (RED)	word/binary
18	Addr + 21	PV Value, rate-of-change alarm	word/binary
19	Addr + 22	PV Value, alarm hysteresis setting	word/binary
20	Addr + 23	PV Value, error deadband setting	word/binary
21	Addr + 24	PV low-pass filter constant	word/BCD
22	Addr + 25	Loop derivative gain limiting factor setting	word/BCD
23	Addr + 26	SP value lower limit setting	word/binary
24	Addr + 27	SP value upper limit setting	word/binary
25	Addr + 30	Control output value lower limit setting	word/binary
26	Addr + 31	Control output value upper limit setting	word/binary
27	Addr + 32	Remote SP Value V-Memory Address Pointer	word/hex
28	Addr + 33	Ramp/Soak Setting Flag	bit
29	Addr + 34	Ramp/Soak Programming Table Starting Address	word/hex
30	Addr + 35	Ramp/Soak Programming Table Error Flags	bits
31	Addr + 36	PV direct access, slot/channel	word/hex
32	Addr + 37	Control output direct access, slot/channel	word/hex

PID Mode Setting 1 Bit Descriptions (Addr + 00)

The individual bit definitions of PID Mode Setting 1 (Addr+00) word are listed in the following table. More details are in the section related to each bit later in this chapter.

Bit	PID Mode Setting 1 Description	Read/Write	Bit=0	Bit=1
0	Manual Mode Loop Operation request	write	–	0→1 request
1	Automatic Mode Loop Operation request	write	–	0→1 request
2	Cascade Mode Loop Operation request	write	–	0→1 request
3	Bumpless Transfer select	write	Mode I	Mode II
4	Direct or Reverse-Acting Loop select	write	Direct	Reverse
5	Position / Velocity Algorithm select	write	Position	Velocity
6	PV Linear / Square Root Extract select	write	Linear	Sq. root
7	Error Term Linear / Squared select	write	Linear	Squared
8	Error Deadband enable	write	Disable	Enable
9	Derivative Gain Limit select	write	Off	On
10	Bias (Integrator) Freeze select	write	Off	On
11	Ramp/Soak Operation select	write	Off	On
12	PV Alarm Monitor select	write	Off	On
13	PV Deviation alarm select	write	Off	On
14	PV rate-of-change alarm select	write	Off	On
15	Loop mode is independent from CPU mode when set	write	Loop with CPU mode	Loop Independent of CPU mode

PID Mode Setting 2 Bit Descriptions (Addr + 01)

The individual bit definitions of PID Mode Setting 2(Addr+01) word are listed in the following table. More details are in the section related to each bit in this chapter.

Bit	PID Mode Setting 2 Description	Read/Write	Bit=0	Bit=1
0	PV Input Unipolar/Bipolar Select	write	Unipolar	Bipolar
1	Loop Data Format select	write	12 bit	15 bit
2	Analog Input Filter	write	Off	On
3	SP Input limit enable	write	Disable	Enable
4	Integral Gain (Reset) units select	write	seconds	minutes
5	Auto Tune Method	write	closed loop	open loop
6	Auto Tune Selection	write	PID	PI only Rate = 0
7	Auto Tune Start	write	Auto Tune Done	Force Start
8	PID Scan Clock (internal use)	read	–	–
9–15	Reserved for Future Use	–	–	–

Mode / Alarm Monitoring Word (Addr + 06)

The individual bit definitions of the Mode / Alarm monitoring (Addr+06) word are listed in the following table. More details are in the Basic Loop Operation and Process Alarms section.

Bit	Mode / Alarm Bit Description	Read/Write	Bit=0	Bit=1
0	Manual Mode Indication	read	–	Manual
1	Automatic Mode Indication	read	–	Auto
2	Cascade Mode Indication	read	–	Cascade
3	PV Input LOW–LOW Alarm	read	Off	On
4	PV Input LOW Alarm	read	Off	On
5	PV Input HIGH Alarm	read	Off	On
6	PV Input HIGH–HIGH Alarm	read	Off	On
7	PV Input YELLOW Deviation Alarm	read	Off	On
8	PV Input RED Deviation Alarm	read	Off	On
9	PV Input Rate-of-Change Alarm	read	Off	On
10	Alarm Value Programming Error	read	–	Error
11	Loop Calculation Overflow/Underflow	read	–	Error
12	Loop in AutoTune Indication	read	Off	On
13	AutoTune Error	read	–	Error
14–15	Reserved for Future Use	–	–	–

Ramp / Soak Table Flags (Addr + 33)

The individual bit definitions of the Ramp / Soak Table Flag (Addr+33) word are listed in the following table. Further details are given in the Ramp / Soak Generator section.

Bit	Ramp / Soak Flag Bit Description	Read/Write	Bit=0	Bit=1
0	Start Ramp / Soak Profile	write	–	0→1 Start
1	Hold Ramp / Soak Profile	write	–	0→1 Hold
2	Resume Ramp / soak Profile	write	–	0→1 Resume
3	Jog Ramp / Soak Profile	write	–	0→1 Jog
4	Ramp / Soak Profile Complete	read	–	Complete
5	PV Input Ramp / Soak Deviation	read	Off	On
6	Ramp / Soak Profile in Hold	read	Off	On
7	Reserved	read	–	–
8–15	Current Step in R/S Profile	read	decode as byte (hex)	

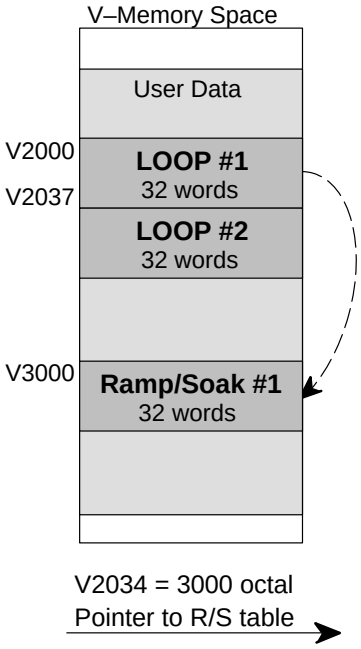
Bits 8–15 must be read as a byte to indicate the current segment number of the Ramp/Soak generator in the profile. This byte will have the values 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and 10. which represent segments 1 to 16 respectively. If the byte=0. then the Ramp/Soak table is not active.

Ramp/Soak Table Location (Addr + 34)

Each loop that you configure has the option of using a built-in Ramp/Soak generator dedicated to that loop. This feature generates SP values in a continuous stream, called a profile. To use the Ramp Soak feature, you must program a separate table of 32 words with appropriate values. A ***Direct***SOFT dialog box makes this easy to do.

In the basic loop table, the Ramp / Soak Table Pointer at Addr+34 must point to the start of the ramp/soak data for that loop. This may be anywhere in user memory, and does not have to be adjoining to the Loop Parameter table, as shown to the left. Each R/S table requires 32 words, regardless of the number of segments programmed.

The ramp/soak table parameters are defined in the table below. Further details are in the section on Ramp / Soak Generator in this chapter.



Addr Offset	Step	Description	Addr Offset	Step	Description
+ 00	1	Ramp End SP Value	+ 20	9	Ramp End SP Value
+ 01	1	Ramp Slope	+ 21	9	Ramp Slope
+ 02	2	Soak Duration	+ 22	10	Soak Duration
+ 03	2	Soak PV Deviation	+ 23	10	Soak PV Deviation
+ 04	3	Ramp End SP Value	+ 24	11	Ramp End SP Value
+ 05	3	Ramp Slope	+ 25	11	Ramp Slope
+ 06	4	Soak Duration	+ 26	12	Soak Duration
+ 07	4	Soak PV Deviation	+ 27	12	Soak PV Deviation
+ 10	5	Ramp End SP Value	+ 30	13	Ramp End SP Value
+ 11	5	Ramp Slope	+ 31	13	Ramp Slope
+ 12	6	Soak Duration	+ 32	14	Soak Duration
+ 13	6	Soak PV Deviation	+ 33	14	Soak PV Deviation
+ 14	7	Ramp End SP Value	+ 34	15	Ramp End SP Value
+ 15	7	Ramp Slope	+ 35	15	Ramp Slope
+ 16	8	Soak Duration	+ 36	16	Soak Duration
+ 17	8	Soak PV Deviation	+ 37	16	Soak PV Deviation

Ramp/Soak Table Programming Error Flags (Addr + 35)

The individual bit definitions of the Ramp / Soak Table programming error flags (Addr+35) word are listed in the following table. Further details are given in the Ramp/Soak Generator section later in this chapter.

Bit	R/S Error Flag Bit Description	Read/ Write	Bit=0	Bit=1
0	Starting Addr out of lower V-memory range	read	–	Error
1	Starting Addr out of upper V-memory range	read	–	Error
2–3	Reserved for Future Use	–	–	–
4	Starting Addr in System Parameter V-memory Range	read	–	Error
5–15	Reserved for Future Use	–	–	–

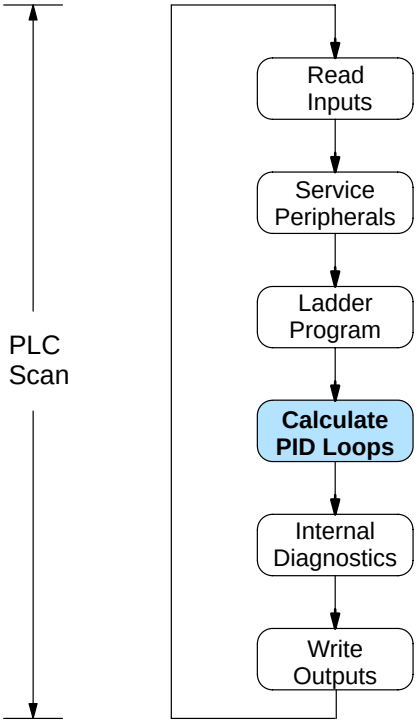
Loop Sample Rate and Scheduling

Loop Sample Rates

The main tasks of the CPU fall into categories as shown to the right. The list represents the tasks done when the CPU is in Run Mode, on each PLC scan. Note that PID loop calculations occur after the ladder logic task. From the user point-of-view, loops can be running when the ladder is not.

The **sample rate** of a control loop is simply the frequency of the PID calculation. Each calculation generates a new control output value. With the DL350 CPU, you can set the sample rate of a loop from 50 mS to 99.99 seconds. So for most loops, the PID calculation will not occur on every PLC scan. In fact, some loops may need calculating only once in 1000 scans.

You select the desired sample rate for each loop, and the CPU automatically schedules and executes PID calculations on the appropriate scans.



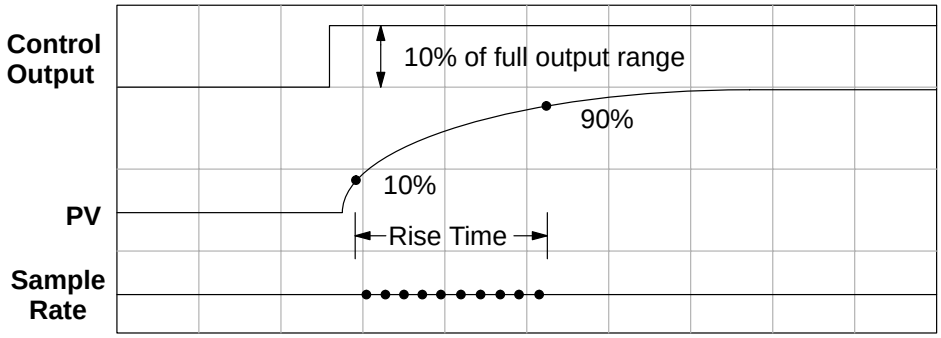
Choosing the Best Sample Rate

For any particular control loop, there is no single perfect sample rate to use. A good sample rate is a compromise that simultaneously satisfies various guidelines:

- The desired sample rate is proportional to the response time of the PV to a change in control output. Usually, a process with a large mass will have a slow sample rate, but a small mass needs a faster sample rate.
- Faster sample rates provide a smoother control output and accurate PV performance, but use more CPU processing time. Sample rates much faster than necessary serve only to waste CPU processing power.
- Slower sample rates provide a rougher control output and less accurate PV performance, but use less CPU processing time.
- A sample rate which is too slow will cause system instability, particularly when a change in the setpoint or a disturbance occurs.

As a starting point, we can determine a sample rate for any particular rate which will be fast enough to avoid control instability (which is extremely important). Do the following procedure to find a starting sample rate:

1. Operate the process open-loop (the loop does not need to be configured). Place the CPU in run mode and the loop in Manual mode. Manually set the control output value so the PV is stable and in the middle of a safe range.
2. Try to choose a time when the process will have negligible external disturbances. Then induce a sudden 10% step change in the control value.
3. Record the rise or fall time of the PV (time between 10% to 90% points).
4. Divide the recorded rise or fall time by 10. This is the initial sample rate you can use to begin tuning your loop.

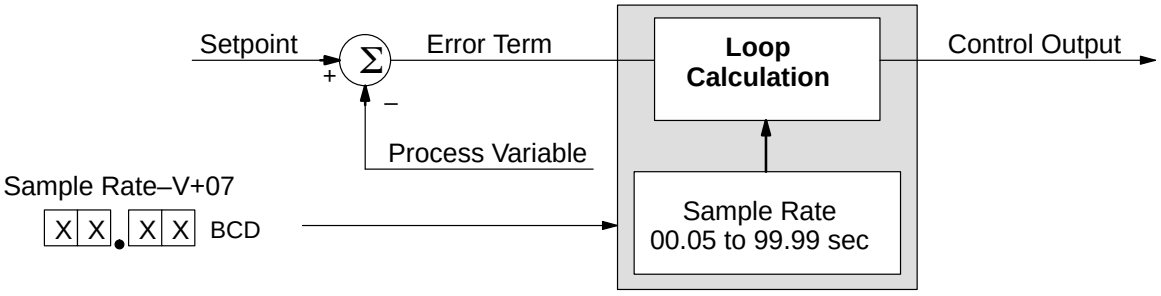


In the figure above, suppose the measured rise time response of the PV was 25 seconds. The suggested sample rate from this measurement will be 2.5 seconds. For illustration, the sample rate time line shows ten samples within the rise time period. These show the frequency of PID calculations as the PV changes values. Of course, the sample rate and PID calculations are continuous during operation.

NOTE: An excessively fast sample rate will diminish the available resolution in the PV Rate-of-Change Alarm, because the alarm rate value is specified in terms of PV change per sample period. For example, a 50 mS sample rate means the smallest PV rate-of-change we can detect is 20 PV counts (least significant bit counts) per second, or 1200 LSB counts per minute.

Programming the Sample Rate

The Loop Parameter table for each loop has data locations for the sample rate. Referring to the figure below, location V+07 contains a BCD number from 00.05 to 99.99 (with an implied decimal point). This represents 50 mS to 99.99 seconds. This number may be programmed using ***Direct***SOFT's PID Setup screen, or any other method of writing to V-memory. It must be programmed before the loop will operate properly.



PID Loop Effect on CPU Scan Time

Since PID loop calculations are a task within the CPU scan activities, the use of PID loops will increase the *average* scan time. The amount of scan time increase is proportional to the number of loops used and the sample rate of each loop.

The execution time for a single loop calculation depends on the number of options selected, such as alarms, error squared, etc. The chart to the right gives the range of times you can expect.

PID Calculation Time

Minimum	150 μS
Typical	250 μS
Maximum	350 μS

To calculate scan time increase, we also must know (or estimate) the scan time of the ladder (without loops), because a fast scan time will increase by a smaller percentage than a slow scan time will, when adding the same PID loop calculation load in each case. The formula for average scan time calculation is:

$$\text{Avg. Scan Time with PID loop} = \left[\frac{\text{Scan time without loop}}{\text{Sample rate of loop}} \times \text{PID calculation time} \right] + \text{Scan time without loop}$$

For example, suppose the estimated scan time without loop calculations is 50 mS, and the loop sample time is 3 seconds. Now, we calculate the new scan time:

$$\text{Average Scan time with PID loop} = \left[\frac{50\ \text{mS}}{3\ \text{sec.}} \times 250\ \mu\text{S} \right] + 50\ \text{mS} = 50.004\ \text{mS}$$

As the calculation shows, the addition of only one loop with a slow sample rate has a very small effect on scan time. Next, we expand equation above to show the effect of adding any number of loops:

$$\text{Avg. Scan Time with PID loops} = \left[\sum_{n=1}^{n=L} \frac{\text{Scan time without loop}}{\text{Sample rate of nth loop}} \times \text{PID calculation time} \right] + \text{Scan time without loops}$$

In the new equation above, we must calculate the summation term (inside the brackets) for each loop from 1 to L (last loop), and add the right-most term “scan time without loops” only once at the end. Suppose we have a DL350 CPU controlling four loops. The table below shows the data and summation term values for each loop.

Loop Number	Description	Sample Rate	Summation Term
1	Steam Flow, Inlet valve	0.25 sec	50 μS
2	Water bath temperature	30 sec	0.42 μS
3	Dye level, main tank	10 sec	1.25 μS
4	Steam Pressure, Autoclave	1.5 sec	8.3 μS

Now adding the summation terms, plus the original scan time value, we have:

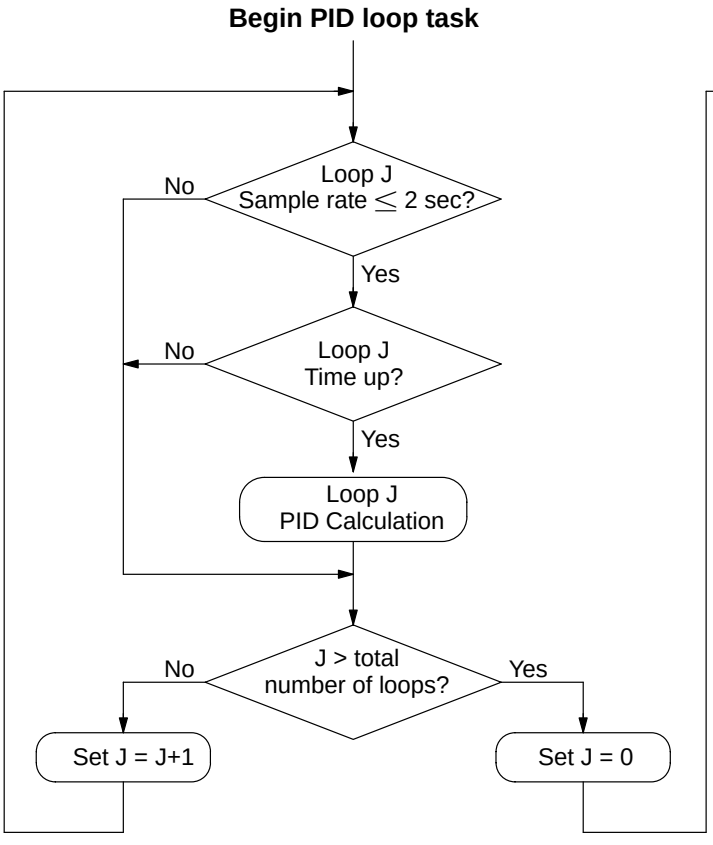
$$\text{Avg. Scan Time with PID loops} = \left[50\ \mu\text{S} + 0.42\ \mu\text{S} + 1.25\ \mu\text{S} + 8.3\ \mu\text{S} \right] + 50\ \text{mS} = 50.06\ \text{mS}$$

The DL350 CPU only does PID calculation on a particular scan for the loop(s) which have sample time periods that are due for an update (calculation). The built-in loop scheduler applies the following rules:

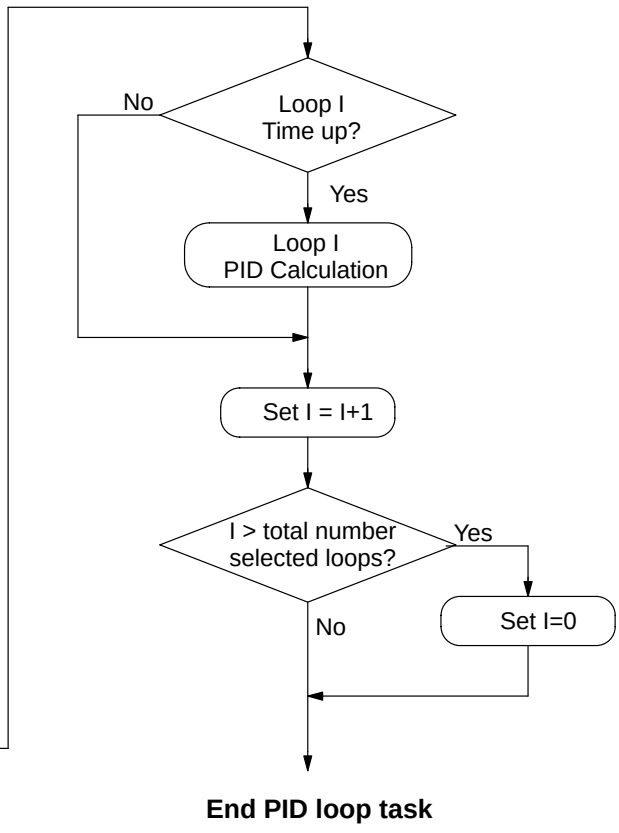
- Loops with sample rates $\leq$ 2 seconds are processed at the rate of as many loops per scan as is required to maintain each loop's sample rate. Specifying loops with fast sample rates will increase the PLC scan time. *So, use this capability only if you need it!*
- Loops with sample rates > 2 seconds are processed at the rate of one or less loops per scan, at the minimum rate required to maintain each loop's sample rate.

The implementation of loop calculation scheduling is shown in the flow chart below. This is a more detailed look at the contents of the "Calculate PID Loops" task in the CPU scan activities flow chart. The pointers "I" and "J" correspond to the slow (> 2 sec) and fast ($\leq$ 2 sec) loops, respectively. The flow chart allows the J pointer to increment from loop 1 to the last loop, if there are any fast loops specified. The I pointer increments only once per scan, and then only when the next slow loop is due for an update. In this way, both I and J pointers cycle from 1 to the highest loop number used, except at different rates. Their combined activity keeps all loops properly updated.

Loop Sample Times $\leq$ 2 seconds:



Loop Sample Times > 2 seconds:



Ten Steps to Successful Process Control

Modern electronic controllers such as the DL350 CPU provide sophisticated process control features. Automated control systems can be very difficult to debug, because a given symptom can have many possible causes. We recommend a careful, step-by-step approach to bringing new control loops online:

Step 1: Know the Recipe

The most important knowledge is – how to make your product. This knowledge is the foundation for designing an effective control system. A good process "recipe" will do the following:

- Identify all relevant Process Variables, such as temperature, pressure, or flow rates, etc. which need precise control.
- Plot the desired Setpoint values for each process variables for the duration of one process cycle.

Step 2: Plan Loop Control Strategy

This simply means choosing the method the machine will use to maintain control over the Process Variable(s) to follow their Setpoints. This involves many issues and trade-offs, such as energy efficiency, equipment costs, ability to service the machine during production, and more. You must also determine how to generate the Setpoint value during the process, and whether a machine operator can change the SP.

Step 3: Size and Scale Loop Components

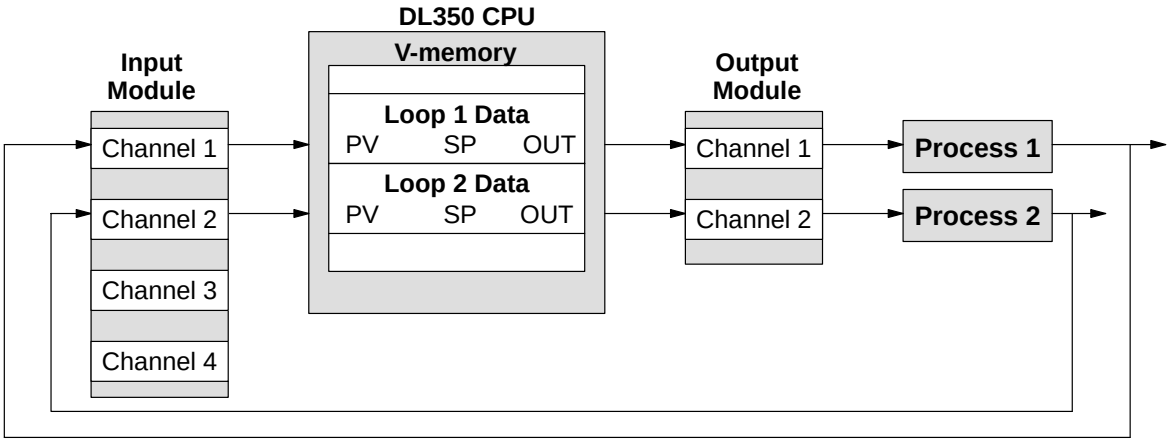
Assuming the control strategy is sound, it is still crucial to *properly size the actuators and properly scale the sensors*.

- Choose an actuator (heater, pump. etc.) which matches the size of the load. An oversized actuator will have an overwhelming effect on your process after a SP change. However, an undersized actuator will allow the PV to lag or drift away from the SP after a SP change or process disturbance.
- Choose a PV sensor which matches the range of interest (and control) for our process. Decide the resolution of control you need for the PV (such as within 2 deg. C), and make sure the sensor input value provides the loop with at least 5 times that resolution (at LSB level). However, an over-sensitive sensor can cause control oscillations, etc. The DL350 provides 12-bit and 15-bit, unipolar and bipolar data format options. This selection affects SP, PV, Control Output, and Integrator sum.

Step 4: Select I/O Modules

After deciding the number of loops, PV variables to measure, and SP values, we can choose the appropriate I/O modules. Refer to the figure on the next page. In many cases, you will be able to share input or output modules among several control loops. The example shown sends the PV and Control Output signals for two loops through the same set of modules.

Remember that PLC***Direct*** offers DL305 analog modules with 2, 4, 8, and 16 channels per module in different signal types and ranges. Refer to the sales catalog for further information on specific modules. The analog modules have their own manual, which will be essential during most installations.



Step 5: Wiring and Installation

After selection and procurement of all loop components and I/O modules, we can perform the wiring and installation. Refer to the wiring guidelines in Chapter 2 of this Manual, and to the DL305 Analog I/O Module manual as needed. The most commonly overlooked wiring details in installing PID loop controls are:

- It's easy to reverse the polarity of connection on sensor wiring.
- Pay attention to signal ground connections between loop components.

Step 6: Loop Parameters

After wiring and installation, we can choose the loop setup parameters. The easiest method for programming the loop tables is ***Direct***SOFT (2.2 or later), using the PID Setup dialog boxes. Be sure to study the meaning of all loop parameters in this chapter before choosing values to enter.

Step 7: Check Open Loop Performance

With the sensors and actuator wiring done, and loop parameters entered, we must manually and carefully check out the new control system (use Manual Mode).

- Verify the PV value from the sensor is correct.
- If it is safe to do so, gradually increase the control output up above 0%, and see if the PV responds (*and moves in the correct direction!*).

Step 8: Loop Tuning

If the open loop test shows the PV reading is good and the control output has the proper effect on the process, we can do the closed loop tuning procedure (Automatic Mode). In this most crucial step, we tune the loop so the PV automatically follows the SP. Refer to the section on Loop Tuning in this chapter.

Step 9: Run Process Cycle

If the closed loop test shows PV will follow small changes in the SP, we can consider running an actual process cycle. Now we must do the programming to generate the desired SP in real time. In this step, you may run a small test batch of product through the machine, while the SP changes according to the recipe.

WARNING: Be sure the Emergency Stop and power-down provision is readily accessible, in case the process goes out of control. Damage to equipment and/or serious injury to personnel can result from loss of control of some processes.

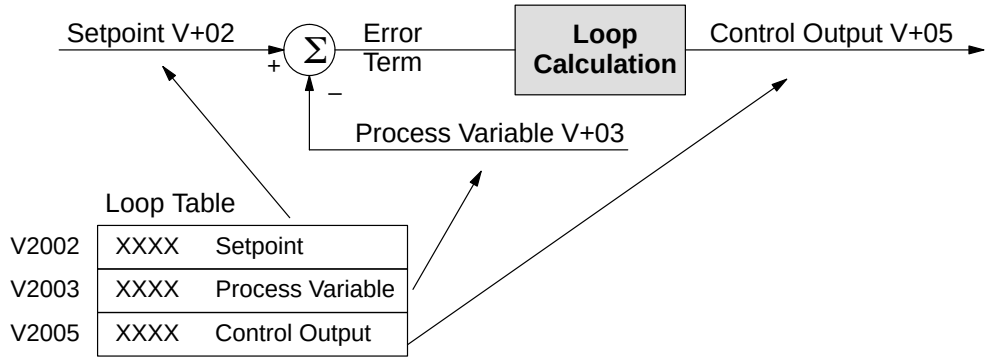
Step 10: Save Loop Parameters

When the loop tests and tuning sessions are complete, be sure to save all loop setup parameters to disk. Loop parameters represent a lot of work in loop tuning, and are well worth saving.

Basic Loop Operation

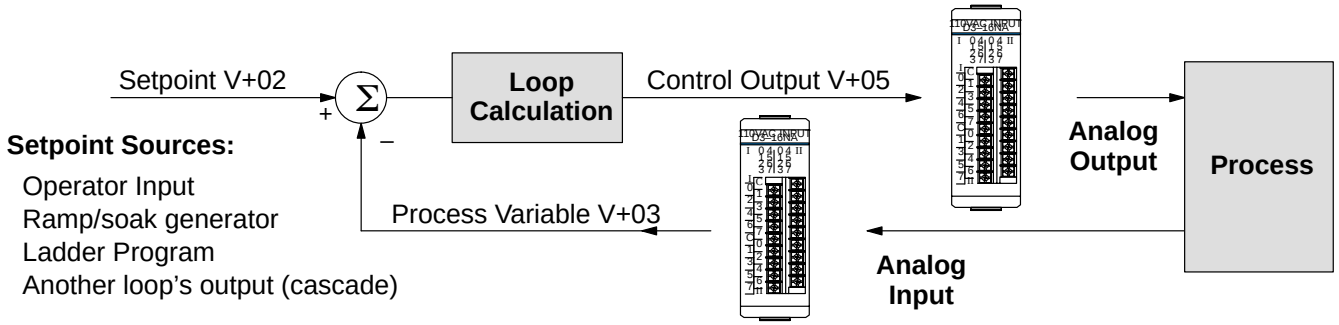
Data Locations

Each PID loop is completely dependent on the instructions and data values in its respective loop table. The following diagram shows the loop table locations corresponding to the main three loop I/O variables: SP, PV, and Control Output. The example loop table below begins at V2000 (an arbitrary location to be chosen by the user). The SP, PV and Control Output are located at the addresses shown.



Data Sources

The data for the SP, PV, and Control Output must interface with real-word sources and devices. In the figure below, the sources or destinations are shown for each loop variable. The Control Output and Process Variable values move through the appropriate analog module to interface with the process itself. Remember that most analog modules have multiplexed data, with two or three channel address decode bits. Refer to the analog module manual for ladder examples that show how to move analog data between DL305 analog modules and an arbitrary V-memory location.



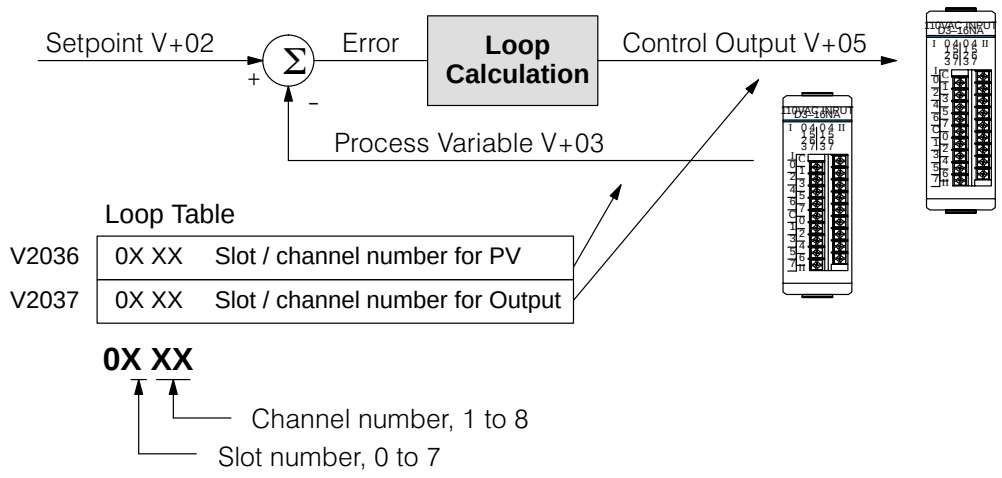
The Setpoint has several possible sources, listed in the figure above. Many applications will use two or more of the sources at various times, depending on the loop mode. In addition, the loop control topology and programming method also determine how the setpoint is generated. When using the built-in Ramp/Soak generator or when cascading a loop, the PID controller automatically writes the setpoint data in location V+02 for you.

Obviously, each of the three main loop parameters will have only one source or destination at any given time. During the application development, it's a good idea to draw loop schematic diagrams showing data sources, etc. to help avoid mistakes.

Direct Access to Analog I/O

The loop controller in the DL350 CPU has ability to directly access analog I/O values apart from the ladder logic scan. In particular, these parameters are the process variable (PV) and the control output. This feature is naturally required because the loop controller can perform closed-loop control while the CPU is in Program Mode. The loop controller can read the analog PV value in the selected data format from the desired analog module, and write the control output value in the same data format to the desired output module. This direct-access feature, when enabled, accesses the analog values only once per PID calculation for each respective loop. The ladder logic, however, may simultaneously access the same analog I/O by the standard method (LD instruction), or by the pointer method whenever the CPU is in Run Mode.

You may optionally configure each loop to access its analog I/O (PV and control output) by placing proper values in the associated loop table registers. The following figure shows the loop table parameters at V+36 and V+37 and their role in direct access to the analog values.



You may program these loop table parameters directly, or use the appropriate ***Direct***SOFT dialog box for easy configuring. For example, a value of “0102” in register V2036 directs the loop controller to read the PV data from slot number 1, and the second channel. Note that slot 1 is the *second* slot to the left of the CPU, because slot 0 is adjacent to the CPU. A value of “0000” in either register tells the loop controller *not* to access the corresponding analog value directly. In that case, ladder logic must transfer the value between the loop table and the physical I/O module.

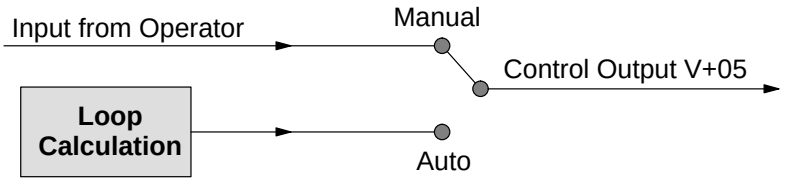
If the PV or control output values require some math manipulation by ladder logic, then it will not be possible to use the direct access function of the loop controller. In this case, ladder logic will have to perform the math and transfer to data to or from the analog modules as required.

NOTE: If desired, it is possible to use direct access to analog modules for some variables and use ladder logic to transfer other variable data to or from the analog modules. However, the loop controller firmware restricts us from transfering analog data from two or more channels within an analog module using different methods. You must designate each analog module for direct access or ladder logic access.

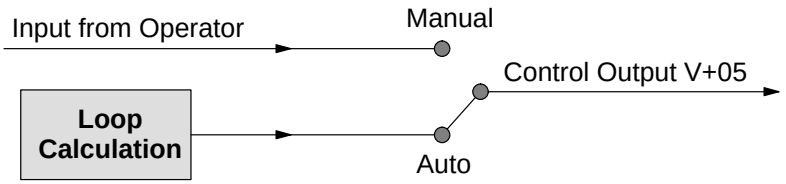
Loop Modes

In PID Loop applications, we have control situations that frequently occur throughout the industry. In each scenario, we slightly modify the source of data for the basic three variables SP, PV, and control output, creating a mode name for each scenario. The modes featured in the DL350 CPU are *Manual*, *Automatic*, and *Cascade*. After this introduction to the modes, we will study how to request mode changes.

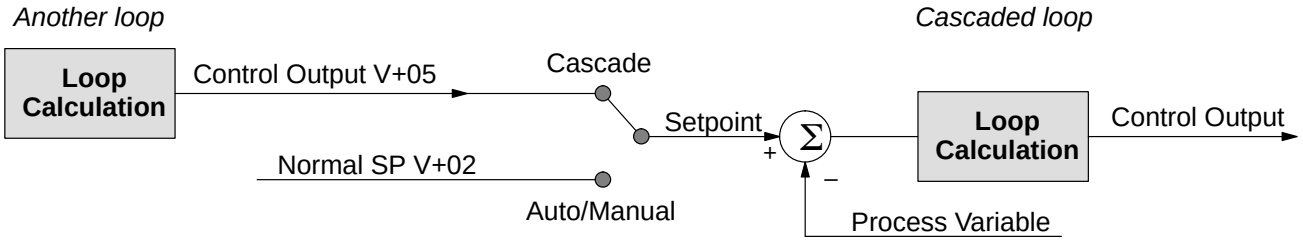
In **Manual Mode**, the loop is not executing PID calculations (however, loop alarms are still active). With regard to the loop table, the CPU stops writing values to location V+05 for that loop. It is expected that an operator or other intelligent source is manually controlling the output, by observing the PV and writing data to V+05 as necessary to keep the process under control. The drawing below shows the equivalent schematic diagram of manual mode operation.



In **Automatic Mode**, the loop operates normally and generates new control output values. It calculates the PID equation and writes the result in location V+05 every sample period of that loop. The equivalent schematic diagram is shown below.



In **Cascade Mode**, the loop operates like in Automatic Mode, with one important change. The data source for the SP changes from its normal location at V+02, using the control output value from another loop (the purpose of cascading loops is covered later in this chapter). So in Auto or Manual modes, the loop calculation uses the data at V+02. In Cascade Mode, the loop calculation reads the control output from another loop's parameter table.



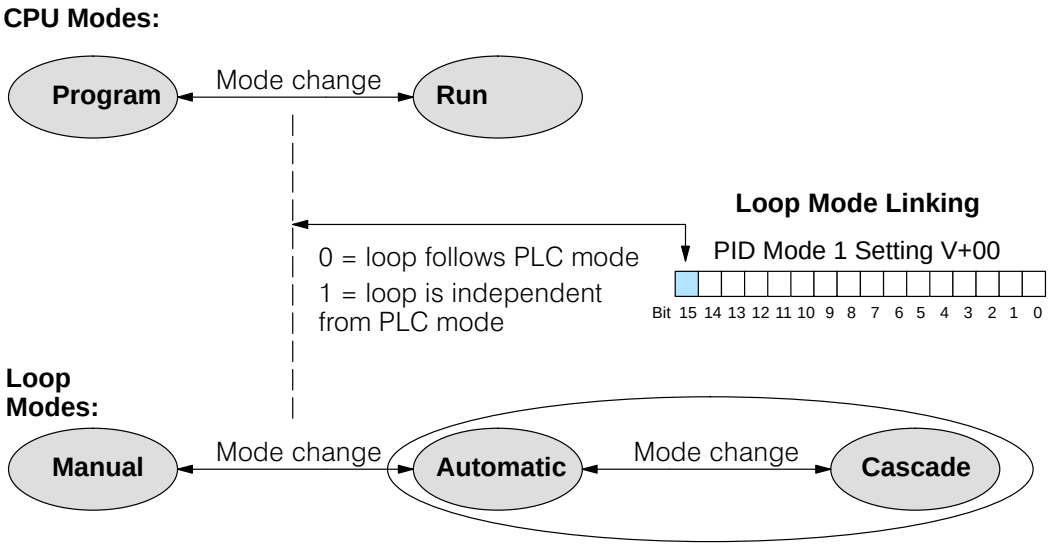
Realizing the way PID calculations change data sources according to the Manual/Auto/Cascade modes, naturally some restrictions on mode changes exist. As pictured below, a loop change from one mode to another, but *cannot go from Manual Mode to Cascade*. This mode change is prohibited because a loop would be changing two data sources at the same time, and could cause a loss of control.

CPU Modes and Loop Modes

One powerful aspect of the loop controller on the DL350 CPU is its ability to run PID calculations while the CPU is in Program Mode. It is usually true that a CPU in Program Mode has halted all operations. However, a DL350 CPU in Program Mode may or may not be running PID calculations, depending on your configuration settings. Having the ability to run loops independently of the ladder logic makes it feasible to make a ladder logic change while the process is still running. This is especially beneficial for large-mass continuous processes that are difficult or costly to interrupt.

Of course, loops that run independent of the ladder scan must have the ability to directly access the analog module channels for the PV and control output values. The loop controller does have this capability, which is covered in the section on direct access to analog I/O (located prior to this section in this chapter).

The relationship between CPU modes and loop modes is depicted in the figure below. The vertical dashed line shows the optional relationship between the mode changes. Bit 15 of PID Mode 1 setting word V+00 determines the selection. If set to zero so the loop follows the CPU mode, then placing the CPU in Program Mode will force all loops into Manual Mode. Similarly, placing the CPU in Run mode will allow each loop to return to the mode it was in previously (which includes Manual, Automatic, and Cascade). With this selection you automatically affect the modes of the loops by changing the CPU mode.

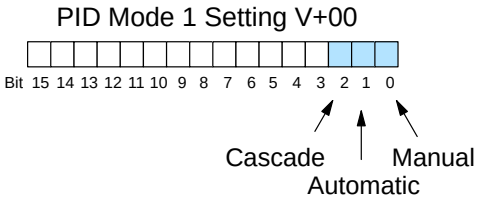


If Bit 15 is set to one, then the loops will run independently of the CPU mode. It is like having two independent processors in the CPU... one is running ladders and the other is running the process loops.

NOTE: If you choose for the loops to be operate independently of the CPU mode, then you must take special steps in order to change any loop table parameter values. The procedure is to temporarily make the loops follow the CPU mode. Then your programming device (such as ***Direct***SOFT) will be able to place the loop you want to change into Manual Mode. After you change the loop's parameter setting, be sure to restore the loop independent operation setting.

How to Change Loop Modes

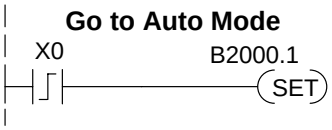
The first three bits of the PID Mode 1 word V+00 requests the operating mode of the corresponding loop. Note: these bits are mode change *requests*, not commands (certain conditions can prohibit a particular mode change – see next page).



The normal state of these mode request bits is “000”. To request a mode change, you must SET the corresponding bit to a “1”, for one scan. The PID loop controller automatically resets the bits back to “000” after it reads the mode change request. Methods of requesting mode changes are:

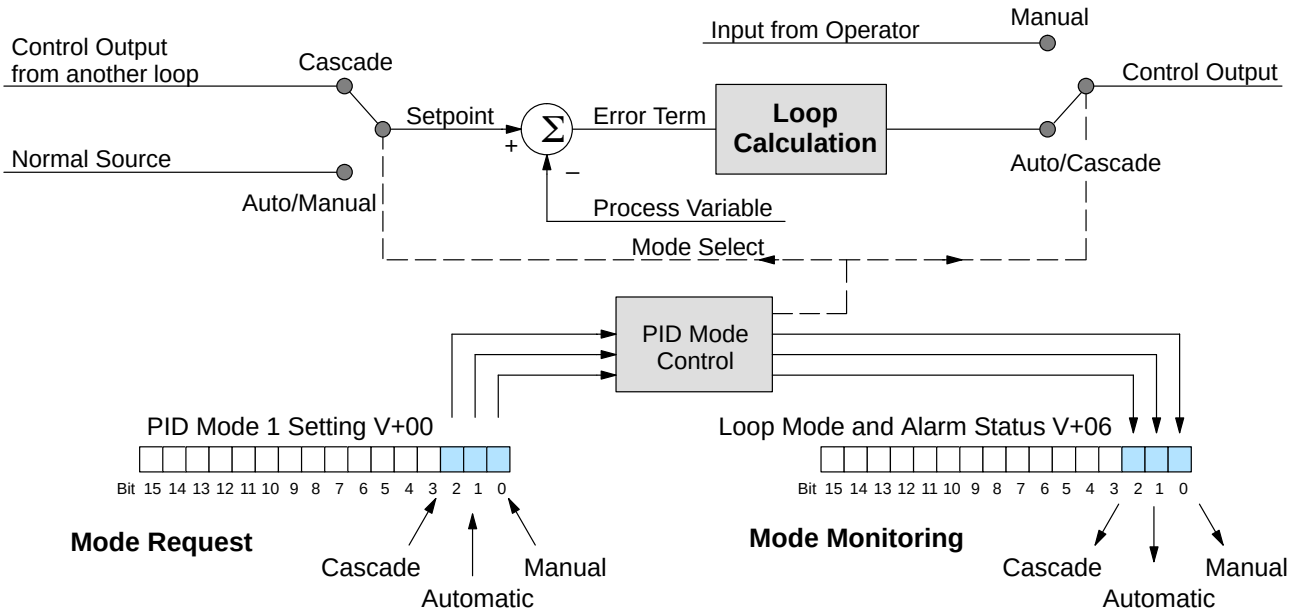
- ***Direct***SOFT’s **PID View** – this is the easiest method. Click on one of the radio buttons, and ***Direct***SOFT sets the appropriate bit.
- **HPP** – Use Word Status (WD ST) to monitor the contents of V+00, which will be a 4-digit BCD/hex value. You must calculate and enter a new value for V+00 that ORs the correct mode bit with its current value.
- **Ladder program**– ladder logic can request any loop mode when the PLC is in Run Mode. This will be necessary after application startup.

  Use the program shown to the right to SET the mode bit on (do not use an out coil). On a 0–1 transition of X0, the rung sets the Auto bit = 1. The loop controller resets it.



- **Operator panel** – interface the operator’s panel to ladder logic using standard methods, then use the technique above to set the mode bit.

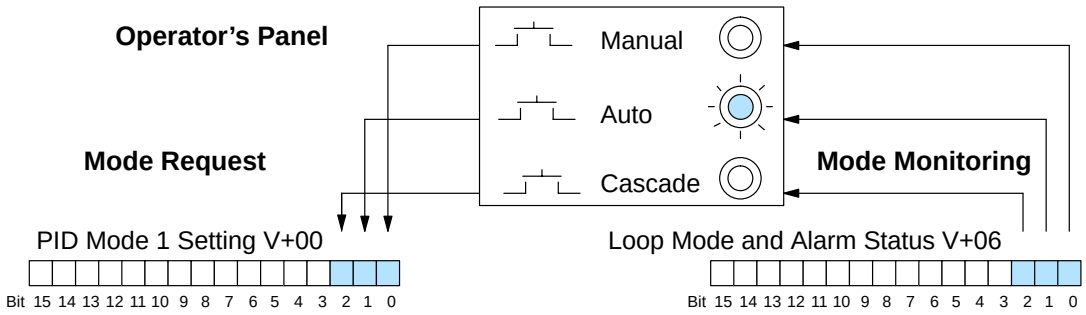
Since we can only *request* mode changes, the PID loop controller decides when to permit mode changes and provides the loop mode status. It reports the current mode on bits 0, 1, and 2 of the Loop Mode and Alarm Status word, location V+06 in the loop table. The parallel request / monitoring functions are shown in the figure below. The figure also shows the mode-dependent two possible SP sources, and the two possible Control Output sources.



Operator Panel Control of PID Modes

Since the modes Manual, Auto, and Cascade are the most fundamental and important PID loop controls, you may want to “hard-wire” mode control switches to an operator’s panel. Most applications will need only Manual and Auto selections (Cascade is used in a few advanced applications). Remember that mode controls are really *mode request* bits, and the actual loop mode is indicated elsewhere.

The following figure shows an operator’s panel using momentary push-buttons to request PID mode changes. The panel’s mode indicators do not connect to the switches, but interface to the corresponding data locations.



PLC Modes’ Effect on Loop Modes

If you have selected the option for the loops to follow the PLC mode, the PLC modes (Program, Run) interact with the loops as a group. The following summarizes this interaction:

- When the PLC is in Program Mode, all loops are placed in Manual Mode and no loop calculations occur. However, note that output modules (including analog outputs) turn off in PLC Program Mode. So, actual manual control is not possible when the PLC is in Program Mode.
- The only time the CPU will allow a loop mode change is during PLC run Mode operation. As such, the CPU records the modes of all 4 loops as the desired mode of operation. If power failure and restoration occurs during PLC Run Mode, the CPU returns all loops to their prior mode (which could be Manual, Auto, or Cascade).
- On a Program-to-Run mode transition, the CPU forces each loop to return to its prior mode recorded during the last PLC Run Mode.
- You can add and configure new loops only when the PLC is in Program Mode. New loops automatically begin in Manual Mode.

Loop Mode Override

In normal conditions the mode of a loop is determined by the request to V+00, bits 0, 1, and 2. However, some conditions exist which will prevent a requested mode change from occurring:

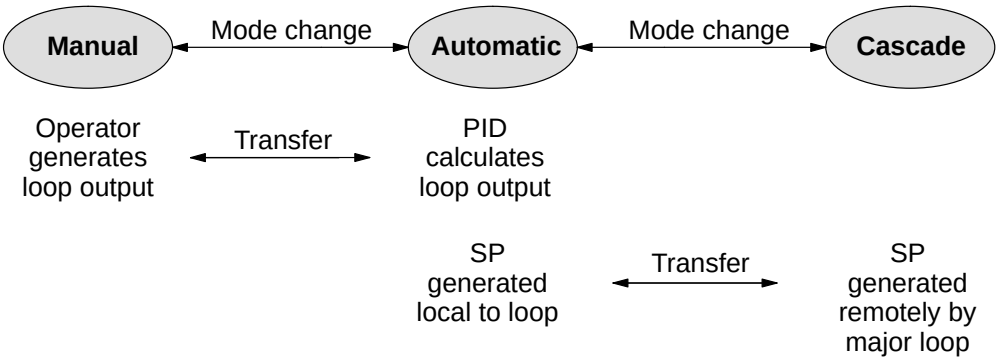
- A loop that is not set independent of PLC mode cannot change modes when the PLC is in Program mode.
- A major loop of a cascaded pair of loops cannot go from Manual to Auto until its minor loop is in Cascade mode.

In other situations, the PID loop controller will automatically change the mode of the loop to ensure safe operation:

- A loop which develops an error condition automatically goes to Manual.
- If the minor loop of a cascaded pair of loops leaves Cascade Mode for any reason, its major loop automatically goes to Manual Mode.

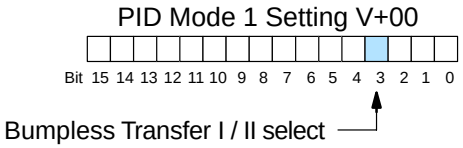
Bumpless Transfers

In process control, the word “transfer” has a particular meaning. A loop transfer occurs when we change its mode of operation, as shown below. When we change loop modes, what we are really doing is causing a transfer of control of some loop parameter from one source to another. For example, when a loop changes from Manual Mode to Automatic Mode, control of the output changes from the operator to the loop controller. When a loop changes from Automatic Mode to Cascade Mode, control of the SP changes from its original source in Auto Mode to the output of another loop (the major loop).



The basic problem of loop transfers is the two different sources of the loop parameter being transferred will have different numerical values. This causes the PID calculation to generate an undesirable step change, or “bump” on the control output, thereby upsetting the loop to some degree. The “bumpless transfer” feature arbitrarily forces one parameter equal to another at the moment of loop mode change, so the transfer is smooth (no bump on the control output).

The bumpless transfer feature of the DL350 loop controller is available in two types: Bumpless I, and Bumpless II. Use ***Direct***SOFT’s PID Setup dialog box to select transfer type. Or, you can use bit 3 of PID Mode 1 V+00 setting as shown.



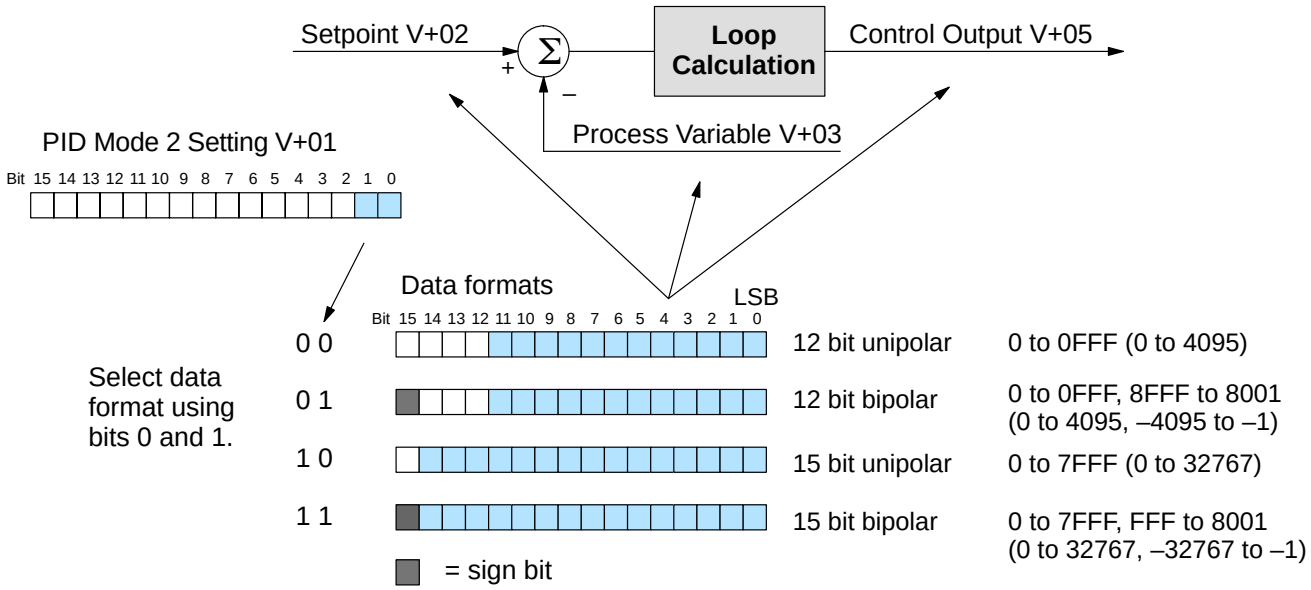
The characteristics of Bumpless I and II transfer types are listed in the chart below. Note that their operation also depends on which PID algorithm you are using, the position or velocity form of the PID equation. Note that you must use Bumpless Transfer type I when using the velocity form of the PID algorithm.

Transfer Type	Transfer Select Bit	PID Algorithm	Manual-to-Auto Transfer Action	Auto-to-Cascade Transfer Action
Bumpless Transfer I	0	Position	Forces Bias = Control Output Forces SP = PV	Forces Major Loop Output = Minor Loop PV
		Velocity	Forces SP = PV	Forces Major Loop Output = Minor Loop PV
Bumpless Transfer II	1	Position	Forces Bias = Control Output	none
		Velocity	none	none

PID Loop Data Configuration

Loop Parameter Data Formats

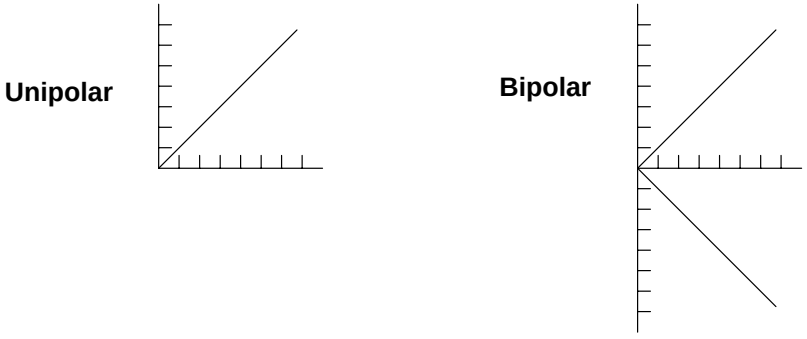
In choosing the Process Variable range and resolution, a related choice to make is the data format of the three main loop variables: SP, PV, and Control Output (the Integrator sum in V+04 also uses this data format). The four data formats available are 12 or 15 bit (right justified), signed or unsigned (MSB is sign bit in bipolar formats). The four binary combinations of bits 0 and 1 of PID Mode 2 word V+01 choose the format. The ***Direct***SOFT PID Setup dialog sets these bits automatically when you select the data format from the menu.



The data format is a very powerful setting, because it determines the numerical interface between the PID loop and the PV sensor, and the Control Output device. The Setpoint must also be in the same data format. Normally, the data format is chosen during the initial loop configuration and is not changed again.

Choosing Unipolar or Bipolar Format

Choosing the data format involves deciding whether to use unipolar or bipolar numbers. Most applications such as temperature control will use only positive numbers, and therefore need unipolar format. Usually it is the Control Output which determines bipolar/unipolar selection. For example, velocity control may include control of forward and reverse directions. At a zero velocity setpoint the desired control output is also zero. In that case, bipolar format must be used.



Handling Data Offsets

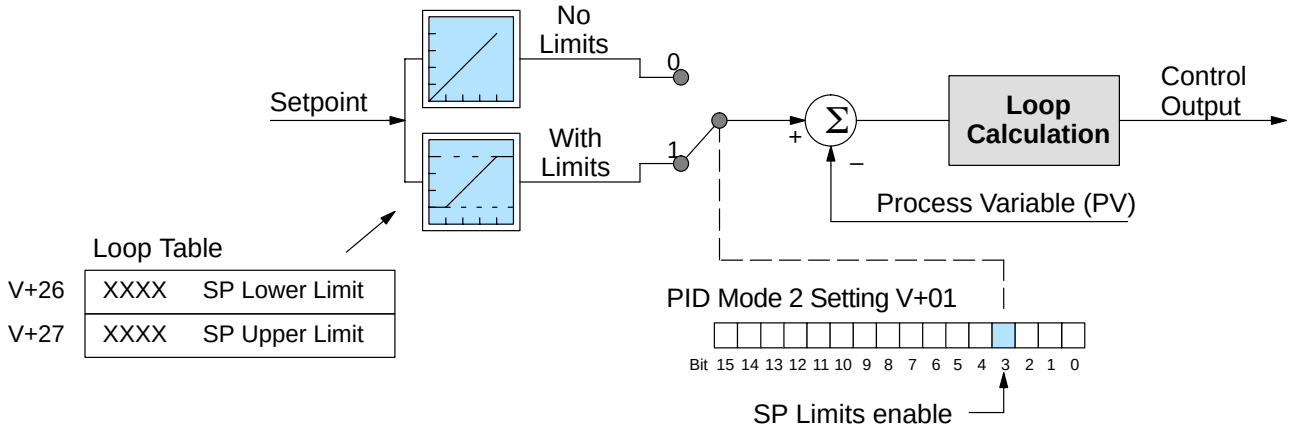
In many batch process applications, sensors or actuators interface to DL305 analog modules using 4–20 mA signals. This signal type has a built-in 20% offset, because the zero-point is a 4 mA instead of 0 mA. However, remember the analog modules convert the signals into data *and remove the offset at the same time*. For example, a 4–20 mA signal is often converted to 0000 – 0FFF hex, or 0 to 4095 decimal. In this case, all you need to do is choose 12-bit unipolar data format, and make sure the ladder program copies the data appropriately between the loop table and the analog modules.

- **PV Offset** – In the event you have a PV value with a 20% offset, convert it to zero–offset by subtracting 20% of the top of its range, and multiply by1.25.
- **Control Output** – In the event the Control Output is going to a device with 20% offset, all you need to do is have the ladder program write a value equivalent to the offset to the integrator register (V+04), before transitioning from Manual to Auto mode. The loop will then see this offset as a part of the process, taking care of it for you automatically.

Setpoint (SP) Limits

The Setpoint in loop table location V+02 represents the desired value of the process variable. After selecting the data format for these variables, you can set limits on the range of SP values which the loop calculation will use. Many loops have two or more possible sources writing the Setpoint at various times, and the limits you set will help safeguard the process from the effects of a bad SP value.

In the figure below, the SP has a selectable limit function, enabled by PID Mode 2 Setting V+01 word, bit 3. If enabled, then locations V+26 and V+27 determine the lower and upper SP limits, respectively. The loop calculation applies this limit internally, so it is always possible to write any value to V+02.

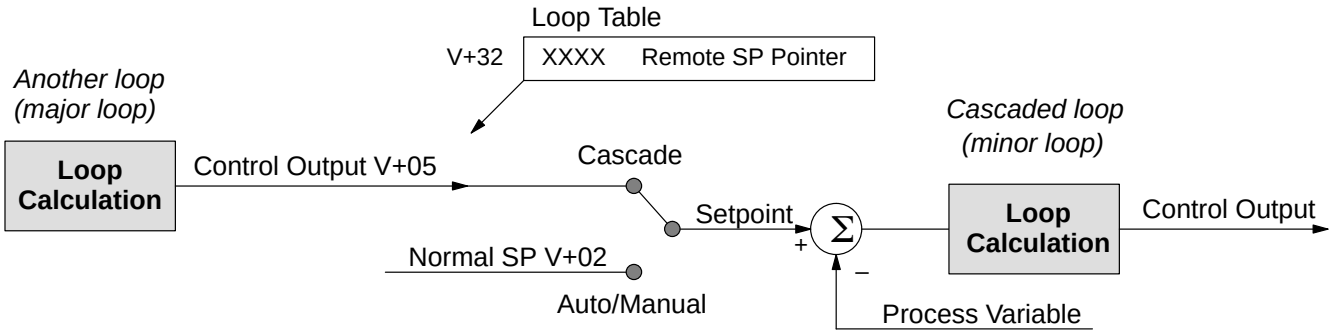


The loop calculation checks these SP upper and lower limits before each calculation. This means ladder logic can change the limit settings while a process is in progress, allowing you to keep a tighter guard band on the SP input value.

Remote Setpoint (SP) Location

You may recall there are generally several possible data sources for the SP value. The PID loop controller has the built-in ability to select between two sources according to the current loop mode. Refer to the figure below. A loop reads its setpoint from table location V+02 in Auto or Manual modes. If you plan to use Cascade Mode for the loop at any time, then you must program its loop parameter table with a *remote setpoint pointer.*

The Remote SP pointer resides in location V+32 in the loop table. For loops that will be cascaded (made a minor loop), you will need to program this location with the address of the major loop's Control Output address. Find the starting location of the major loop's parameter table and add offset +05 to it.

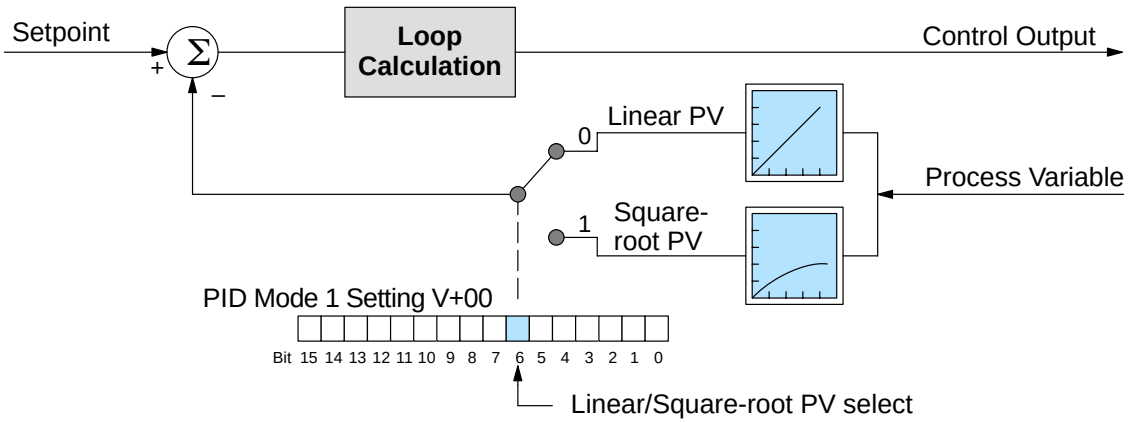


A ***Direct***SOFT Loop Setup dialog box will allow you to enter the Remote SP pointer if you know the address. Otherwise, you can enter it with a HPP or program it through ladder logic using the LDA instruction.

Process Variable (PV) Configuration

The process variable input to each loop is the value the loop is ultimately trying to control, to make it equal to the setpoint and follow setpoint changes as quickly as possible. Most sensors for process variables have a primarily linear response curve. Most temperature sensors are mostly linear across their sensing range. However, flow sensing using an orifice plate technique gives a signal representing (approximately) the square of the flow. Therefore, a square-root extract function is necessary before using the signal in a linear control system (such as PID).

Some flow transducers are available which will do the square-root extract, but they add cost to the sensor package. The PID loop PV input has a selectable square-root extract function, pictured below. You can select between normal (linear) PV data, and data needing a square-root extract by using PID Mode setting V+00 word, bit 6.



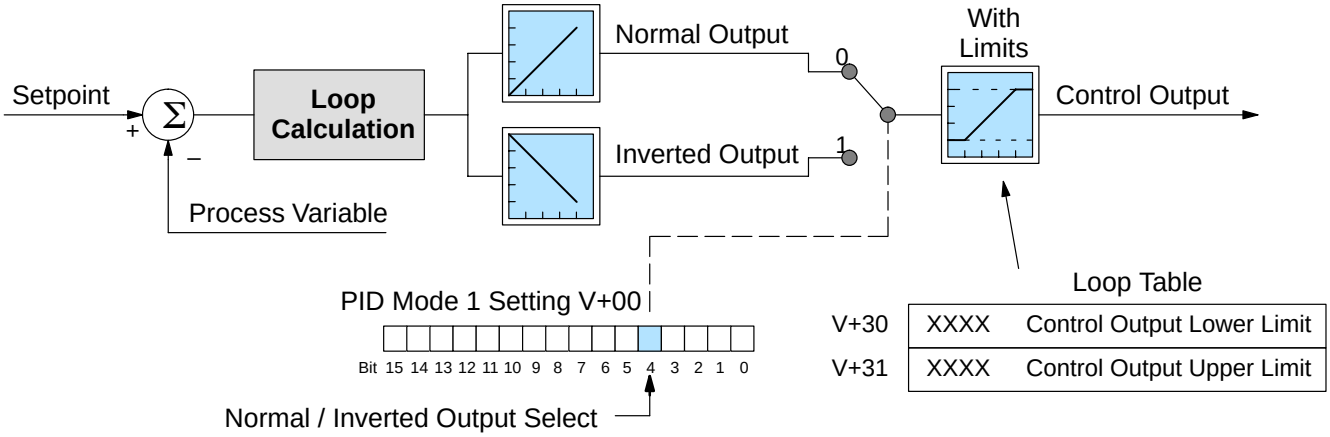
IMPORTANT: The scaling of the SP must be adjusted if you use PV square-root extract, because the loop drives the output so the *square root* of the PV is equal to the PV input. Divide the desired SP value by the square root of the analog span, and use the result in the V+02 location for the SP. This does reduce the resolution of the SP, but most flow control loops do not require a lot of precision (the recipient of the flow is integrating the errors). Use one of the following formulas for the SP according to the data format you are using. It's a good idea to set the SP upper limit to the top of the allowed range.

Data Format	SP Scaling	SP Range	PV range
12-bit	SP = PV input / 64	0 – 64	0 – 4095
15-bit	SP = PV input / 181.02	0 – 181	0 – 32767

Control Output Configuration

The Control Output is the numerical result of the PID calculation. All of the other parameter choices ultimately influence the value of a loop's Control Output for each calculation. Some final processing selections dedicated to the Control Output are available, shown below. At the far right of the figure, the final output may be restricted by lower and upper limits that you program. The values for V+30 and V+31 may be set once using ***Direct***SOFT's PID Setup dialog box. Or, you may use the ladder program to write to the table locations and change the limits while the loop is running.

The Control Output lower and upper limits can help guard against commanding an excessive correction to an error when a loop fault occurs (such as PV sensor signal loss). However, do not use these limits to restrict mechanical motion that might otherwise damage a machine (use hard-wired limit switches instead).



The other available selection is the normal/inverted output selection (called "forward/reverse" in ***Direct***SOFT). Use bit 4 of the PID Mode 1 Setting V+00 word to configure the output. Independently of unipolar or bipolar format, a normal output goes upward on positive errors and downward on negative errors (where Error=(SP–PV)). The inverted output reverses the direction of the output change.

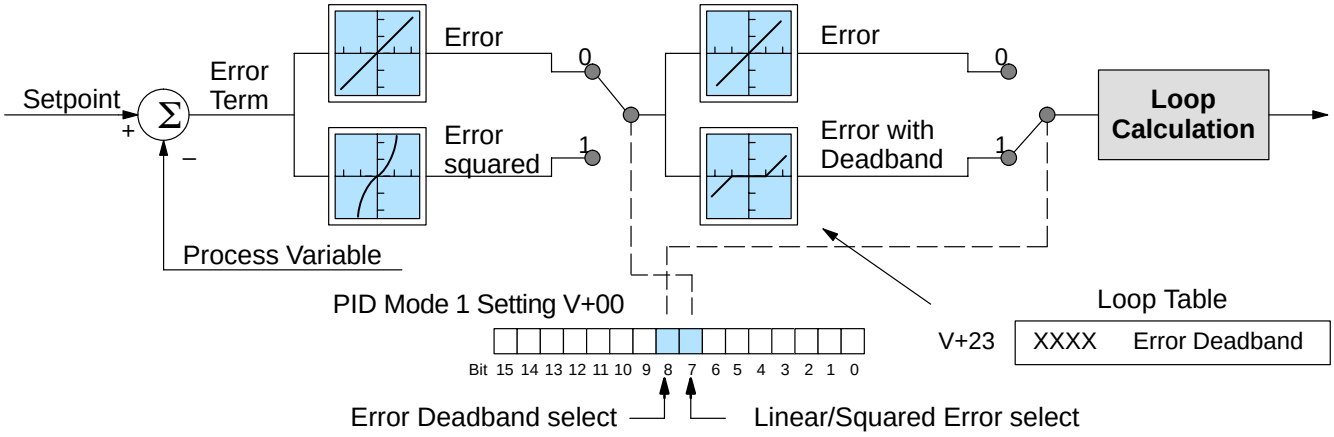
The normal/inverted output selection is used to configure direct-acting/reverse-acting loops. This selection is ultimately determined by the direction of the response of the process variable to a change in the control output in a particular direction. Refer to the PID Algorithms section for more on direct-acting and reverse-acting loops.

Error Term Configuration

The Error term is internal to the CPUs PID loop controller, and is generated again in each PID calculation. Although its data is not directly accessible, you can easily calculate it by subtracting: Error = (SP–PV). If the PV square-root extract is enabled, then Error = (SP – (sqrt(PV)). In any case, the size of the error and algebraic sign determine the next change of the control output for each PID calculation.

Now we will superimpose some “special effects” on to the error term as described. Refer to the diagram below. Bit 7 of the PID Mode Setting 1 V+00 word lets you select a linear or squared error term, and bit 8 enables or disables the error deadband.

NOTE: When first configuring a loop, it’s best to use the standard error term. After the loop is tuned, then you will be able to tell if these functions will enhance control.



Error Squared – When selected, the squared error function simply squares the error term (but preserves the original algebraic sign), which is used in the calculation. This affects the Control Output by diminishing its response to smaller error values, but maintaining its response to larger errors. Some situations in which the error squared term might be useful:

- Noisy PV signal – using a squared error term can reduce the effect of low-frequency electrical noise on the PV, which will make the control system jittery. A squared error maintains the response to larger errors.
- Non-linear process – some processes (such as chemical pH control) require non-linear controllers for best results. Another application is surge tank control, where the Control Output signal must be smooth.

Error Deadband – When selected, the error deadband function takes a range of small error values near zero, and simply substitutes zero as the value of the error. If the error is larger than the deadband range, then the error value is used normally.

Loop parameter location V+23 must be programmed with a desired deadband amount. Units are the same as the SP and PV units (0 to FFF in 12-bit mode, and 0 to 7FFF in 15-bit mode). The PID loop controller automatically applies the deadband symmetrically about the zero-error point.

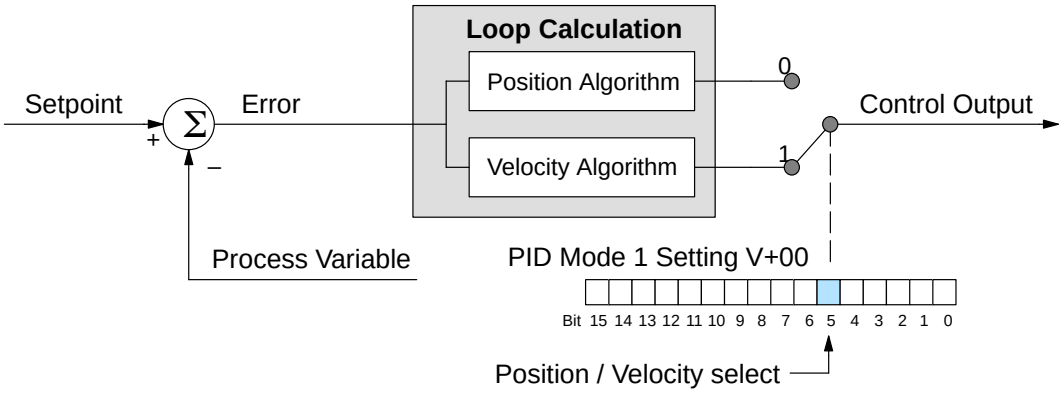
PID Algorithms

The Proportional–Integral–Derivative (PID) algorithm is widely used in process control. The PID method of control adapts well to electronic solutions, whether implemented in analog or digital (CPU) components. The DL350 CPU implements the PID equations digitally by solving the basic equations in software. I/O modules serve only to convert electronic signals into digital form (or vise-versa).

The DL350 features two types of PID controls: “position” and “velocity”. These terms usually refer to motion control situations, but here we use them in a different sense:

- PID *Position* Algorithm – The control output is calculated so it responds to the displacement (position) of the PV from the SP (error term).
- PID *Velocity* Algorithm – The control output is calculated to represent the rate of change (velocity) for the PV to become equal to the SP.

The vast majority of applications will use the position form of the PID equation. If you are not sure of which algorithm to use, try the Position Algorithm first. Use ***Direct***SOFT’s PID View Setup dialog box to select the desired algorithm. Or, use bit 5 of PID Mode 1 Setting V+00 word as shown below to select the desired algorithm.



NOTE: The selection of a PID algorithm is very fundamental to control loop operation, and is normally never changed after the initial configuration of a loop.

Position Algorithm

The Position Algorithm causes the PID equation to calculate the Control Output M_n:

$$M_n = K_c * e_n + K_i * \sum_{i=1}^{n} e_i + K_r * (e_n - e_{n-1}) + M_o$$

In the formula above, the sum of the integral terms and the initial output are combined into the “Bias” term, Mx. Using the bias term, we define formulas for the Bias and Control Output as a function of sampling time:

$Mx_o = M_o$

$Mx_n = K_i * e_n + Mx_{n-1}$

$M_n = K_i * \sum_{i=1}^{n} e_i + M_o$

$M_n = K_c * e_n + K_r * (e_n - e_{n-1}) + Mx_n$.....Output for sampling time “n”

The position algorithm variables and related variables are:

T_s = Sample rate
K_c = Proportional gain
$K_i = K_c * (T_s/T_i)$ coefficient of integral term
$K_r = K_c * (T_d/T_s)$ coefficient of derivative term
T_i = Reset time (integral time)
T_d = Rate time (derivative time)
SP_n = Set Point for sampling time "n" (SP value)
PV_n = Process variable for sampling time "n" (PV)
$e_n = SP_n – PV_n$ = Error term for sampling time "n"
M_0 = Control Output for sampling time "0"
M_n = Control Output for sampling time "n"

Analysis of these equations will be found in most good text books on process control. At a glance, we can isolate the parts of the PID Position Algorithm which correspond to the P, I, and D terms, and the Bias as shown below.

$$M_n = K_c * e_n + K_i * \sum_{i=1}^{n} e_i + K_r * (e_n – e_{n–1}) + M_o$$

Control Output — Proportional Term — Integral Term — Derivative Term — Initial Output — Bias Term

The initial output is the output value assumed from Manual mode control when the loop transitioned to Auto Mode. The sum of the initial output and the integral term is the bias term, which holds the "position" of the output. Accordingly, the Velocity Algorithm discussed next does not have a bias component.

Velocity Algorithm

The Velocity Algorithm form of the PID equation can be obtained by transforming Position Algorithm formula with subtraction of the equation of (n–1)th degree from the equation of nth degree.

The velocity algorithm variables and related variables are:

T_s = Sample rate
K_c = Proportional gain
$K_i = K_c * (T_s/T_i)$ = coefficient of integral term
$K_r = K_c * (Td/T_s)$ = coefficient of derivative term
T_i = Reset time (integral time)
T_d = Rate time (derivative time)
SP_n = Set Point for sampling time "n" (SP value)
PV_n = Process variable for sampling time "n" (PV)
$e_n = SP_n – PV_n$ = Error term for sampling time "n"
M_n = Control Output for sampling time "n"

The resulting equations for the Velocity Algorithm form of the PID equation are:

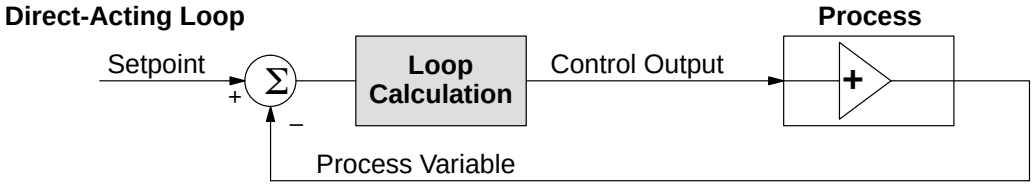
$$\Delta M_n = M_n – M_{n–1}$$

$$\Delta M_n = K_c * (e_n – e_{n–1}) + K_i * e_n + K_r * (e_n – 2*e_{n–1} + e_{n–2})$$

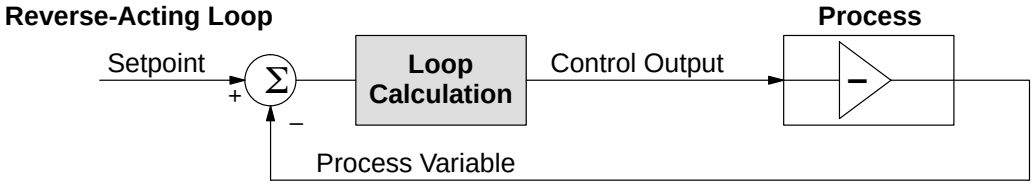
Direct-Acting and Reverse-Acting Loops

The gain of a process determines, in part, how it must be controlled. The process shown in the diagram below has a positive gain, which we call “direct-acting”. This means that when the control output increases, the process variable also eventually increases. Of course, a true process is usually a complex transfer function that includes time delays. Here, we are only interested in the direction of change of the process variable in response to a control output change.

Most process loops will be direct-acting, such as a temperature loop. An increase in the heat applied increases the PV (temperature). Accordingly, direct-acting loops are sometimes called *heating loops*.



A “reverse-acting” loop is one in which the process has a negative gain, as shown below. An increase in the control output results in a decrease in the PV. This is commonly found in refrigeration controls, where an increase in the cooling input causes a decrease in the PV (temperature). Accordingly, reverse-acting loops are sometimes called *cooling loops*.

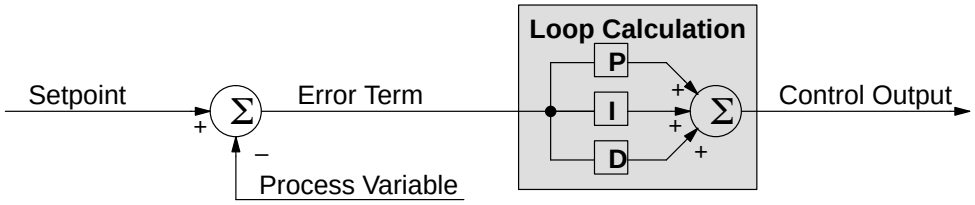


It is crucial to know whether a particular loop is direct or reverse-acting! Unless you are controlling temperature, there is no obvious answer. In a flow control loop, a valve positioning circuit can be configured and wired reverse-acting as easily as direct-acting. One easy way to find out is to run the loop in Manual Mode, where you must manually generate control output values. Observe whether the PV goes up or down in response to a step increase in the control output.

To run a loop in Auto or Cascade Mode, the control output must be correctly programmed (refer to the previous section on Control Output Configuration). Use “normal output” for direct-acting loops, and “inverted output” for reverse-acting loops. To compensate for a reverse-acting loop, the PID controller must know to invert the control output. If you have a choice, configure and wire the loop to be direct-acting. This will make it easier to view and interpret loop data during the loop tuning process.

P-I-D Loop Terms

You may recall the introduction of the position and velocity forms of the PID loop equations. The equations basically show the three components of the PID calculation. The following figure shows a schematic form of the PID calculation, in which the control output is the sum of the proportional, integral and derivative terms. On each calculation of the loop, each term receives the same error signal value.

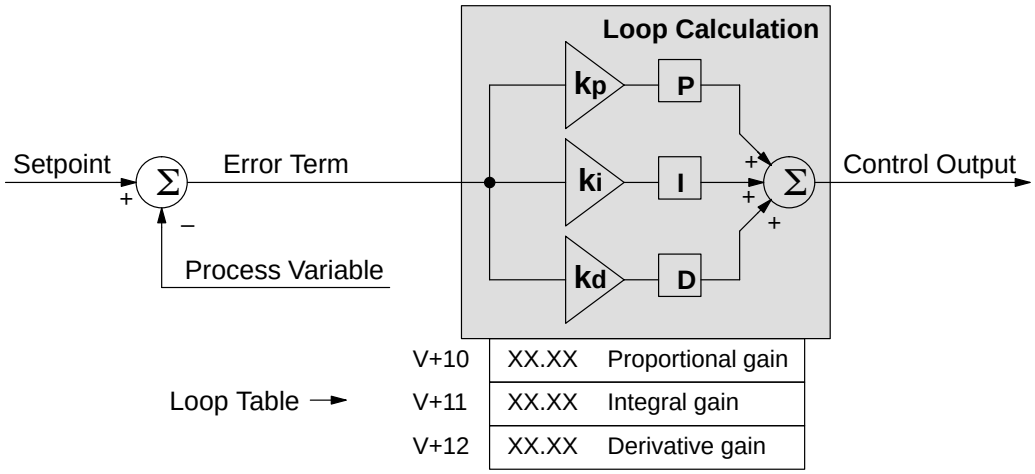


The role of the P, I, and D terms in the control task are as follows:

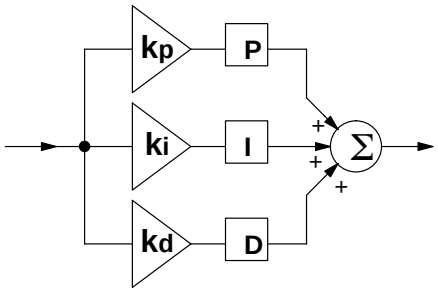
- **Proportional** – the proportional term simply responds proportionally to the current size of the error. This loop controller calculates a proportional term value for each PID calculation. When the error is zero, the proportional term is also zero.
- **Integral** – the integrator (or reset) term integrates (sums) the error values. Starting from the first PID calculation after entering Auto Mode, the integrator keeps a running total of the error values. For the position form of the PID equation, when the loop reaches equilibrium and there is no error, the running total represents the constant output required to hold the current position of the PV.
- **Derivative** – the derivative (or rate) term responds to change in the current error value from the error used in the previous PID calculation. Its job is to anticipate the probable growth of the error and generate a contribution to the output in advance.

The P, I, and D terms work together as a team. To do that effectively, they will need some additional instructions from us. The figure below shows the P, I, and D terms contain programmable **gain** values k_p, k_i, and k_d respectively. The values reside in the loop table in the locations shown. The goal of the loop tuning process (covered later) is to derive gain values that result in good overall loop performance.

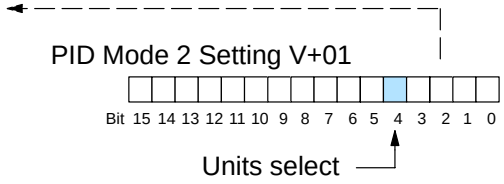
NOTE: The proportional gain is also simply called “gain”, in PID loop terminology.



The P, I and D gains are 4-digit BCD numbers with values from 0000 to 9999. They contain an implied decimal point in the middle, so the values are actually 00.00 to 99.99. Some gain values have units – Integral gain may be in units of seconds or minutes, by programming the bit shown. Derivative gain is in seconds.



V+10	XX.XX P gain	–
V+11	XX.XX I gain	0=sec, 1=min.
V+12	XX.XX D gain	sec.



In ***Direct***SOFT's trend view, you can program the gains values and units in real time while the loop is running. This is typically done only during the loop tuning process.

Proportional Gain – This is the most basic gain of the three. Values range from 0000 to 9999, but they are used internally as xx.xx. An entry of "0000" effectively removes the proportional term from the PID equation. This accommodates applications which need integral-only loops.

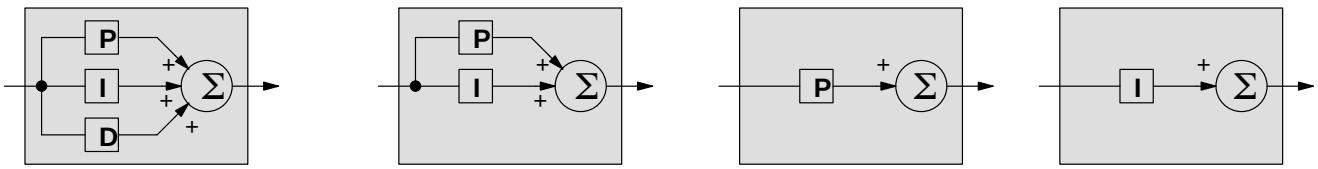
Integral Gain – Values range from 0001 to 9998, but they are used internally as xx.xx. An entry of "0000" or "9999"causes the integral gain to be "∞", effectively removing the integrator term from the PID equation. This accommodates applications which need proportional-only loops. The units of integral gain may be either seconds or minutes, as shown above.

Derivative Gain – Values range from 0001 to 9999, but they are used internally as xx.xx. An entry of "0000" allows removal of the derivative term from the PID equation (a common practice). This accommodates applications which need proportional and/or integral-only loops. The derivative term has an optional gain limiting feature, discussed in the next section.

NOTE: It is very important to know how to increase and decrease the gains. The proportional and derivative gains are as one might expect... smaller numbers produce less gains and larger numbers produce more gain. However, the integral term has a reciprocal gain(1/Ts), so smaller numbers produce more gain and larger numbers produce less gain. *This is very important to know during loop tuning.*

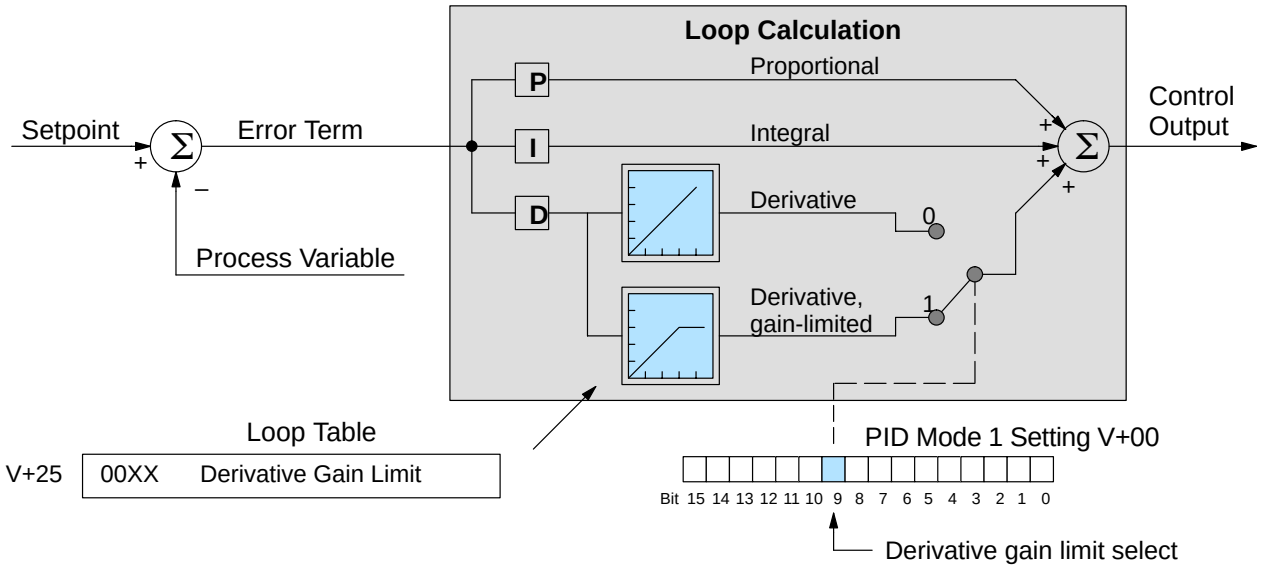
Using a Subset of PID Control

Each of the P, I, and D gains allows a setting to eliminate that term from the PID equation. Many applications actually work best by using a subset of PID control. The figure below shows the various combinations of PID control offered on the DL350. We do not recommend using any other combination of control, because most of them are inherently unstable.



Derivative Gain Limiting

The derivative term is unique in that it has an optional gain-limiting feature. This is provided because the derivative term reacts badly to PV signal noise or other causes of sudden PV fluctuations. The function of the gain-limiting is shown in the diagram below. Use bit 9 of PID Mode 1 Setting V+00 word to enable the gain limit.



The derivative gain limit in location V+25 must have a value between 0 and 20, in BCD format. This setting is operational only when the enable bit = 1.

The gain limit can be particularly useful during loop tuning. Most loops can tolerate only a little derivative gain without going into wild oscillations.

Bias Term

In the widely-used *position* form of the PID equation, an important component of the control output value is the bias term shown below. Its location in the loop table is in V+04. the loop controller writes a new bias term after each loop calculation.

$$M_n = K_c * e_n + K_i * \sum_{i=1}^{n} e_i + K_r * (e_n - e_{n-1}) + M_o$$

Control Output | Proportional Term | Integral Term | Derivative Term | Initial Output

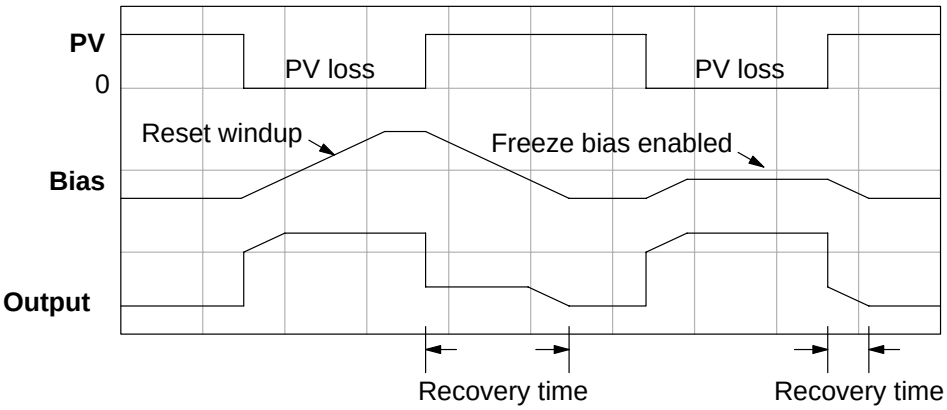
V+04 XXXX Bias term — Bias Term

If we cause the error (e_n) to go to zero for two or more sample periods, the proportional and derivative terms cancel. The bias term is the sum of the integral term and the initial output (M_o). It represents the steady, constant part of the control output value, and is similar to the DC component of a complex signal waveform.

The bias term value establishes a “working region” for the control output. *When the error fluctuates around its zero point, the output fluctuates around the bias value.* This concept is very important, because it shows us why the integrator term must respond more slowly to errors than either the proportional or derivative terms.

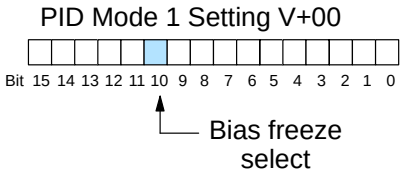
Bias Freeze

The term “reset windup” refers to an undesirable characteristic of integrator behavior which occurs naturally under certain conditions. Refer to the figure below. Suppose the PV signal becomes disconnected, and the PV value goes to zero. While this is a serious loop fault, it is made worse by *reset windup*. Notice the bias (reset) term keeps integrating normally during the PV disconnect, until its upper limit is reached. When the PV signal returns, the bias value is saturated (windup) and takes a long time to return to normal. The loop output consequently has an extended recovery time. Until recovery, the output level is wrong and causes further problems.



In the second PV signal loss episode in the figure, the freeze bias feature is enabled. It causes the bias value to freeze when the control output goes out of bounds. Much of the reset windup is thus avoided, and the output recovery time is much less.

For most applications, the freeze bias feature will work with the loop as described above. You may enable the feature using the ***Direct***SOFT PID View setup dialog, or set bit 10 of PID Mode 1 Setting word as shown to the right.



NOTE: The bias freeze feature stops the bias term from changing when the control output reaches the end of the data range. If you have set limits on the control output other than the range (i.e, 0–4095 for a unipolar/12bit loop), the bias term still uses the end of range for the stopping point and bias freeze will not work.

In the feedforward method discussed later in this chapter, ladder logic writes directly to the bias term value. However, there is no conflict with the freeze bias feature, because bias term writes due to feedforward are relatively infrequent when in use.

Loop Tuning Procedure

This is perhaps the most important step in closed-loop process control. The goal of a loop tuning procedure is to adjust the loop gains so the loop has optimal performance in dynamic conditions. The quality of a loop's performance may generally be judged by how well the PV follows the SP after a SP step change.

Auto Tuning versus Manual Tuning – you may change the PID gain values directly (manual tuning), or you can have the PID processing engine in the CPU automatically calculate the gains (auto tuning). Most experienced process engineers will have a favorite method, and the DL350 will accommodate either preference. The use of the auto tuning can eliminate much of the trial-and-error of the manual tuning approach, especially if you do not have a lot of loop tuning experience. However, note that performing the auto tuning procedure will get the gains *close* to optimal values, but additional manual tuning changes can take the gain values to their optimal values.

WARNING: Only authorized personnel fully familiar with all aspects of the process should make changes that affect the loop tuning constants. Using the loop auto tune procedures will affect the process, including inducing large changes in the control output value. Make sure you thoroughly consider the impact of any changes to minimize the risk of injury to personnel or damage to equipment. The auto tune in the DL350 is not intended to perform as a replacement for your process knowledge.

Open-Loop Test

Whether you use manual or auto tuning, it is very important to verify basic characteristics of a newly-installed process before attempting to tune it. With the loop in Manual Mode, verify the following items for each new loop.

- **Setpoint** – verify the source which is to generate the setpoint can do so. You can put the PLC in Run Mode, but leave the loop in Manual Mode. Then monitor the loop table location V+02 to see the SP value(s). The ramp/soak generator (if you are using it) should be tested now.
- **Process Variable** – verify the PV value is an accurate measurement, and the PV data arriving in the loop table location V+03 is correct. If the PV signal is very noisy, consider filtering the input either through hardware (RC low-pass filter), or using a digital S/W filter.
- **Control Output** – if it is safe to do so, manually change the output a small amount (perhaps 10%) and observe its affect on the process variable. Verify the process is direct-acting or reverse acting, and check the setting for the control output (inverted or non-inverted). Make sure the control output upper and lower limits are not equal to each other.
- **Sample Rate** – while operating open-loop, this is a good time to find the ideal sample rate (procedure give earlier in this chapter). However, if you are going to use auto tuning, note the auto tuning procedure will automatically calculate the sample rate in addition to the PID gains.

The discussion beginning on the following page covers the manual tuning procedure. If want to perform only auto tuning, please skip the next section and proceed directly to the section on auto tuning.

Manual Tuning Procedure

Now comes the exciting moment when we actually close the loop (go to Auto Mode) for the first time. Use the following checklist **before** switching to Auto mode:

- Monitor the loop parameters with a loop trending instrument. We recommend using the PID view feature of ***Direct***SOFT.

NOTE: We recommend using the PID trend view setup menu to select the vertical scale feature to *manual,* for both SP/PV area and Bias/Control Output areas. The auto scaling feature will otherwise change the vertical scale on the process parameters and add confusion to the loop tuning process.

- Adjust the gains so the Proportional Gain = 10, Integrator Gain = 9999, and Derivative Gain =0000. This disables the integrator and derivative terms, and provides a little proportional gain.
- Check the bias term value in the loop parameter table (V+04). If it is not zero, then write it to zero using ***Direct***SOFT or HPP, etc.

Now we can transition the loop to Auto Mode. Check the mode monitoring bits to verify its true mode. If the loop will not stay in Auto Mode, check the troubleshooting tips at the end of this chapter.

CAUTION: If the PV and Control Output values begin to oscillate, reduce the gain values immediately. If the loop does not stabilize immediately, then transfer the loop back to Manual Mode and manually write a safe value to the control output. **During the loop tuning procedure, always be near the Emergency Stop switch which controls power to the loop actuator in case a shutdown is necessary.**

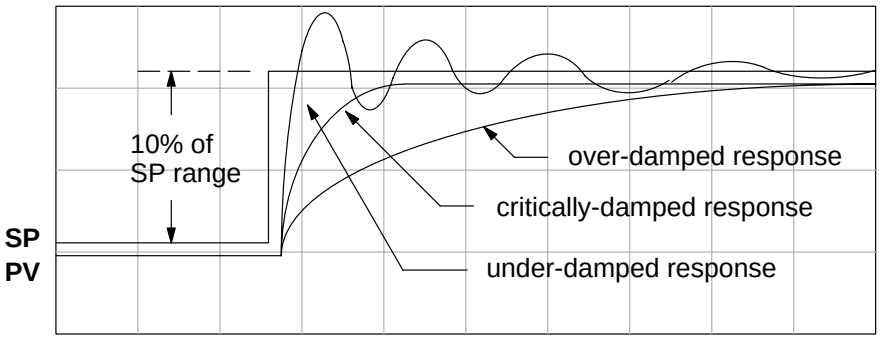
- At this point, the SP should = PV because of the bumpless transfer feature. Increase the SP a little, in order to develop an error value. With only the proportional gain active and the bias term=0, we can easily check the control output value:

  Control Output = (SP – PV) x proportional gain

- If the control output value changed, the loop should be getting more energy from the actuator, heater, or other device. Soon the PV should move in the direction of the SP. If the PV does not change, then increase the proportional gain until it moves slightly.
- Now, add a small amount of integral gain. **Remember that large numbers are small integrator gains and small numbers are large integrator gains!** After this step, the PV should = SP, or be very close.

Until this point we have only used proportional and integrator gains. Now we can “bump the process” (change the SP by 10%), and adjust the gains so the PV has an optimal response. Refer to the figure below. Adjust the gains according to what you see on the PID trend view. The critically- damped response shown gives the fastest PV response without oscillating.

- Over-damped response – the gains are too small, so gradually increase them, concentrating on the proportional gain first.
- Under-damped response – the gains are too large. Reduce the integral gain first, and then the proportional gain if necessary.
- Critically-damped response – this is the the optimal gain setting. You can verify that this is the best response by increasing the proportional gain slightly. the loop then should make one or two small oscillations.



Now you may want to add a little derivative gain to further improve the critically-damped response above. Note the proportional and integral gains will be very close to their final values at this point. Adding some derivative action will allow you to increase the proportional gain slightly without causing loop oscillations. The derivative action tends to tame the proportional response slightly, so adjust these gains together.

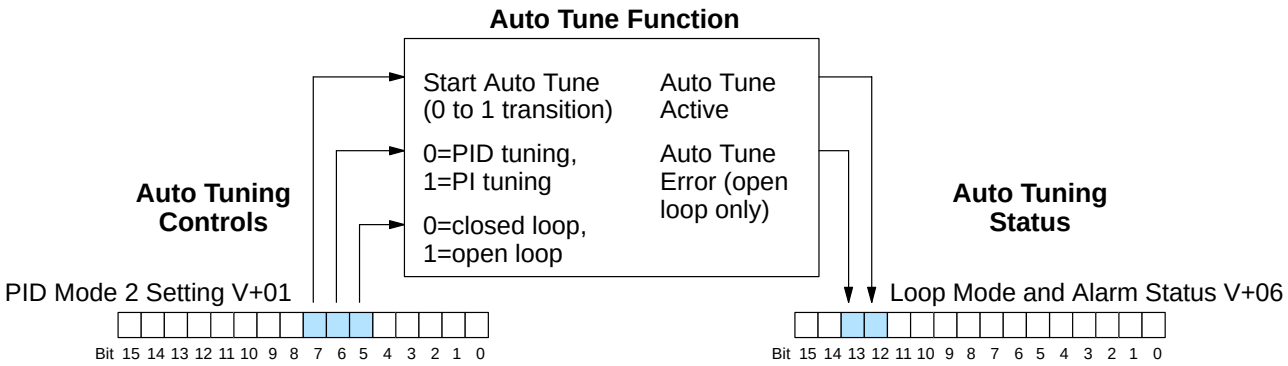
Auto Tuning Procedure

The auto tuning feature in the DL350 CPU loop controller runs only at the command of the process control engineer. The auto tuning therefore does not run continuously during operation (this would be *adaptive* control). Whenever a substantial change in loop dynamics occurs (mass of process, size of actuator, etc.), you will need to repeat the tuning procedure to derive the new gains that are required for optimal control.

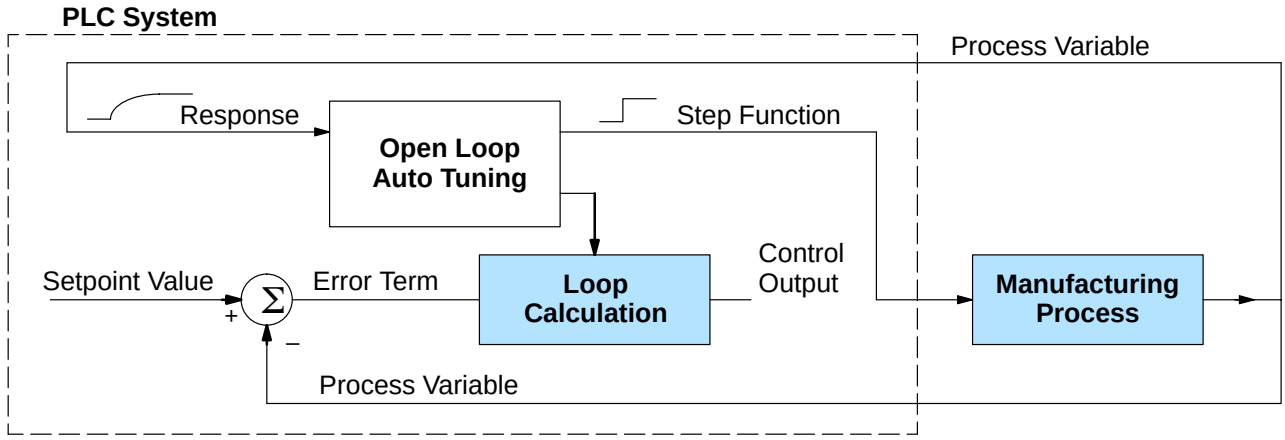
WARNING: Only authorized personnel fully familiar with all aspects of the process should make changes that affect the loop tuning constants. Using the loop auto tuning procedures will affect the process, including inducing large changes in the control output value. Make sure you thoroughly consider the impact of any changes to minimize the risk of injury to personnel or damage to equipment. The auto tune in the DL350 is not intended to perform as a replacement for your process knowledge.

The loop controller offers both closed-loop and open-loop methods. If you intend to use the auto tune feature, we recommend you use the open-loop method first. This will permit you to use the closed-loop method of auto tuning when the loop is operational (Auto Mode) and cannot be shut down (Manual Mode). The following sections describe how to use the auto tuning feature, and what occurs in open and closed-loop auto tuning.

The controls for the auto tuning function use three bits in the PID Mode 2 word V+01, as shown below. ***Direct***SOFT will manipulate these bits automatically when you use the auto tune feature within ***Direct***SOFT. Or, you may have ladder logic access these bits directly for allowing control from another source such as a dedicated operator interface. The individual control bits let you to start the auto tune procedure, select PID or PI tuning, and select closed-loop or open-loop tuning. If you select PI tuning, the auto tune procedure leaves the derivative gain at 0. The Loop Mode and Alarm Status word V+06 reports the auto tune status as shown. Bit 12 will be on (1) when during the auto tuning cycle, automatically returning to off (0) when done.



Open-Loop Auto Tuning – During an open-loop auto tuning cycle, the loop controller operates as shown in the diagram below. Before starting this procedure, place the loop in Manual mode and ensure the PV and control output values are in the middle of their ranges (away from the end points).

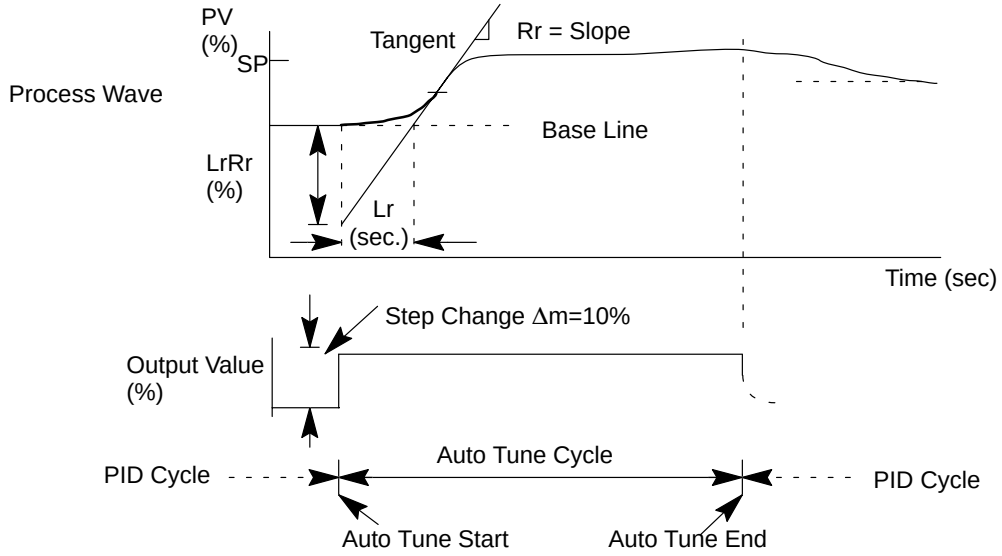


NOTE: In theory, the SP value does not matter in this case, because the loop is not closed. However, the firmware requires that the SP value be more than 205 counts away from the PV value before starting the auto tune cycle (205 counts or more below the SP for forward-acting loops, or 205 counts or more above the SP for reverse-acting loops).

When auto tuning, the loop controller induces a step change on the output and simply observes the response of the PV. From the PV response, the auto tune function calculates the gains and the sample time. It automatically places the results in the corresponding registers in the loop table.

The following timing diagram shows the events which occur in the open-loop auto tuning cycle. The auto tune function takes control of the control output and induces a 10%-of-span step change. If the PV change which the loop controller observes is less than 2%, then the step change on the output is increased to 20%-of-span.

Open Loop Auto Tune Cycle Wave: Step Response Method



* When Auto Tune starts, step change output Δm=10%

* During Auto Tune, the controller output reached the full scale positive limit. Auto Tune stopped and the Auto Tune Error bit in the Alarm word bit turned on.

* When PV change is under 2%, output is changed at 20%.

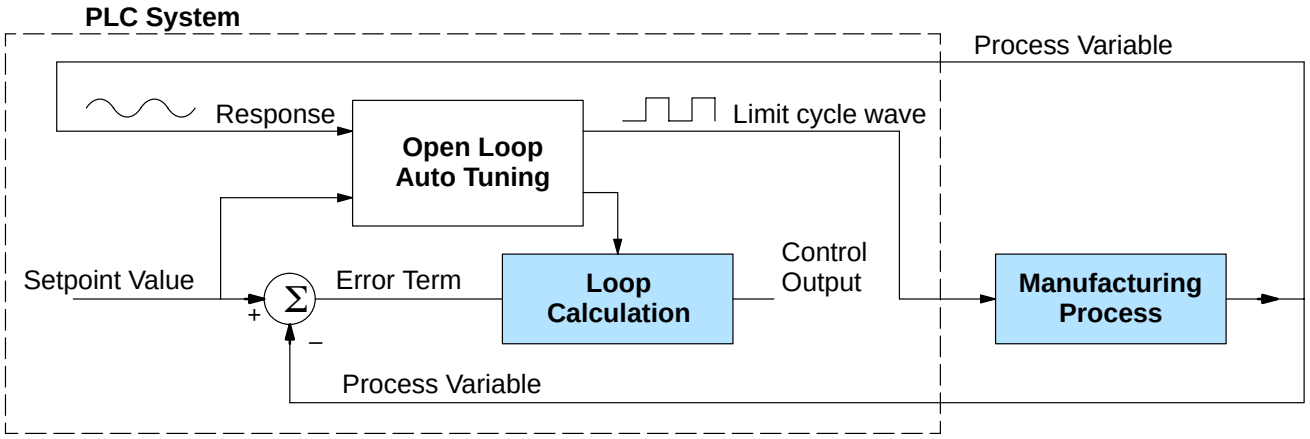
When the loop tuning observations are complete, the loop controller computes Rr (maximum slope in %/sec.) and Lr (dead time in sec). The auto tune function computes the gains according to the Ziegler-Nichols equations, shown below:

PID tuning:	**PI tuning:**
P = 1.2 * Δm/LrRr	P = 0.9 * Δm/LrRr
I = 2.0 * Lr	I = 3.33 * Lr
D = 0.5 * Lr	D = 0
Sample Rate = 0.056 * Lr	Sample Rate = 0.12 * Lr

Δm = Output step change (10% = 0.1, 20% = 0.2)

We highly recommend using ***Direct***SOFT for the auto tuning interface. the duration of each auto tuning cycle will depend on the mass of our process. A slowly-changing PV will result in a longer auto tune cycle time. When the auto tuning is complete, the proportional, integral, and derivative gain values are automatically updated in loop table locations V+10, V+11, and V+12 respectively. The sample time in V+07 is also updated automatically. You can test the validity of the values the auto tuning procedure yields by measuring the closed-loop response of the PV to a step change in the output. The instructions on how to do this are in the section on the manual tuning procedure (located prior to this section on auto tuning).

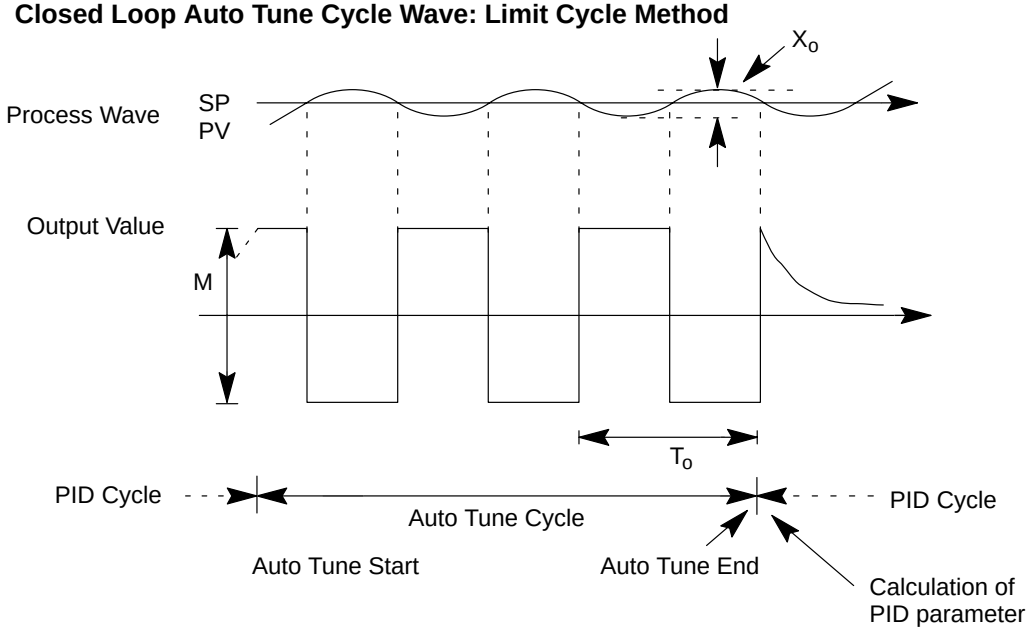
Closed-Loop Auto Tuning – During an close-loop auto tuning cycle, the loop controller operates as shown in the diagram below.



When auto tuning, the loop controller imposes a square wave on the output. Each transition of the output occurs when the PV value crosses over (or under) the SP value. Therefore, the frequency of the limit cycle is roughly proportional to the mass of the process. From the PV response, the auto tune function calculates the gains and the sample time. It automatically places the results in the corresponding registers in the loop table.

The following timing diagram shows the events which occur in the closed-loop auto tuning cycle. The auto tune function takes control of the control output and induces a 10%-of-span step increase. When the sign of the error (SP – PV) changes, the output changes to a 10%-of-span step decrease (below the nominal output value).

Closed Loop Auto Tune Cycle Wave: Limit Cycle Method



*$M_{max} \leq$ Output Value upper limit setting $M_{min} \geq$ Output Value lower limit setting.

* This example is direct–acting. When set at reverse–acting, output is inverted.

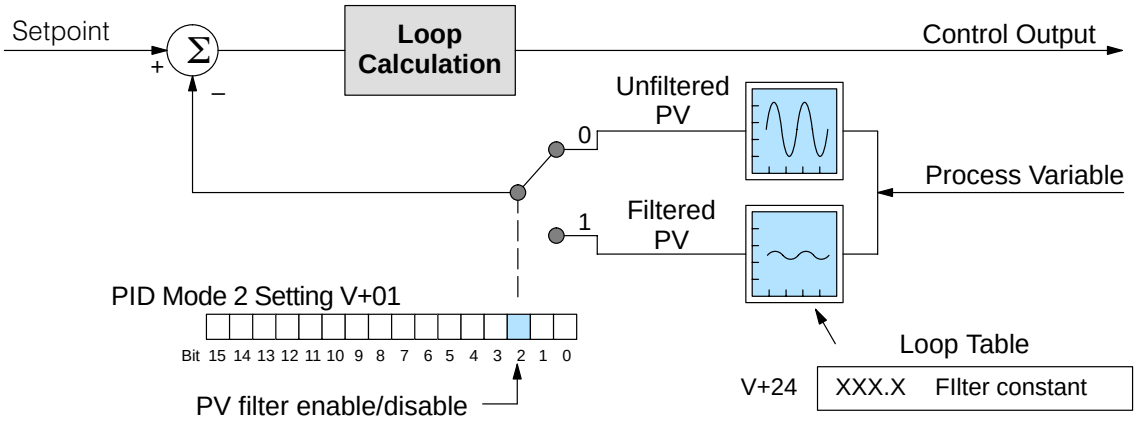
When the loop tuning observations are complete, the loop controller computes To (bump period) and Xo (amplitude of the PV). Then it uses these values to compute Kpc (sensitive limit) and Tpc (period limit). From these values, the loop controller auto tune function computes the PID gains and the sample rate according to the Ziegler-Nichols equations shown below:

$Kpc = 4M / (\pi * Xo)$ $\quad Tpc = 0$

M = amplitude of output

PID tuning:	**PI tuning:**
P = 0.45 * Kpc	P = 0.30 *Kpc
I = 0.60 * Tpc	I = 1.00 * Tpc
D = 0.10 * Tpc	D = 0
Sample Rate = 0.014 * Tpc	Sample Rate = 0.03 * Tpc

As you can see from the timing diagram on the previous page, the zero-crossing of the SP and PV difference is important. Obviously, a noisy PV signal can create extra zero-crossings and give a false indication of loop characteristics to the loop controller. The DL350 provides a selectable first-order low-pass PV input filter specifically for you to use during auto tuning, using the closed-loop method. Shown in the figure below, **we strongly recommend the use of this filter during auto tuning.** You may disable the filter after auto tuning is complete, or continue to use it if the PV input signal is noisy.



Bit 2 of PID Mode 2 Setting provides the enable/disable control for the low-pass PV filter (0=disable, 1=enable). The roll-off frequency of the single-pole low-pass filter is controlled by using register V+24 in the loop parameter table, the filter constant. The data format of the filter constant value is BCD, with an implied decimal point 00X.X, as follows:

- The filter constant has a range of 000.1 to 001.0.
- A setting of 000.0 or 001.1 to 999.9 essentially disables the filter.
- Values close to 001.0 result in higher roll-off frequencies, while values closer to 000.1 result in lower roll-off frequencies.

We highly recommend using ***Direct***SOFT for the auto tuning interface. The duration of each auto tuning cycle will depend on the mass of our process. A slowly-changing PV will result in a longer auto tune cycle time. When the auto tuning is complete, the proportional, integral, and derivative gain values are automatically updated in loop table locations V+10, V+11, and V+12 respectively. The sample time in V+07 is also updated automatically. You can test the validity of the values the auto tuning procedure yields by measuring the closed-loop response of the PV to a step change in the output. The instructions on how to so this are in the section on the manual tuning procedure (located before this section).

Auto tuning error – if the auto tune error bit (bit 13 of Loop Mode and Alarm status word V+06) is on, please verify the PV and SP values are within 5% of full scale difference, as required by the auto tune function. The bit will also turn on if the closed-loop method is in use, and the output goes to the limits of the range.

Tuning Cascaded Loops

In tuning cascaded loops, we will need to de-couple the cascade relationship and tune the loops individually, using one of the loop tuning procedures previously covered.

1. If you are not using auto tuning, then find the loop sample rate for the minor loop, using the method discussed earlier in this chapter. Then set the sample rate of the major loop slower than the minor loop by a factor of 10. Use this as a starting point.

2. Tune the minor loop first. Leave the major loop in Manual Mode, and you will need to generate SP changes for the minor loop manually as described in the loop tuning procedure.

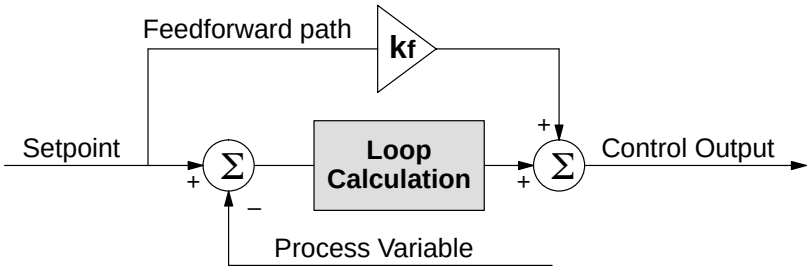
3. Verify the minor loop gives a critically-damped response to a 10% SP change while in Auto Mode. Then we are finished tuning the minor loop.

4. In this step, you will need to get the minor loop in Cascade Mode, and then the Major loop in Auto Mode. We will be tuning the major loop with the minor loop treated as a series component its overall process. Therefore, do not go back and tune the minor loop again while tuning the major loop.

5. Tune the major loop, following the standard loop tuning procedure in this section. The response of the major loop PV is actually the overall response of the cascaded loops together.

Feedforward Control

Feedforward control is an enhancement to standard closed-loop control. It is most useful for diminishing the effects of a *quantifiable and predictable* loop disturbance or sudden change in setpoint. Use of this feature is an option available to you on the DL350. However, it's best to implement and tune a loop without feedforward, and adding it only if better loop performance is still needed. The term "feed-forward" refers to the control technique involved, shown in the diagram below. The incoming setpoint value is fed forward around the PID equation, and summed with the output.

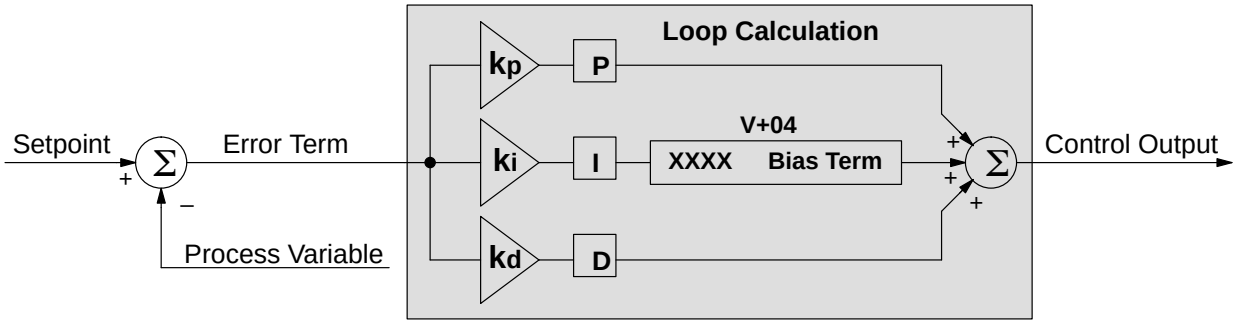


In the previous section on the bias term, we said that "the bias term value establishes a "working region" or operating point for the control output. *When the error fluctuates around its zero point, the output fluctuates around the bias value.*" Now, when there is a change in setpoint, an error is generated and the output must change to a new operating point. This also happens if a disturbance introduces a new offset in the loop. The loop does not really "know its way" to the new operating point... the integrator (bias) must increment/decrement until the error disappears, and then the bias has found the new operating point.

Suppose that we are able to know a sudden setpoint change is about to occur (common in some applications). We can avoid much of the resulting error in the first place, if we can quickly change the output to the new operating point. If we know (from previous testing) what the operating point (bias value) will be after the setpoint change, we can artificially change the output directly (which is feedforward). The benefits from using feedforward are:

- The SP–PV error is reduced during predictable setpoint changes or loop offset disturbances.
- Proper use of feedforward will allow us to reduce the integrator gain. Reducing integrator gain gives us an even more stable control system.

Feedforward is very easy to use in the DL350 loop controller, as shown below. The bias term has been made available to the user in a special read/write location, at PID Parameter Table location V+04.

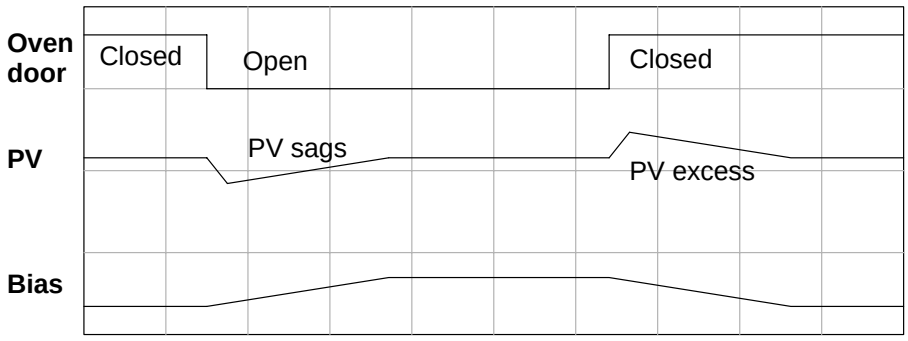


To change the bias (operating point), ladder logic only has to write the desired value to V+04. The PID loop calculation first reads the bias value from V+04 and modifies the value based on the current integrator calculation. Then it writes the result back to location V+04. This arrangement creates a sort of “transparent” bias term. All you have to do to implement feed forward control is write the correct value to the bias term at the right time (the example below shows you how).

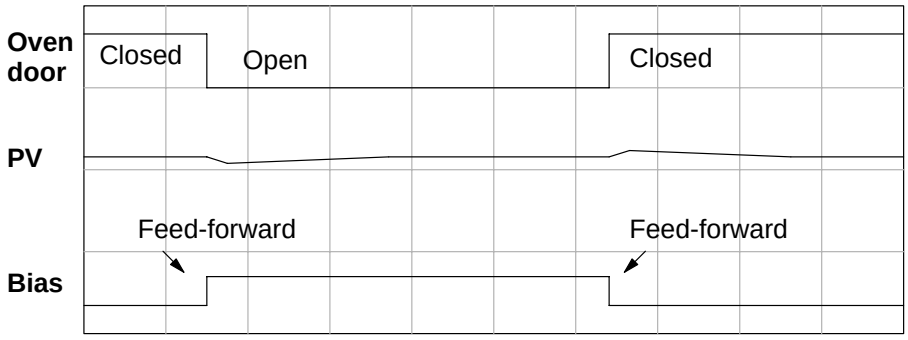
NOTE: When writing the bias term, one must be careful to design ladder logic to write the value only once, at the moment when the new bias operating point is to occur. If ladder logic writes the bias value on every scan, the loop’s integrator is effectively disabled.

Feedforward Example

How do we know when to write to the bias term, and what value to write? Suppose we have an oven temperature control loop, and we have already tuned the loop for optimal performance. Refer to the figure below. We notice that when the operator opens the oven door, the temperature sags a bit while the loop bias adjusts to the heat loss. Then when the door closes, the temperature rises above the SP until the loop adjusts again. Feedforward control can help diminish this effect.



First, we record the amount of bias change the loop controller generates when the door opens or closes. Then, we write a ladder program to monitor the position of an oven door limit switch. When the door opens, our ladder program reads the current bias value from V+04, adds the desired change amount, and writes it back to V+04. When the door closes, we duplicate the procedure, but subtracting desired change amount instead. The following figure shows the results.



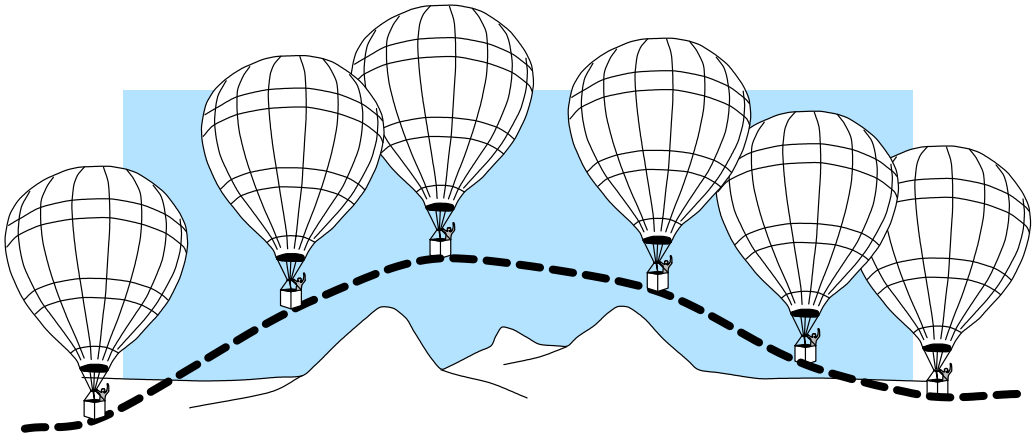
The step changes in the bias are the result of our two feed-forward writes to the bias term. We can see the PV variations are greatly reduced. The same technique may be applied for changes in setpoint.

Time-Proportioning Control

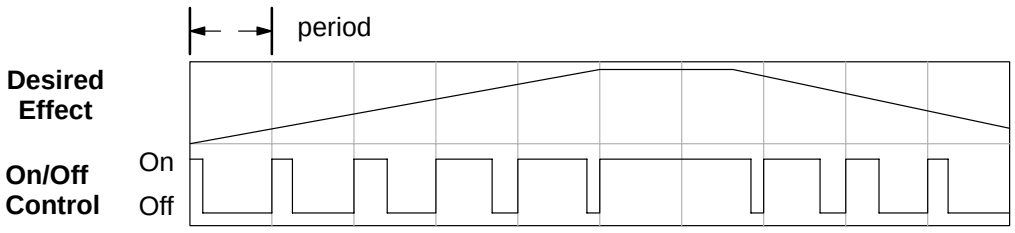
The PID loop controller in the DL350 CPU generates a smooth control output signal across a numerical range. The control output value is suitable to drive an analog output module, which connects to the process. In the process control field, this is called *continuous control*, because the output is on (at some level) continuously.

While continuous control can be smooth and robust, the cost of the loop components (such as actuators, heater amplifiers) can be expensive. A simpler form of control is called *time-proportioning control*. This method uses actuators which are either on or off (no in-between). Loop components for on/off-based control systems are lower cost than their continuous control counterparts.

In this section, we will show you how to convert the control output of a loop to time-proportioning control for the applications that need it. Let's take a moment to review how alternately turning a load on and off can control a process. The diagram below shows a hot-air balloon following a path across some mountains. The desired path is the *setpoint*. The balloon pilot turns the burner on and off alternately, which is his *control output*. The large mass of air in the balloon effectively averages the effect of the burner, converting the bursts of heat into a continuous effect: slowly changing balloon temperature and ultimately the altitude, which is the *process variable*.



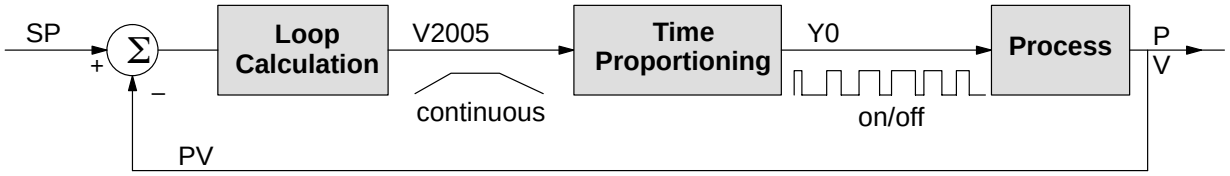
Time-proportioning control approximates continuous control by virtue of its duty-cycle – the ratio of ON time to OFF time. The following figure shows an example of how duty cycle approximates a continuous level when it is averaged by a large process mass.



If we were to plot the on/off times of the burner in the hot-air balloon, we would probably see a very similar relationship to its effect on balloon temperature and altitude.

On/Off Control Program Example

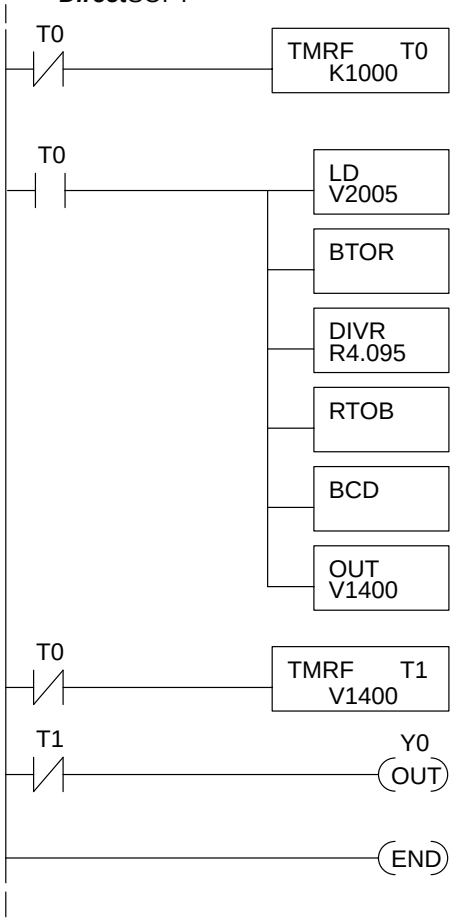
The following ladder example program does the time proportioning function as shown below. It converts the continuous output in V2005 to on/off control on Y0.



The example program uses two timers to generate On/Off control. It makes the following assumptions, which you can adapt to your application:

- The loop table starts at V2000, so the control output is at V2005.
- The data format of the control output is 12-bit, unipolar (0 – FFF).
- The time base (one cycle time) for the On/Off waveform is 10 seconds. This number should be approximately equal to the loop's sample rate. We use a fast timer (0.01 sec/tick), counting to 10.00. This gives the on/off output a resolution of 1 part in 1000. If your loop sample rate is much faster than this, you will have proportionately less resolution on the control output (and probably need to use continuous control).
- The On/Off control output is Y0. The duty cycle of the waveform on Y0 matches the value of the control output at V2005, with a period of 10 seconds.

DirectSOFT



Use a fast timer (0.01 sec. resolution) for the main time base. The K1000 provides a preset of 10 seconds. The Not T0 contact makes this self-resetting. T0 is true for one scan each 10 seconds.

At the beginning of the 10 second period, T0 is true. Load the binary control output value from loop table V+05 location, V2005.

Change this value to a real number, because we need to do some scaling.

Divide the control output by 4.095. This converts our 0 – 4095 range to 0 – 1000, in order to match our timer range for 10 seconds.

Convert the real number back to binary. We insert this step, because there is no real-to-BCD instruction.

Convert the number in the accumulator to BCD format, to satisfy the timer preset format required.

Output our result to V1400. This is our arbitrary location of the timer preset for the second timer.

The second fast timer also counts from 0 to 1000, in 10 seconds. However, its output, T1, goes from on to off when the preset is reached.

We invert the timer output, so the On/Off control output is On at the beginning of the 10-second timebase. Y0 goes to the actuator, heater, etc. for the loop.

END coil marks the end of the main program.

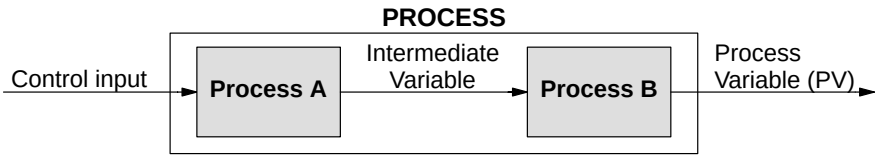
Cascade Control

Introduction

Cascaded loops are an advanced control technique that is superior to individual loop control in certain situations. As the name implies, cascade means that one loop is connected to another loop. In addition to Manual (open loop) and Auto (closed loop) Modes, the DL350 also provides Cascaded Mode.

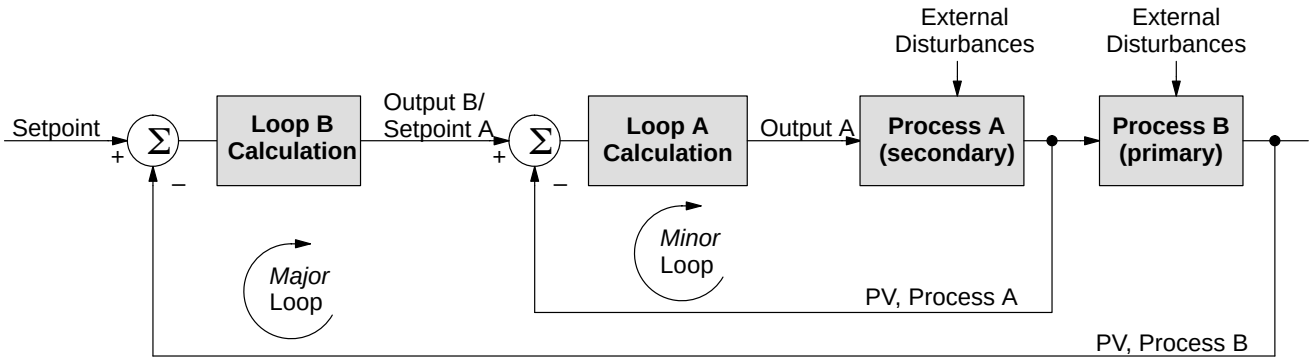
NOTE: Cascaded loops are an advanced process control technique. Therefore we recommend their use only for experienced process control engineers.

When a manufacturing process is complex and contains a lag time from control input to process variable output, even the most perfectly tuned single loop around the process may yield slow and inaccurate control. It may be the actuator operates on one physical property, which eventually affects the process variable, measured by a different physical property. Identifying the intermediate variable allows us to divide the process into two parts as shown in the following figure.



The principle of cascaded loops is simply that we add another process loop to more precisely control the intermediate variable! This separates the source of the control lag into two parts, as well.

The diagram below shows a cascade control system, showing that it is simply one loop nested inside another. The inside loop is called the minor loop, and the outside loop is called the major loop. For overall stability, the minor loop must be the fastest responding loop of the two. We do have to add the additional sensor to measure the intermediate variable (PV for process A). Notice the setpoint for the minor loop is automatically generated for us, by using the output of the major loop. Once the cascaded control is programmed and debugged, we only need to deal with the original setpoint and process variable at the system level. The cascaded loops behave as one loop, but with improved performance over the previous single-loop solution.



One of the benefits to cascade control can be seen by examining its response to external disturbances. Remember the minor loop is faster acting than the major loop. Therefore, if a disturbance affects process A in the minor loop, the Loop A PID calculation can correct the resulting error before the major loop sees the effect.

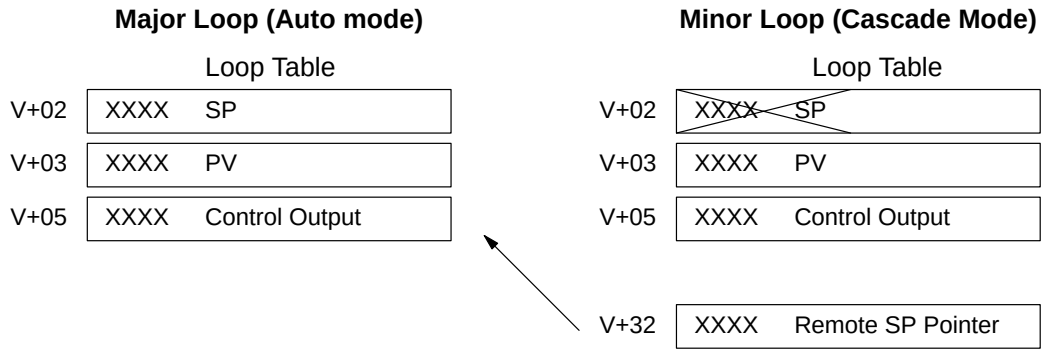
Cascaded Loops in the DL350 CPU

In the use of the term “cascaded loops”, we must make an important distinction. Only the minor loop will actually be in the Cascade Mode. In normal operation, the major loop must be in Auto Mode. If you have more than two loops cascaded together, the outer-most (major) loop must be in Auto Mode during normal operation, and all inner loops in Cascade Mode.

NOTE: Technically, both major and minor loops are “cascaded” in strict process control terminology. Unfortunately, we are unable to retain this convention when controlling loop modes. Remember that all minor loops will be in Cascade Mode, and only the outer-most (major) loop will be in Auto Mode.

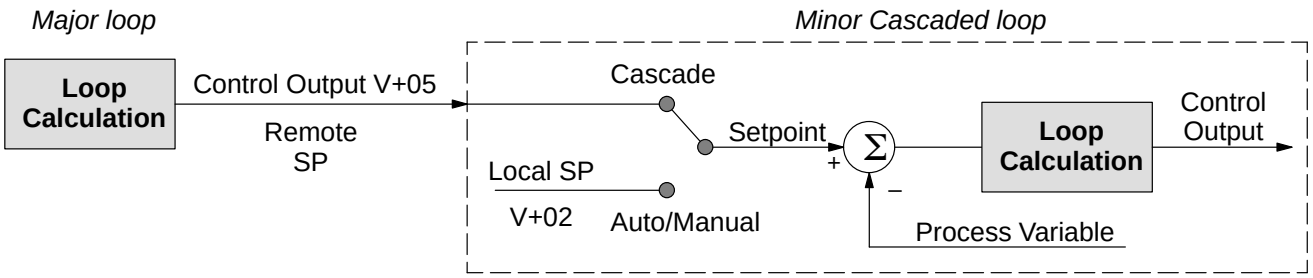
You can cascade together as many loops as necessary on the DL350, and you may have multiple groups of cascaded loops. For proper operation on cascaded loops you must use the same data range (12/15 bit) and polar/bipolar settings on the major and minor loop.

To prepare a loop for Cascade Mode operation as a minor loop, you must program its remote Setpoint Pointer in its loop parameter table location V+32, as shown below. The pointer must be the address of the V+05 location (control output) of the major loop. In Cascade Mode, the minor loop will ignore the its local SP register (V+02), and read the major loop’s control output as its SP instead.



When using ***Direct***SOFT’s PID View to watch the SP value of the minor loop, ***Direct***SOFT automatically reads the major loop’s control output and displays it for the minor loop’s SP. The minor loop’s normal SP location, V+02, remains unchanged.

Now, we use the loop parameter arrangement above and draw its equivalent loop schematic, shown below.



Remember that a major loop goes to Manual Mode automatically if its minor loop is taken out of Cascade Mode.

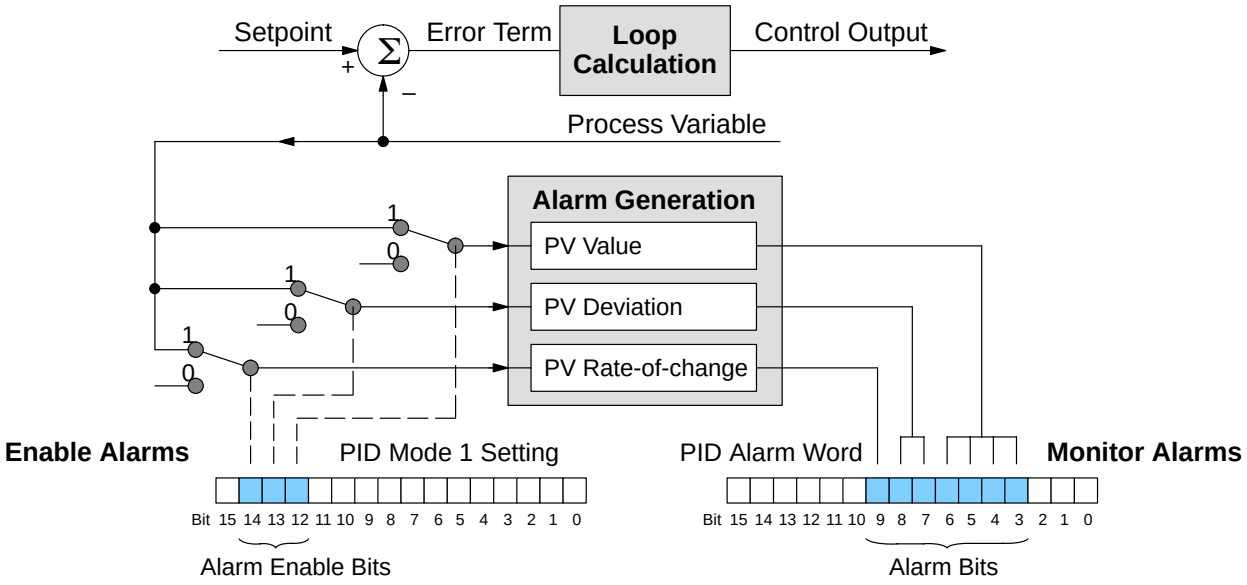
Process Alarms

The performance of a process control loop may be generally measured by how closely the process variable matches the setpoint. Most process control loops in industry operate continuously, and will eventually lose control of the PV due to an error condition. Process alarms are vital in early discovery of a loop error condition, and can alert plant personnel to manually control a loop or take other measures until the error condition has been repaired.

The DL350 CPU has a sophisticated set of alarm features for each loop:

- **PV Absolute Value Alarms** – monitors the PV with respect to two lower limit values and two upper limit values. It generates alarms whenever the PV goes outside these programmed limits.
- **PV Deviation Alarm** – monitors the PV value as compared to the SP. It alarms when the difference between the PV and SP exceed the programmed alarm value.
- **PV Rate-of-change Alarm** – computes the rate-of-change of the PV, and alarms if it exceeds the programmed alarm amount
- **Alarm Hysteresis** – works in conjunction with the absolute value and deviation alarms to eliminate alarm “chatter” near alarm thresholds.

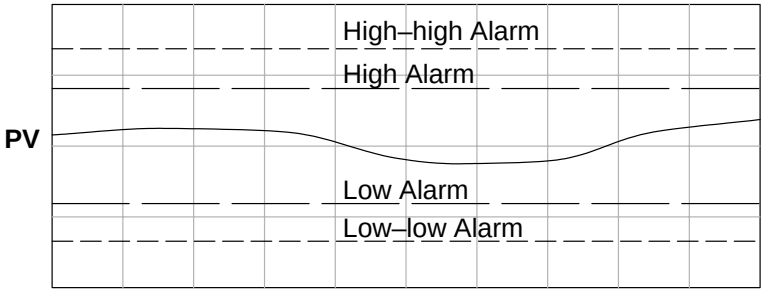
The alarm thresholds are fully programmable, and each type of alarm may be independently enabled and monitored. The following diagram shows the PV monitoring function. Bits 12, 13, and 14 of PID Mode 1 Setting V+00 word in the loop parameter table to enable/disable the alarms. ***Direct***SOFT’s PID View setup dialog screens allow easy programming, enabling, and monitoring of the alarms. Ladder logic may monitor the alarm status by examining bits 3 through 9 of PID Mode and alarm Status word V+06 in the loop table.



Unlike the PID calculations, the alarms are always functioning any time the CPU is in Run Mode. The loop may be in Manual, Auto, or Cascade, and the alarms will be functioning if the enable bit(s) as listed above are set =1.

PV Absolute Value Alarms

The PV absolute value alarms are organized as two upper and two lower alarms. The alarm status is false as long as the PV value remains in the region between the upper and lower alarms, as shown below. The alarms nearest the safe zone are named *High Alarm* and *Low Alarm*. If the loop loses control, the PV will cross one of these thresholds first. Therefore, you can program the appropriate alarm threshold values in the loop table locations shown below to the right. The data format is the same as the PV and SP (12-bit or 15-bit). The threshold values for these alarms should be set to give an operator an early warning if the process loses control.



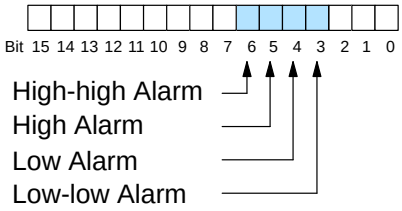
Loop Table

V+16	XXXX	High-high Alarm
V+15	XXXX	High Alarm
V+14	XXXX	Low Alarm
V+13	XXXX	Low-low Alarm

If the process remains out of control for some time, the PV will eventually cross one of the outer alarm thresholds, named High-high alarm and Low-low alarm. Their threshold values are programmed using the loop table registers listed above. A High-high or Low-low alarm indicates a serious condition exists, and needs the immediate attention of the operator.

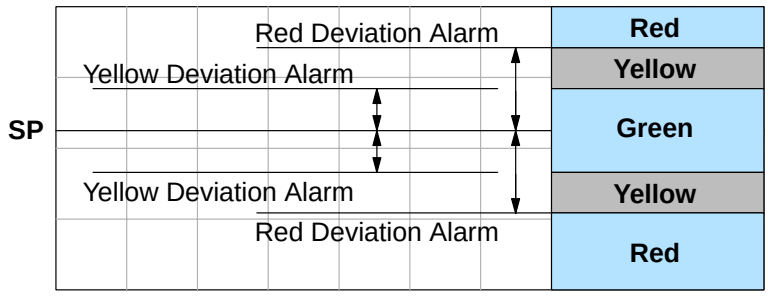
The PV Absolute Value Alarms are reported in the four bits in the PID Mode and Alarm Status word in the loop table, as shown to the right. We highly recommend using ladder logic to monitor these bits. The bit-of-word instructions make this easy to do. Additionally, you can monitor PID alarms using ***Direct***SOFT.

PID Mode and Alarm Status V+06



PV Deviation Alarms

The PV Deviation Alarms monitor the PV deviation with respect to the SP value. The deviation alarm has two programmable thresholds, and each threshold is applied equally above and below the current SP value. In the figure below, the smaller deviation alarm is called the “Yellow Deviation”, indicating a cautionary condition for the loop. The larger deviation alarm is called the “Red Deviation”, indicating a strong error condition for the loop. The threshold values use the loop parameter table locations V+17 and V+20 as shown.

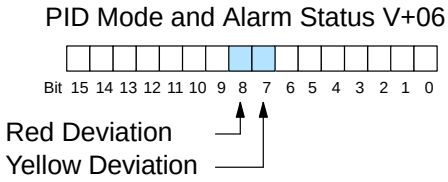


Loop Table

V+17	XXXX	Yellow Deviation Alarm
V+20	XXXX	Red Deviation Alarm

The thresholds define zones, which fluctuate with the SP value. The green zone which surrounds the SP value represents a safe (no alarm) condition. The yellow zones lie outside the green zone, and the red zones are beyond those.

The PV Deviation Alarms are reported in the two bits in the PID Mode and Alarm Status word in the loop table, as shown to the right. We highly recommend using ladder logic to monitor these bits. The bit-of-word instructions make this easy to do. Additionally, you can monitor PID alarms using ***Direct***SOFT.



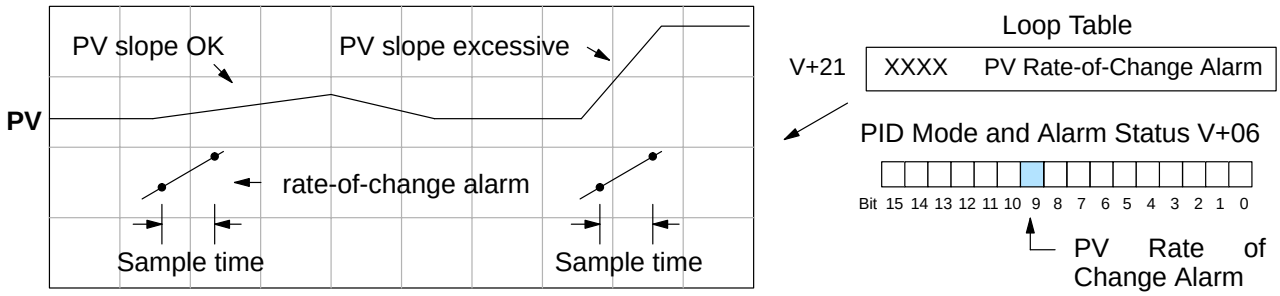
The PV Deviation Alarm can be independently enabled and disabled from the other PV alarms, using bit 13 of the PID Mode 1 Setting V+00 word.

Remember the alarm hysteresis feature works in conjunction with both the deviation and absolute value alarms, and is discussed at the end of this section.

PV Rate-of-Change Alarm

One powerful way to get an early warning of a process fault is to monitor the *rate-of-change* of the PV. Most batch processes have large masses and slowly-changing PV values. A relatively fast-changing PV will result from a broken signal wire for either the PV or control output, a SP value error, or other causes. If the operator responds to a PV Rate-of-Change Alarm quickly and effectively, the PV absolute value will not reach the point where the material in process would be ruined.

The DL350 loop controller provides a programmable PV Rate-of-Change Alarm, as shown below. The rate-of-change is specified in PV units change per loop sample time. This value is programmed into the loop table location V+21.



As an example, suppose the PV is temperature for our process, and we want an alarm when the temperature changes faster than 15 degrees / minute. We must know PV counts per degree and the loop sample rate. Then, suppose the PV value (in V+03 location) represents 10 counts per degree, and the loop sample rate is 2 seconds. We will use the formula below to convert our engineering units to counts / sample period:

$$\text{Alarm Rate-of-Change} = \frac{15 \text{ degrees}}{1 \text{ minute}} \times \frac{10 \text{ counts / degree}}{30 \text{ loop samples / min.}} = \frac{150}{30} = 5 \text{ counts / sample period}$$

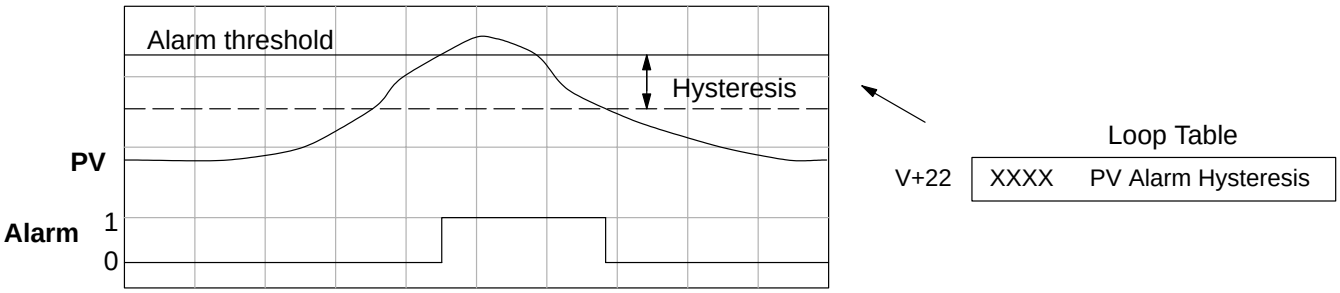
From the calculation result, we would program the value “5” in the loop table for the rate-of-change. The PV Rate-of-Change Alarm can be independently enabled and disabled from the other PV alarms, using bit 14 of the PID Mode 1 Setting V+00 word.

The alarm hysteresis feature (discussed next) does not affect the Rate-of-Change Alarm.

PV Alarm Hysteresis

The PV Absolute Value Alarm and PV Deviation Alarm are programmed using threshold values. When the absolute value or deviation exceeds the threshold, the alarm status becomes true. Real-world PV signals have some noise on them, which can cause some fluctuation in the PV value in the CPU. As the PV value crosses an alarm threshold, its fluctuations cause the alarm to be intermittent and annoy process operators. The solution is to use the PV Alarm Hysteresis feature.

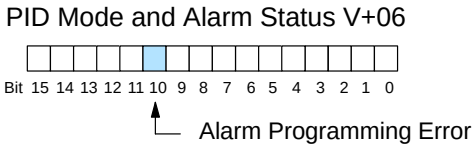
The PV Alarm Hysteresis amount is programmable from 1 to 200 (hex). When using the PV Deviation Alarm, the programmed hysteresis amount must be less than the programmed deviation amount. The figure below shows how the hysteresis is applied when the PV value goes past a threshold and descends back through it.



The hysteresis amount is applied *after* the threshold is crossed, and toward the safe zone. In this way, the alarm activates immediately above the programmed threshold value. It delays turning off until the PV value has returned through the threshold by the hysteresis amount.

Alarm Programing Error

The PV Alarm threshold values must have certain mathematical relationships to be valid. The requirements are listed below. If not met, the Alarm Programming Error bit will be set, as indicated to the right.

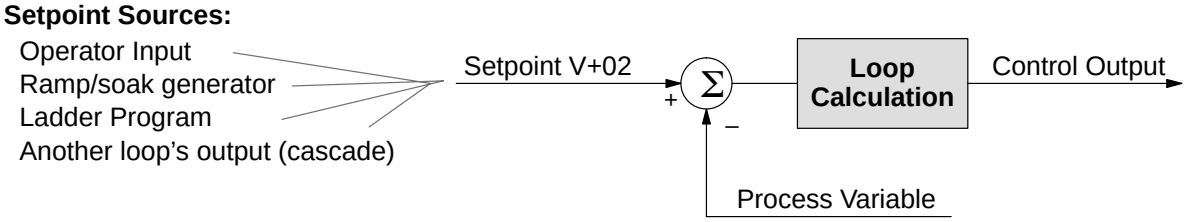


- PV Absolute Alarm value requirements:
  Low-low < Low < High < High-high
- PV Deviation Alarm requirements:
  Yellow < Red

Ramp/Soak Generator

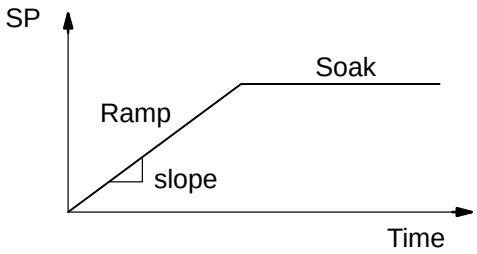
Introduction

Our discussion of basic loop operation noted the setpoint for a loop will be generated in various ways, depending on the loop operating mode and programming preferences. In the figure below, the ramp / soak generator is one of the ways the SP may be generated. *It is the responsibility of your ladder program to ensure only one source attempts to write the SP value at V+02 at any particular time.*



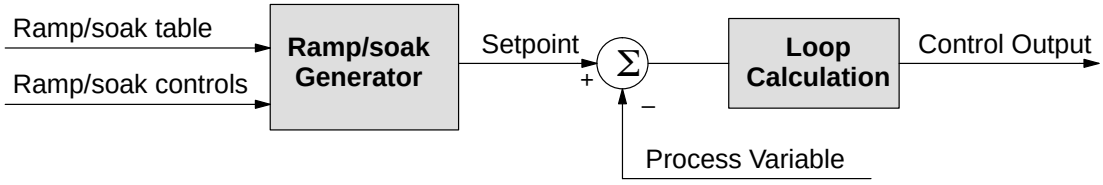
If the SP for your process rarely changes or can tolerate step changes, you probably will not need to use the ramp/soak generator. However, some processes require precisely-controlled SP value changes. The ramp / soak generator can greatly reduce the amount of programming required for these applications.

The terms “ramp” and “soak” have special meanings in the process control industry, and refer to desired setpoint (SP) values in temperature control applications. In the figure to the right, the setpoint increases during the ramp segment. It remains steady at one value during the soak segment.



Complex SP profiles can be generated by specifying a series of ramp/soak segments. The ramp segments are specified in SP units per second time. The soak time is also programmable in minutes.

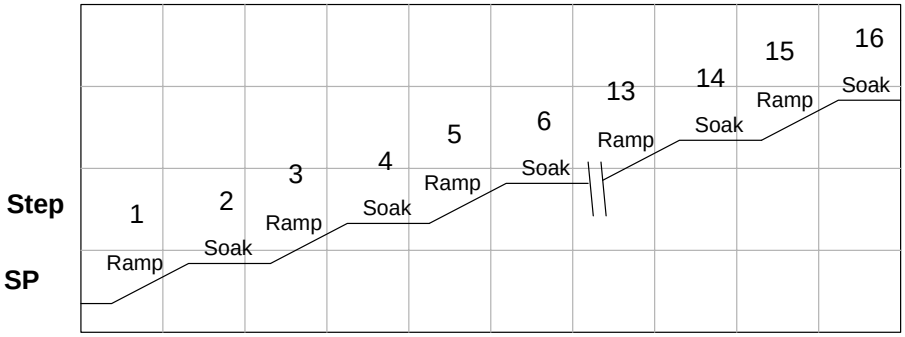
It is instructive to view the ramp/soak generator as a dedicated function to generate SP values, as shown below. It has two categories of inputs which determine the SP values generated. The *ramp/soak table* must be programmed in advance, containing the values that will define the ramp/soak profile. The loop reads from the table during each PID calculation as necessary. The ramp/soak controls are bits in a special loop table word that control the real-time start/stop functionality of the ramp/soak generator. The ladder program can monitor the status of the ramp soak profile (current ramp/segment number).



Now that we have described the general ramp/soak generator operation, we list its specific features:

- Each loop has its own ramp/soak generator (use is optional).
- You may specify up to eight ramp/soak steps (16 segments).
- The ramp soak generator can run anytime the PLC is in Run mode. Its operation is independent of the loop mode (Manual or Auto).
- Ramp/soak real-time controls include Start, Hold, Resume, and Jog.
- Ramp/soak monitoring includes Profile Complete, Soak Deviation (SP minus PV), and current ramp/soak step number.

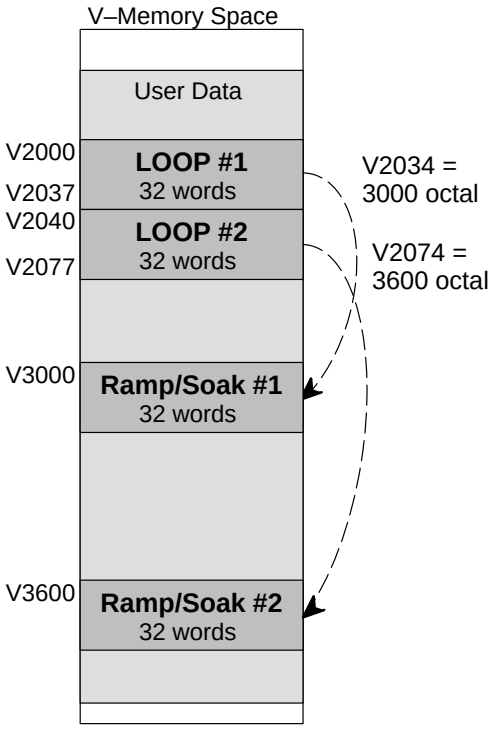
The following figure shows a SP profile consisting of ramp/soak segment pairs. The segments are individually numbered as steps from 1 to 16. The slope of each of the ramp may be either increasing or decreasing. The ramp/soak generator automatically knows whether to increase or decrease the SP based on the relative values of a ramp's end points. These values come from the ramp/soak table.



Ramp/Soak Table

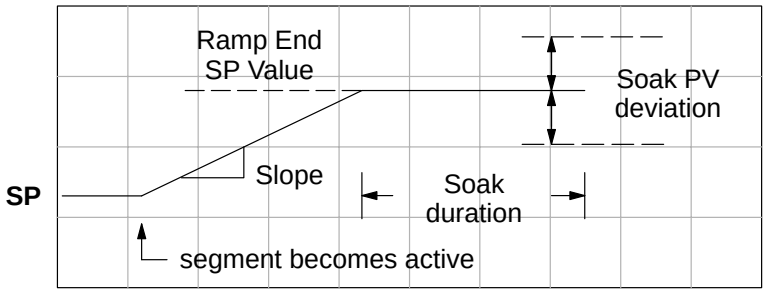
The parameters which define the ramp/soak profile for a loop are in a ramp/soak table. Each loop may have its own ramp/soak table, but it is optional. Recall the Loop Parameter table consists a 32-word block of memory for each loop, and together they occupy one contiguous memory area. However, the ramp/soak table for a loop is individually located, because it is optional for each loop. An address pointer in location V+34 in the loop table specifies the starting location of the ramp/soak table.

In the example to the right, the loop parameter tables for Loop #1 and #2 occupy contiguous 32-word blocks as shown. Each has a pointer to its ramp/soak table, independently located elsewhere in user V-memory. Of course, you may locate all the tables in one group, as long as they do not overlap.



The parameters in the ramp/soak table must be user-defined. the most convenient way is to use ***Direct***SOFT, which features a special editor for this table. Four parameters are required to define a ramp and soak segment pair, as pictured below.

- **Ramp End Value** – specifies the destination SP value for the end of the ramp. Use the same data format for this number as you use for the SP. It may be above or below the beginning SP value, so the slope could be up or down (we don't have to know the starting SP value for ramp #1).
- **Ramp Slope** – specifies the SP increase in counts (units) per second. It is a BCD number from 00.00 to 99.99 (uses implied decimal point).
- **Soak Duration** – specifies the time for the soak segment in minutes, ranging from 000.1 to 999.9 minutes in BCD (implied decimal point).
- **Soak PV Deviation** – (optional) specifies an allowable PV deviation above and below the SP value during the soak period. A PV deviation alarm status bit is generated by the ramp/soak generator.



Ramp/Soak Table

V+00	XXXX	Ramp End SP Value
V+01	XXXX	Ramp Slope
V+02	XXXX	Soak Duration
V+03	XXXX	Soak PV Deviation

The ramp segment becomes active when the previous soak segment ends. If the ramp is the first segment, it becomes active when the ramp/soak generator is started, and automatically assumes the present SP as the starting SP.

Offset	Step	Description	Offset	Step	Description
+ 00	1	Ramp End SP Value	+ 20	9	Ramp End SP Value
+ 01	1	Ramp Slope	+ 21	9	Ramp Slope
+ 02	2	Soak Duration	+ 22	10	Soak Duration
+ 03	2	Soak PV Deviation	+ 23	10	Soak PV Deviation
+ 04	3	Ramp End SP Value	+ 24	11	Ramp End SP Value
+ 05	3	Ramp Slope	+ 25	11	Ramp Slope
+ 06	4	Soak Duration	+ 26	12	Soak Duration
+ 07	4	Soak PV Deviation	+ 27	12	Soak PV Deviation
+ 10	5	Ramp End SP Value	+ 30	13	Ramp End SP Value
+ 11	5	Ramp Slope	+ 31	13	Ramp Slope
+ 12	6	Soak Duration	+ 32	14	Soak Duration
+ 13	6	Soak PV Deviation	+ 33	14	Soak PV Deviation
+ 14	7	Ramp End SP Value	+ 34	15	Ramp End SP Value
+ 15	7	Ramp Slope	+ 35	15	Ramp Slope
+ 16	8	Soak Duration	+ 36	16	Soak Duration
+ 17	8	Soak PV Deviation	+ 37	16	Soak PV Deviation

Many applications do not require all 16 R/S steps. Use all zeros in the table for unused steps. The R/S generator ends the profile when it finds ramp slope=0.

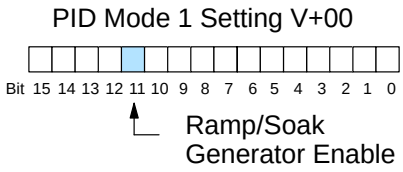
Ramp / Soak Table Flags

The individual bit definitions of the Ramp / Soak Table Flag (Addr+33) word is listed in the following table.

Bit	Ramp / Soak Flag Bit Description	Read/Write	Bit=0	Bit=1
0	Start Ramp / Soak Profile	write	–	0→1 Start
1	Hold Ramp / Soak Profile	write	–	0→1 Hold
2	Resume Ramp / soak Profile	write	–	0→1 Resume
3	Jog Ramp / Soak Profile	write	–	0→1 Jog
4	Ramp / Soak Profile Complete	read	–	Complete
5	PV Input Ramp / Soak Deviation	read	Off	On
6	Ramp / Soak Profile in Hold	read	Off	On
7	Reserved	read	–	–
8–15	Current Step in R/S Profile	read	decode as byte (hex)	

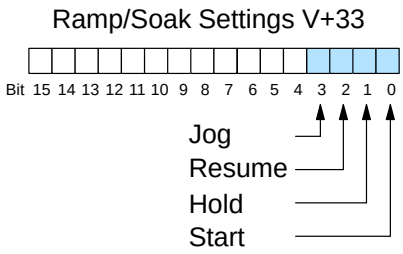
Ramp/Soak Generator Enable

The main enable control to permit ramp/soak generation of the SP value is accomplished with bit 11 in the PID Mode 1 Setting V+00 word, as shown to the right. The other ramp/soak controls in V+33 shown in the table above will not operate unless this bit=1 during the entire ramp/soak process.



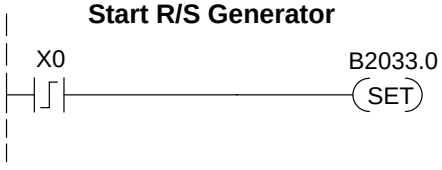
Ramp/Soak Controls

The four main controls for the ramp/soak generator are in bits 0 to 3 of the ramp/soak settings word in the loop parameter table. ***Direct***SOFT controls these bits directly from the ramp/soak settings dialog. However, you must use ladder logic to control these bits during program execution. We recommend using the bit-of-word instructions.



Ladder logic must set a control bit to a “1” to command the corresponding function. When the loop controller reads the ramp/soak value, it automatically turns off the bit for you. Therefore, a reset of the bit is not required, when the CPU is in Run Mode.

The example program rung to the right shows how an external switch X0 can turn on, and the PD contact uses the leading edge to set the proper control bit to start the ramp soak profile. This uses the Set Bit-of-word instruction.



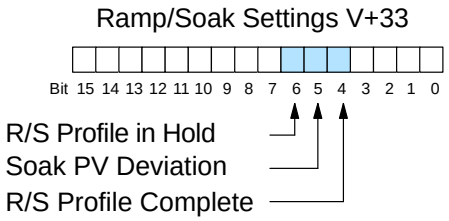
The normal state for the ramp/soak control bits is all zeros. Ladder logic must set only one control bit at a time.

- **Start** – a 0-to-1 transition will start the ramp soak profile. The CPU must be in Run Mode, and the loop can be in Manual or Auto Mode. If the profile is not interrupted by a Hold or Jog command, it finishes normally.
- **Hold** – a 0-to-1 transition will stop the ramp/soak profile in its current state, and the SP value will be frozen.
- **Resume** – a 0-to-1 transition cause the ramp/soak generator to resume operation if it is in the hold state. The SP values will resume from their previous value.
- **Jog** – a 0-to-1 transition will cause the ramp/soak generator to truncate the current segment (step), and go to the next segment.

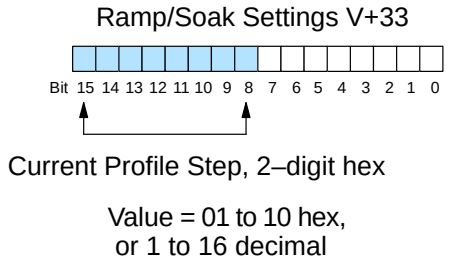
Ramp/Soak Profile Monitoring

You can monitor the Ramp/Soak profile status using other bits in the Ramp/Soak Settings V+33 word, shown to the right.

- R/S Profile Complete – =1 when the last programmed step is done.
- Soak PV Deviation – =1 when the error (SP–PV) exceeds the specified deviation in the R/S table.
- R/S Profile in Hold – =1 when the profile was active but is now in hold.

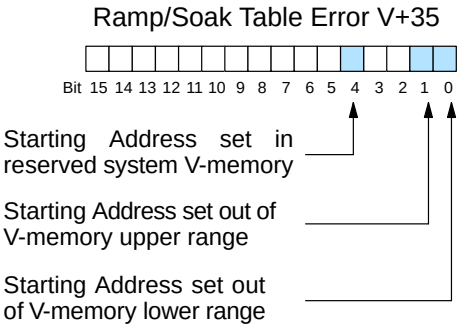


The number of the current step is available in the upper 8 bits of the Ramp/Soak Settings V+33 word. The bits represent a 2-digit hex number, ranging from 1 to 10. Ladder logic can monitor these to synchronize other parts of the program with the ramp/soak profile. Load this word to the accumulator and shift right 8 bits, and you have the step number.



Ramp/Soak Programming Errors

The starting address for the ramp/soak table must be a valid location. If the address points outside the range of user V-memory, one of the bits to the right will turn on when the ramp/soak generator is started. We recommend using ***Direct***SOFT to configure the ramp/soak table. It automatically range checks the addresses for you.



Testing Your Ramp/Soak Profile

It's a good idea to test your ramp/soak profile before using it to control the process. This is easy to do, because the ramp/soak generator will run even when the loop is in Manual Mode. Using ***Direct***SOFT's PID View will be a real time-saver, because it will draw the profile on-screen for you. Be sure to set the trending timebase slow enough to display completed ramp-soak segment pairs in the waveform window.

Troubleshooting Tips

Q. The loop will not go into Automatic Mode.

A. Check the following for possible causes:

- A PV alarm exists, or a PV alarm programming error exists.
- CPU is in Program Mode and Loop mode is depedent on CPU mode.
- The loop is the major loop of a cascaded pair, and the minor loop is not in Cascade Mode.

Q. The Control Output stays at zero constantly when the loop is in Automatic Mode.

A. Check the following for possible causes:

- The Control Output upper limit in loop table location V+31 is zero.
- All tuning paramters are = 0.
- The loop is driven into saturation, because the error never goes to zero value and changes (algebraic) sign.

Q. The Control Output value is not zero, but it is incorrect.

A. Check the following for possible causes:

- The gain values are entered improperly. Gains are entered in the loop table in BCD, while the SP and PV are in decimal. If you are using ***Direct***SOFT, it displays the SP, PV, Bias and Control output in decimal (BCD), converting it to decimal (binary) before updating the loop table.

Q. The Ramp/Soak Generator does not operate when I activate the Start bit.

A. Check the following for possible causes:

- The Ramp/Soak enable bit is off. Check the status of bit 11 of loop parameter table location V+00. It must be set =1.
- The hold bit or other bits in the Ramp/Soak control are on.
- The beginning SP value and the first ramp ending SP value are the same, so first ramp segment has no slope and consequently has no duration. The ramp/soak generator moves quickly to the soak segment, giving the illusion the first ramp is not working.
- The loop is in Cascade Mode, and is trying to get the SP remotely.
- The SP upper limit value in the loop table location V+27 is too low or the upper limit value in V+26 is too high.
- Check your ladder program to verify it is not writing to the SP location (V+02 in the loop table). A quick way to do this is to temporarily place an end coil at the beginning of your program, then go to PLC Run Mode, and manually start the ramp/soak generator.

Q. The PV value in the table is constant, even though the analog module receives the PV signal.

A. Your ladder program must read the analog value from the module successfully and write it into the loop table V+03 location. Verify the analog module is generating the value, and the ladder is working.

Q. The Derivative gain doesn't seem to have any affect on the output.

A. The derivative limit is probably enabled (see section on derivative gain limiting).

Q. The loop Setpoint appears to be changing by itself.

A. Check the following for possible causes:

- The Ramp/Soak generator is enabled, and is generating setpoints.
- If this symptom occurs on loop Manual-to-Auto Mode changes, the loop automatically sets the SP=PV (bumpless transfer feature).
- Check your ladder program to verify it is not writing to the SP location (V+02 in the loop table). A quick way to do this is to temporarily place an end coil at the beginning of your program, then go to PLC Run Mode.

Q. The SP and PV values I enter with *Direct*SOFT work okay, but these values do not work properly when the ladder program writes the data.

A. The PID View in ***Direct***SOFT lets you enter SP, PV, and Bias values in decimal, and displays them in decimal for your convenience. For example, when the data format is 12 bit unipolar, the values range from 0 to 4095. However, the loop table actually requires these in hex, so ***Direct***SOFT converts them for you. The values in the table range from 0 to FFF, for 12-bit unipolar format.

Q. The loop seems unstable and impossible to tune, no matter what I gains I use.

A. Check the following for possible causes:

- The loop sample time is set too long. Refer to the section near the front of this chapter on selecting the loop update time.
- The gains are too high. Start out by reducing the derivative gain to zero. Then reduce the integral gain (remember a small number is a large gain), and the proportional gain if necessary.
- There is too much transfer lag in your process. This means the PV reacts sluggishly to control output changes. There may be too much “distance” between actuator and PV sensor, or the actuator may be weak in its ability to transfer energy into the process.
- There may be a process disturbance that is over-powering the loop. Make sure the PV is relatively steady when the SP is not changing.

Bibliography

Fundamentals of Process Control Theory, Second Edition Author: Paul W. Murrill Publisher: Instrument Society of America ISBN 1–55617–297–4	Application Concepts of Process Control Author: Paul W. Murrill Publisher: Instrument Society of America ISBN 1–55617–080–7
PID Controllers: Theory, Design, and Tuning, 2nd Edition Author: K. Astrom and T Hagglund Publisher: Instrument Society of America ISBN 1–55617–516–7	Fundamentals of Temperature, Pressure, and Flow Measurements, Third edition Author: Robert P. Benedict Publisher: John Wiley and Sons ISBN 0–471–89383–8
Process / Industrial Instruments & Controls Handbook, Fourth Edition Author (Editor-in-Chief): Douglas M. Considine Publisher: McGraw-Hill, Inc. ISBN 0–07–012445–0	pH Measurement and Control, Second Edition Author: Gregory K. McMillan Publisher: Instrument Society of America ISBN 1–55617–483–7
Process Control, Third Edition Instrument Engineer’s Handbook Author (Editor-in-Chief): Bela G. Liptak Publisher: Chilton ISBN 0–8019–8242–1	Process Measurement and Analysis, Third Edition Instrument Engineer’s Handbook Author (Editor-in-Chief): Bela G. Liptak Publisher: Chilton ISBN 0–8019–8197–2

Glossary of PID Loop Terminology

Automatic Mode	An operational mode of a loop, in which it makes PID calculations and updates the loop's control output.
Bias Freeze	A method of preserving the bias value (operating point) for a control output, by inhibiting the integrator when the output goes out-of-range. The benefit is a faster loop recovery.
Bias Term	In the position form of the PID equation, it is the sum of the integrator and the initial control output value.
Bumpless Transfer	A method of changing the operation mode of a loop while avoiding the usual sudden change in control output level. This consequence is avoided by artificially making the SP and PV equal, or the bias term and control output equal at the moment of mode change.
Cascaded Loops	A cascaded loop receives its setpoint from the output of another loop. Cascaded loops have a major/minor relationship, and work together to ultimately control one PV.
Cascade Mode	An operational mode of a loop, in which it receives its SP from another loop's output.
Continuous Control	Control of a process done by delivering a smooth (analog) signal as the control output.
Direct-Acting Loop	A loop in which the PV increases in response to a control output increase. In other words, the process has a positive gain.
Error	The difference in value between the SP and PV, Error=SP – PV
Error Deadband	An optional feature which makes the loop insensitive to errors when they are small. You can specify the size of the deadband.
Error Squared	An optional feature which multiplies the error by itself, but retains the original algebraic sign. It reduces the effect of small errors, while magnifying the effect of large errors.
Feedforward	A method of optimizing the control response of a loop when a change in setpoint or disturbance offset is known and has a quantifiable effect on the bias term.
Control Output	The numerical result of a PID equation which is sent by the loop with the intention of nulling out the current error.
Derivative Gain	A constant that determines the magnitude of the PID derivative term in response to the current error.
Integral Gain	A constant that determines the magnitude of the PID integral term in response to the current error.
Major Loop	In cascade control, it is the loop that generates a setpoint for the cascaded loop.
Manual Mode	An operational mode of a loop, it which the PID calculations are stopped. The operator must manually control the loop by writing to the control output value directly.
Minor Loop	In cascade control, the minor loop is the subordinate loop that receives its SP from the major loop.
On / Off Control	A simple method of controlling a process, through on/off application of energy into the system. The mass of the process averages the on/off effect for a relatively smooth PV. A simple ladder program can convert the DL350's continuous loop output to on/off control.
PID Loop	A mathematical method of closed-loop control involving the sum of three terms based on proportional, integral, and derivative error values. The three terms have independent gain constants, allowing one to optimize (tune) the loop for a particular physical system.
Position Algorithm	The control output is calculated so it responds to the displacement (position) of the PV from the SP (error term)
Process	A manufacturing procedure which adds value to raw materials. Process control particularly refers to inducing *chemical* changes to the material in process.
Process Variable (PV)	A quantitative measurement of a physical property of the material in process, which affects final product quality and is important to monitor and control.

Proportional Gain	A constant that determines the magnitude of the PID proportional term in response to the current error.
PV Absolute Alarm	A programmable alarm that compares the PV value to alarm threshold values.
PV Deviation Alarm	A programmable alarm that compares the difference between the SP and PV values to a deviation threshold value.
Ramp / Soak Profile	A set of SP values called a profile, which is generated in real time upon each loop calculation. The profile consists of a series of ramp and soak segment pairs, greatly simplifying the task of programming the PLC to generate such SP sequences.
Rate	Also called differentiator, the rate term responds to the *changes* in the error term.
Remote Setpoint	The location where a loop reads its setpoint when it is configured as the minor loop in a cascaded loop topology.
Reset	Also called integrator, the reset term adds each sampled error to the previous, maintaining a running total called the bias.
Reset Windup	A condition created when the loop is unable to find equilibrium, and the persistent error causes the integrator (reset) sum to grow excessively (windup). Reset windup causes an extra recovery delay when the original loop fault is remedied.
Reverse-Acting Loop	A loop in which the PV increases in response to a control output decrease. In other words, the process has a negative gain.
Sampling time	The time between PID calculations. The CPU method of process control is called a sampling controller, because it samples the SP and PV only periodically.
Setpoint (SP)	The desired value for the process variable. The setpoint (SP) is the input command to the loop controller during closed loop operation.
Soak Deviation	The soak deviation is a measure of the difference between the SP and PV during a soak segment of the Ramp / Soak profile, when the Ramp / Soak generator is active.
Step Response	The behavior of the process variable in response to a step change in the SP (in closed loop operation), or a step change in the control output (in open loop operation)
Transfer	To change from one loop operational mode to another (between Manual, Auto, or Cascade). The word “transfer” probably refers to the transfer of control of the control output or the SP, depending on the particular mode change.
Velocity Algorithm	The control output is calculated to represent the rate of change (velocity) for the PV to become equal to the SP.

9

Maintenance and Troubleshooting

In This Chapter. . . .

- Hardware Maintenance
- Diagnostics
- CPU Indicators
- PWR Indicator
- RUN Indicator
- CPU Indicator
- BATT Indicator
- Communications Problems
- I/O Module Troubleshooting
- Noise Troubleshooting
- Machine Startup and Program Troubleshooting

Hardware Maintenance

Standard Maintenance

The DL305 is a low maintenance system requiring only a few periodic checks to help reduce the risks of problems. Routine maintenance checks should be made regarding two key items.

- Air quality (cabinet temperature, airflow, etc.)
- CPU battery

Air Quality Maintenance

The quality of the air your system is exposed to can affect system performance. If you have placed your system in an enclosure, check to see the ambient temperature is not exceeding the operating specifications. If there are filters in the enclosure, clean or replace them as necessary to ensure adequate airflow. A good rule of thumb is to check your system environment every one to two months. Make sure the DL305 is operating within the system operating specifications.

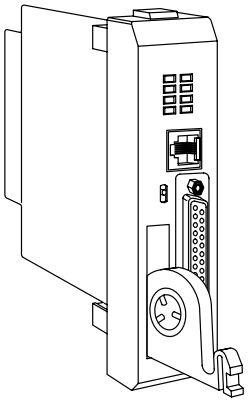
Low Battery Indicator

The CPU has a battery LED that indicates the battery voltage is low. You should check this indicator periodically to determine if the battery needs replacing. You can also detect low battery voltage from within the CPU program. SP43 is a special relay that comes on when the battery needs to be replaced.

CPU Battery Replacement

The CPU battery is used to retain program V memory and the system parameters. The life expectancy of this battery is five years.

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 D3–BAT–1 CPU battery in the DL350 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.
3. Close the battery door making sure that it locks securely in place.
4. Make a note of the date the battery was installed.

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

Diagnostics

Diagnostics

Your DL305 system performs many pre-defined diagnostic routines with every CPU scan. The diagnostics have been designed to detect various types of failures for the CPU and I/O modules. There are two primary error classes, fatal and non-fatal.

Fatal Errors

Fatal errors are errors the CPU has detected that offer a risk of the system not functioning safely or properly. If the CPU is in Run Mode when the fatal error occurs, the CPU will switch to Program Mode. (Remember, in Program Mode all outputs are turned off.) If the fatal error is detected while the CPU is in Program Mode, the CPU will not enter Run Mode until the error has been corrected.

Here are some examples of fatal errors.

- Base power supply failure
- Parity error or CPU malfunction
- I/O configuration errors
- Certain programming errors

Non-fatal Errors

Non-fatal errors are errors that are flagged by the CPU as requiring attention. They can neither cause the CPU to change from Run Mode to Program Mode, nor do they prevent the CPU from entering Run Mode. There are special relays the application program can use to detect if a non-fatal error has occurred. The application program can then be used to take the system to an orderly shutdown or to switch the CPU to Program Mode if necessary.

Some examples of non-fatal errors are:

- Backup battery voltage low
- All I/O module errors
- Certain programming errors

Finding Diagnostic Information

Diagnostic information can be found in several places with varying levels of message detail.

- The CPU automatically logs error codes and any FAULT messages into two separate tables which can be viewed with the Handheld or ***Direct***SOFT™.
- The handheld programmer displays error numbers and short descriptions of the error.
- ***Direct***SOFT provides the error number and an error message.
- Appendix B in this manual has a complete list of error messages sorted by error number.

Many of these messages point to supplemental memory locations which can be referenced for additional related information. These memory references are in the form of V-memory and SPs (special relays).

The following two tables name the specific memory locations that correspond to certain types of error messages. The special relay table also includes status indicators which can be used in programming. For a more detailed description of each of these special relays refer to Appendix D.

V-memory Locations Corresponding to Error Codes

Error Class	Error Category	Diagnostic Vmemory
User-Defined	Error code used with FAULT instruction	V7751
System Error	Fatal Error code	V7755
	Major Error code	V7756
	Minor Error code	V7757
Grammatical	Address where syntax error occurs	V7763
	Error Code found during syntax check	V7764
CPU Scan	Number of scans since last Program to Run Mode transition	V7765
	Current scan time (ms)	V7775
	Minimum scan time (ms)	V7776
	Maximum scan time (ms)	V7777

Special Relays (SP) Corresponding to Error Codes

Startup and Real-time Relays	
SP0	On first scan only
SP1	Always ON
SP3	1 minute clock
SP4	1 second clock
SP5	100 millisecond clock
SP6	50 millisecond clock
SP7	On alternate scans
CPU Status Relays	
SP11	Forced run mode
SP12	Terminal run mode
SP13	Test run mode
SP14	Test hold mode
SP15	Test program mode
SP16	Terminal program mode
SP20	STOP instruction was executed
SP21	BREAK instruction was executed
SP22	Interrupt enabled
System Monitoring Relays	
SP40	Critical error
SP41	Non-critical error
SP43	Battery low
SP46	Communications error
SP47	I/O configuration error
SP50	Fault instruction was executed
SP51	Watchdog timeout
SP52	Syntax error
SP53	Cannot solve the logic
SP54	Intelligent module communication error

Accumulator Status Relays	
SP60	Acc. is less than value
SP61	Acc. is equal to value
SP62	Acc. is greater than value
SP63	Acc. result is zero
SP64	Half borrow occurred
SP65	Borrow occurred
SP66	Half carry occurred
SP67	Carry occurred
SP70	Result is negative (sign)
SP71	Pointer reference error
SP73	Overflow
SP75	Data is not in BCD
SP76	Load zero
Communication Monitoring Relays	
SP116	Port 2 is communicating with another device
SP117	Communication error on Port 2
SP120	Module busy, Slot 0
SP121	Communication error Slot 0
SP122	Module busy, Slot 1
SP123	Communication error Slot 1
SP124	Module busy, Slot 2
SP125	Communication error Slot 2
SP126	Module busy, Slot 3
SP127	Communication error Slot 3
SP130	Module busy, Slot 4
SP131	Communication error Slot 4
SP132	Module busy, Slot 5
SP133	Communication error Slot 5
SP134	Module busy, Slot 6
SP135	Communication error Slot 6
SP136	Module busy, Slot 7
SP137	Communication error Slot 7

Error Message Tables

The DL350 CPU will automatically log any system error codes and any custom messages you have created in your application program with the FAULT instructions. The CPU logs the error code, the date, and the time the error occurred. There are two separate tables that store this information.

- Error Code Table – the system logs up to 32 errors in the table. When an error occurs, the errors already on the table are pushed down and the most recent error is loaded into the top slot. If the table is full when an error occurs, the oldest error is pushed (erased) from the table.
- Message Table – the system logs up to 16 messages in this table. When a message is triggered, the messages already stored in the table are pushed down and the most recent message is loaded into the top slot. If the table is full when an error occurs, the oldest message is pushed (erased) from the table.

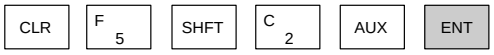
The following diagram shows an example of an error table for messages.

Date	Time	Message
1993–05–26	08:41:51:11	*Conveyor–2 stopped
1993–04–30	17:01:11:56	* Conveyor–1 stopped
1993–04–30	17:01:11:12	* Limit SW1 failed
1993–04–28	03:25:14:31	* Saw Jam Detect

You can access the error code table and the message table through ***Direct***SOFT's PLC Diagnostic sub-menus or from the Handheld Programmer. Details on how to access these logs are provided in the ***Direct***SOFT and D2–HPP manual.

The following examples show you how to use the Handheld and AUX Function 5C to show the error codes. The most recent error or message is always displayed. You can use the PREV and NXT keys to scroll through the messages.

Use AUX 5C to view the tables




```
AUX 5C HISTORY D
    ERROR/MESAGE
```

Use the arrow key to select Errors or Messages


```
AUX 5C HISTORY D
    ERROR/MESAGE
```

Example of an error display

```
E252NEW I/O CFG
93/09/21 10:11:15
```

Year Month Day Time

System Error Codes

The System error log contains 32 of the most recent errors that have been detected. The errors that are trapped in the error log are a subset of all the error messages which the DL305 systems generate. These errors can be generated by the CPU or by the Handheld Programmer, depending on the actual error. Appendix B provides a more complete description of the error codes.

The errors can be detected at various times. However, most of them are detected at power-up, on entry to Run Mode, or when a Handheld Programmer key sequence results in an error or an illegal request.

Error Code	Description
E003	Software time-out
E004	Invalid instruction (RAM parity error)
E041	CPU battery low
E043	Memory cartridge battery low
E099	Program memory exceeded
E101	CPU memory cartridge missing
E104	Write fail
E151	Invalid command
E155	RAM failure
E201	Terminal block missing
E202	Missing I/O module
E203	Blown fuse
E206	User 24V power supply failure
E210	Power fault
E250	Communication failure in the I/O chain
E251	I/O parity error
E252	New I/O configuration
E262	I/O out of range
E312	Communications error 2
E313	Communications error 3
E316	Communications error 6
E320	Time out
E321	Communications error
E499	Invalid Text entry for Print Instruction
E501	Bad entry
E502	Bad address
E503	Bad command
E504	Bad reference / value
E505	Invalid instruction
E506	Invalid operation

Error Code	Description
E520	Bad operation – CPU in Run
E521	Bad operation – CPU in Test Run
E523	Bad operation – CPU in Test Program
E524	Bad operation – CPU in Program
E525	Mode switch not in TERM
E526	Unit is offline
E527	Unit is online
E528	CPU mode
E540	CPU locked
E541	Wrong password
E542	Password reset
E601	Memory full
E602	Instruction missing
E604	Reference missing
E610	Bad I/O type
E611	Bad Communications ID
E620	Out of memory
E621	EEPROM Memory not blank
E622	No Handheld Programmer EEPROM
E624	V memory only
E625	Program only
E627	Bad write operation
E628	Memory type error (should be EEPROM)
E640	Miscompare
E650	Handheld Programmer system error
E651	Handheld Programmer ROM error
E652	Handheld Programmer RAM error

Program Error Codes

The following list shows the errors that can occur when there are problems with the program. These errors will be detected when you try to place the CPU into Run Mode, or, when you use AUX 21 – Check Program. The CPU will also turn on SP52 and store the error code in V7755. Appendix B provides a more complete description of the error codes.

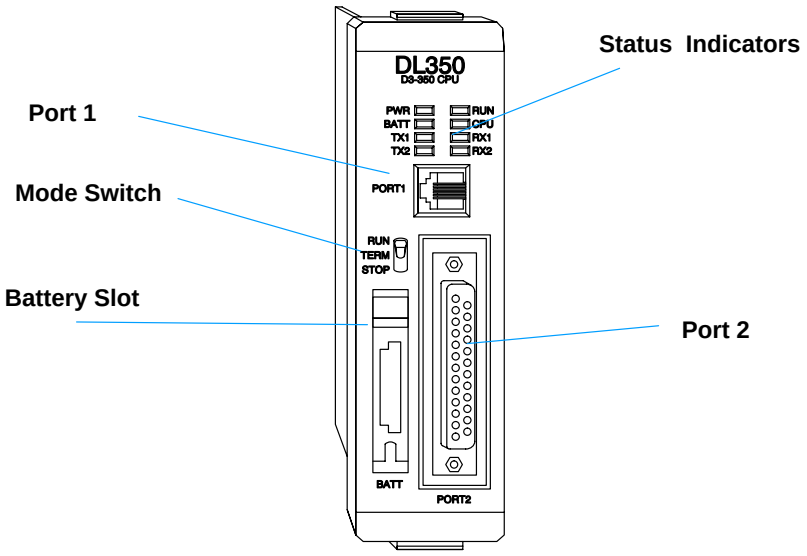
Error Code	Description
E4**	No Program in CPU
E401	Missing END statement
E402	Missing LBL
E403	Missing RET
E404	Missing FOR
E405	Missing NEXT
E406	Missing IRT
E412	SBR/LBL >64
E413	FOR/NEXT >64
E421	Duplicate stage reference
E422	Duplicate SBR/LBL reference
E423	Nested loops
E431	Invalid ISG/SG address
E432	Invalid jump (GOTO) address
E433	Invalid SBR address
E434	Invalid RTC address
E435	Invalid RT address
E436	Invalid INT address
E437	Invalid IRTC address
E438	Invalid IRT address
E440	Invalid Data Address
E441	ACON/NCON
E451	Bad MLS/MLR
E452	X input used as output coil
E453	Missing T/C
E454	Bad TMRA
E455	Bad CNT
E456	Bad SR

Error Code	Description
E461	Stack Overflow
E462	Stack Underflow
E463	Logic Error
E464	Missing Circuit
E471	Duplicate coil reference
E472	Duplicate TMR reference
E473	Duplicate CNT reference
E480	CV position error
E481	CV not connected
E482	CV exceeded
E483	CVJMP placement error
E484	No CV
E485	No CVJMP
E486	BCALL placement error
E487	No Block defined
E488	Block position error
E489	Block CR identifier error
E490	No Block stage
E491	ISG position error
E492	BEND position error
E493	BEND I error
E494	No BEND

CPU Indicators

The DL350 CPU has indicators on the front to help you diagnose problems with the system. The table below gives a quick reference of potential problems associated with each status indicator. Following the table will be a detailed analysis of each of these indicator problems.

Indicator Status	Potential Problems
PWR (off)	1. System voltage incorrect. 2. Power supply/CPU is faulty 3. Other component such an I/O module has power supply shorted 4. Power budget exceeded for the base being used
RUN (will not come on)	1. CPU programming error 2. Switch in TERM position 3. Switch in STOP position
CPU (on)	1. Electrical noise interference 2. CPU defective
BATT (on)	1. CPU battery low 2. CPU battery missing, or disconnected
TX1	1. Transmitting data from Port 1
RX1	1. Receiving data at Port 1
TX2	1. Transmitting data from Port 2
RX2	1. Receiving data at Port 2



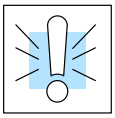
PWR Indicator

There are four general reasons for the CPU power status LED (PWR) to be OFF:

1. Power to the base is incorrect or is not applied.
2. Base power supply is faulty.
3. Other component(s) have the power supply shut down.
4. Power budget for the base has been exceeded.

Incorrect Base Power

If the voltage to the power supply is not correct, the CPU and/or base may not operate properly or may not operate at all. Use the following guidelines to correct the problem.



WARNING: To minimize the risk of electrical shock, always disconnect the system power before inspecting the physical wiring.

1. First, disconnect the system power and check all incoming wiring for loose connections.
2. If you are using a separate termination panel, check those connections to make sure the wiring is connected to the proper location.
3. If the connections are acceptable, reconnect the system power and measure the voltage at the base terminal strip to insure it is within specification. If the voltage is not correct shut down the system and correct the problem.
4. If all wiring is connected correctly and the incoming power is within the specifications required, the base power supply should be returned for repair.

Faulty CPU

There is not a good check to test for a faulty CPU other than substituting a known good one to see if this corrects the problem. If you have experienced major power surges, it is possible the CPU and power supply have been damaged. If you suspect this is the cause of the power supply damage, a line conditioner which removes damaging voltage spikes should be used in the future.

Device or Module causing the Power Supply to Shutdown

It is possible a faulty module or external device using the system 5V can shut down the power supply. This 5V can be coming from the base or from the CPU communication ports.

To test for a device causing this problem:

1. Turn off power to the CPU.
2. Disconnect all external devices (i.e., communication cables) from the CPU.
3. Reapply power to the system.

If the power supply operates normally you may have either a shorted device or a shorted cable. If the power supply does not operate normally then test for a module causing the problem by following the steps below:

If the PWR LED operates normally the problem could be in one of the modules. To isolate which module is causing the problem, disconnect the system power and remove one module at a time until the PWR LED operates normally.

Follow the procedure below:

- Turn off power to the base.
- Remove a module from the base.
- Reapply power to the base.

Bent base connector pins on the module can cause this problem. Check to see the connector is not the problem.

Power Budget Exceeded

If the machine had been operating correctly for a considerable amount of time prior to the indicator going off, the power budget is not likely to be the problem. Power budgeting problems usually occur during system start-up when the PLC is under operation and the inputs/outputs are requiring more current than the base power supply can provide.

WARNING: The PLC may reset if the power budget is exceeded. If there is any doubt about the system power budget please check it at this time. Exceeding the power budget can cause unpredictable results which can cause damage and injury. Verify the modules in the base operate within the power budget for the chosen base. You can find these tables in Chapter 4, System Design and Configuration.

RUN Indicator

If the CPU will not enter the Run mode (the RUN indicator is off), the problem is usually in the application program, unless the CPU has a fatal error. If a fatal error has occurred, the CPU LED should be on. You can use a programming device to determine the cause of the error.

If you are using a DL350 and you are trying to change the modes with a programming device, make sure the mode switch is in the TERM position.

Both of the programming devices, Handheld Programmer and ***Direct***SOFT™, will return a error message describing the problem. Depending on the error, there may also be an AUX function you can use to help diagnose the problem. The most common programming error is “Missing END Statement”. All application programs require an END statement for proper termination. A complete list of error codes can be found in Appendix B.

CPU Indicator

If the CPU indicator is on, a fatal error has occurred in the CPU. Generally, this is not a programming problem but an actual hardware failure. You can power cycle the system to clear the error. If the error clears, you should monitor the system and determine what caused the problem. You will find this problem is sometimes caused by high frequency electrical noise introduced into the CPU from an outside source. Check your system grounding and install electrical noise filters if the grounding is suspected. If power cycling the system does not reset the error, or if the problem returns, you should replace the CPU.

BATT Indicator

If the BATT indicator is on, the CPU battery is either disconnected or needs replacing. The battery voltage is continuously monitored while the system voltage is being supplied.

Communications Problems

If you cannot establish communications with the CPU, check these items.

- The cable is disconnected.
- The cable has a broken wire or has been wired incorrectly.
- The cable is improperly terminated or grounded.
- The device connected is not operating at the correct baud rate (9600 baud for the top port. Use AUX 56 to select the baud rate for the bottom port on a DL350).
- The device connected to the port is sending data incorrectly.
- A grounding difference exists between the two devices.
- Electrical noise is causing intermittent errors
- The CPU has a bad comm port and the CPU should be replaced.
- If you are using ***Direct***SOFT, refer to the troubleshooting section of the Quick Start Manual.

If an error occurs the indicator will come on and stay on until a successful communication has been completed.

I/O Module Troubleshooting

Things to Check

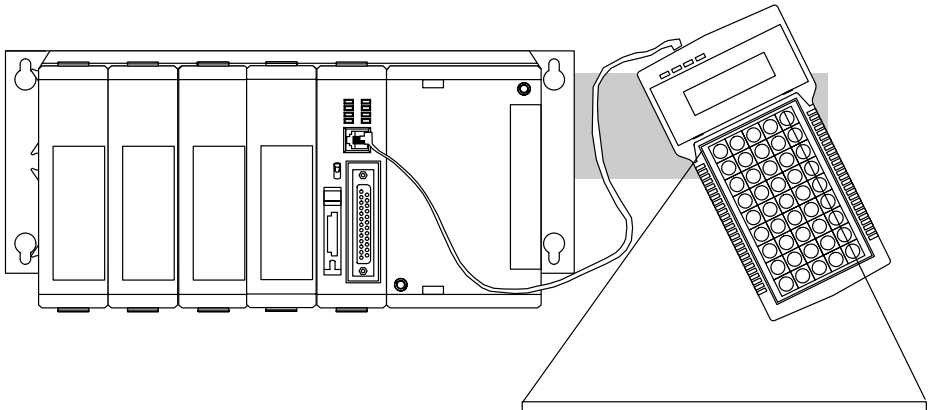
If you suspect an I/O error, there are several things that could be causing the problem.

- A blown fuse
- A loose terminal block
- The 24 VDC supply has failed
- The module has failed
- The I/O configuration check detects a change in the I/O configuration

I/O Diagnostics

If the modules are not providing any clues to the problem, run AUX 42 from the handheld programmer or I/O diagnostics in ***Direct***SOFT. Both options will provide the base number, the slot number and the problem with the module. Once the problem is corrected the indicators will reset.

An I/O error will not cause the CPU to switch from the run to program mode, however there are special relays (SPs) available in the CPU which will allow this error to be read in ladder logic. The application program can then take the required action such as entering the program mode or initiating an orderly shutdown. The following figure shows an example of the failure indicators.



Program Control Information

V7752	0020	Desired module ID code
V7753	0021	Current module ID code
V7754	0002	Location of conflict
V7755	0252	Fatal error code
SP47	⊣⊢	I/O Configuration Error

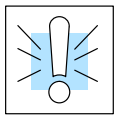
Some Quick Steps When troubleshooting the DL305 series I/O modules there are a few facts you should be aware of. These facts may assist you in quickly correcting an I/O problem.

- The output modules cannot detect shorted or open output points. If you suspect one or more points on a output module to be faulty, you should measure the voltage drop from the common to the suspect point. Remember when using a Digital Volt Meter, leakage current from an output device such as a triac or a transistor must be considered. A point which is off may appear to be on if no load is connected the the point.
- The I/O point status indicators on the modules are logic side indicators. This means the LED which indicates the on or off status reflects the status of the point in respect to the CPU. On an output module the status indicators could be operating normally while the actual output device (transistor, triac etc.) could be damaged. With an input module if the indicator LED is on, the input circuitry should be operating properly. To verify proper functionality check to see the LED goes off when the input signal is removed.
- Leakage current can be a problem when connecting field devices to I/O modules. False input signals can be generated when the leakage current of an output device is great enough to turn on the connected input device. To correct this, install a resistor in parallel with the input or output of the circuit. The value of this resistor will depend on the amount of leakage current and the voltage applied but usually a 10K to 20KΩ resistor will work. Insure the wattage rating of the resistor is correct for your application.
- The easiest method to determine if a module has failed is to replace it if you have a spare. However, if you suspect another device to have caused the failure in the module, that device may cause the same failure in the replacement module as well. As a point of caution, you may want to check devices or power supplies connected to the failed module before replacing it with a spare module.

Testing Output Points

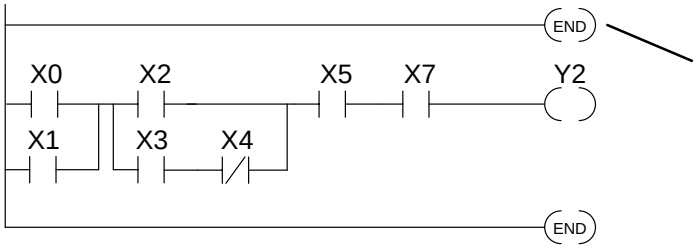
If you want to do an I/O check out independent of the application program, for the DL350 follow the procedure below:

Step	Action
1	Use a handheld programmer or ***Direct***SOFT™ to communicate online to the PLC.
2	Change to Program Mode.
3	Go to address 0.
4	Insert an “END” statement at address 0. (This will cause program execution to occur only at address 0 and prevent the application program from turning the I/O points on or off).
5	Change to Run Mode.
6	Use the programming device to set (turn) on or off the points you wish to test.
7	When you finish testing I/O points delete the “END” statement at address 0.



WARNING: Depending on your application, forcing I/O points may cause unpredictable machine operation that can result in a risk of personal injury or equipment damage. Make sure you have taken all appropriate safety precautions prior to testing any I/O points.

Handheld Programmer Keystrokes Used to Test an Output Point



Insert an END statement at the beginning of the program. This disables the remainder of the program.

From a clear display, use the following keystrokes

STAT ENT

```
16P STATUS
BIT REF   X
```

Use the PREV or NEXT keys to select the Y data type

NEXT A 0 ENT

```
   Y  10   Y   0
□□□□□□□□□□□□□□□□
```

Use arrow keys to select point, then use ON and OFF to change the status

← ← SHFT ON INS

Y2 is now on

```
   Y  10   Y   0
□□□□□□□□□□□□□■□□
```

Noise Troubleshooting

Electrical Noise Problems

Noise is one of the most difficult problems to diagnose. Electrical noise can enter a system in many different ways and fall into one of two categories, conducted or radiated. It may be difficult to determine how the noise is entering the system but the corrective actions for either of the types of noise problems are similar.

- Conducted noise is when the electrical interference is introduced into the system by way of a attached wire, panel connection ,etc. It may enter through an I/O module, a power supply connection, the communication ground connection, or the chassis ground connection.
- Radiated noise is when the electrical interference is introduced into the system without a direct electrical connection, much in the same manner as radio waves.

Reducing Electrical Noise

While electrical noise cannot be eliminated it can be reduced to a level that will not affect the system.

- Most noise problems result from improper grounding of the system. A good earth ground can be the single most effective way to correct noise problems. If a ground is not available, install a ground rod as close to the system as possible. Insure all ground wires are single point grounds and are not daisy chained from one device to another. Ground metal enclosures around the system. A loose wire is no more than a large antenna waiting to introduce noise into the system; therefore, you should tighten all connections in your system. Loose ground wires are more susceptible to noise than the other wires in your system. Review Chapter 2 Installation, Wiring, and Specifications if you have questions regarding how to ground your system.
- Electrical noise can enter the system through the power source for the CPU and I/O. Installing a isolation transformer for all AC sources can correct this problem. DC sources should be well grounded good quality supplies. Switching DC power supplies commonly generate more noise than linear supplies.
- Separate input wiring from output wiring. Never run I/O wiring close to high voltage wiring.

Machine Startup and Program Troubleshooting

The DL350 CPU provides several features to help you debug your program before and during machine startup. This section discusses the following topics which can be very helpful.

- Program Syntax Check
- Duplicate Reference Check
- Test Modes
- Special Instructions
- Run Time Edits
- Forcing I/O Points

Syntax Check

Even though the Handheld Programmer and ***Direct***SOFT provide error checking during program entry, you may want to check a modified program. Both programming devices offer a way to check the program syntax. For example, you can use AUX 21, CHECK PROGRAM to check the program syntax from a Handheld Programmer, or you can use the PLC Diagnostics menu option within ***Direct***SOFT. This check will find a wide variety of programming errors. The following example shows how to use the syntax check with a Handheld Programmer.

Use AUX 21 to perform syntax check

CLR C 2 B 1 AUX ENT

```
AUX 21 CHECK PRO
1:SYN 2:DUP REF
```

Select syntax check (default selection)

ENT (You may not get the busy display if the program is not very long.)

```
BUSY
```

One of two displays will appear

Error Display (example)

```
$00050 E401
MISSING END
```

(shows location in question)

Syntax OK display

```
NO SYNTAX ERROR
?
```

See Appendix B for a complete listing of programming error codes. If you get an error, press CLR and the Handheld will display the instruction where the error occurred. Correct the problem and continue running the Syntax check until the NO SYNTAX ERROR message appears.

Duplicate Reference Check

You can also check for multiple uses of the same output coil. Both programming devices offer a way to check for this condition. For example, you can AUX 21, CHECK PROGRAM to check for duplicate references from a Handheld Programmer, or you can use the PLC Diagnostics menu option within ***Direct***SOFT. The following example shows how to perform the duplicate reference check with a Handheld Programmer.

Use AUX 21 to perform syntax check

CLR | C 2 | B 1 | AUX | ENT

```
AUX 21 CHECK PRO
1:SYN 2:DUP REF
```

Select duplicate reference check

→ | ENT (You may not get the busy display if the program is not very long.)

```
BUSY
```

One of two displays will appear

Error Display (example)

```
$00024 E471
DUP COIL REF
```

(shows location in question)

Syntax OK display

```
NO DUP REFS
?
```

If you get an error, press CLR and the Handheld will display the instruction where the duplicate reference occurred. Correct the problem and continue running the Duplicate Reference check until no duplicate references are found.

NOTE: You can use the same coil in more than one location, especially in programs using the Stage instructions and/or the OROUT instructions. The Duplicate Reference check will find these outputs even though they may be used in an acceptable fashion.

TEST-PGM and TEST-RUN Modes

Test Mode allows the CPU to start in TEST-PGM mode, enter TEST-RUN mode, run a fixed number of scans, and return to TEST-PGM mode. You can select from 1 to 65,525 scans. Test Mode also allows you to maintain output status while you switch between Test-Program and Test-Run Modes. You can select Test Modes from either the Handheld Programmer (by using the MODE key) or from ***Direct***SOFT via a PLC Modes menu option.

The primary benefit of using the TEST mode is to maintain certain outputs and other parameters when the CPU transitions back to Test-program mode. Also, the CPU will maintain timer and counter current values when it switches to TEST-PGM mode.

NOTE: You can only use ***Direct***SOFT™ to specify the number of scans. This feature is not supported on the Handheld Programmer. However, you can use the Handheld to switch between Test Program and Test Run Modes.

With the Handheld, the actual mode entered when you first select Test Mode depends on the mode of operation at the time you make the request. If the CPU is in Run Mode mode, then TEST-RUN is available. If the mode is Program, then TEST-PGM is available. Once you've selected TEST Mode, you can easily switch between TEST-RUN and TEST-PGM. ***Direct***SOFT provides more flexibility in selecting the various modes with different menu options. The following example shows how you can use the Handheld to select the Test Modes.

Use the MODE key to select TEST Modes (example assumes Run Mode)

MODE NEXT ENT

```
*MODE CHANGE*
GO TO T-RUN MODE
```

Press ENT to confirm TEST-RUN Mode

ENT (Note, the TEST LED on the DL205 Handheld indicates the CPU is in TEST Mode.)

```
*MODE CHANGE*
CPU T-RUN
```

You can return to Run Mode, enter Program Mode, or enter TEST-PGM Mode by using the Mode Key

CLR MODE NEXT NEXT ENT

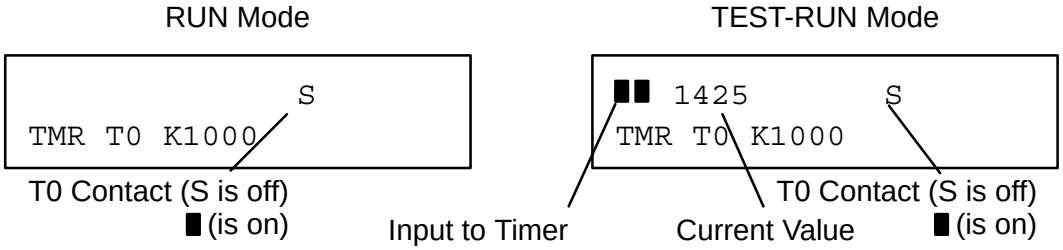
```
*MODE CHANGE*
GO TO T-PGM MODE
```

Press ENT to confirm TEST-PGM Mode

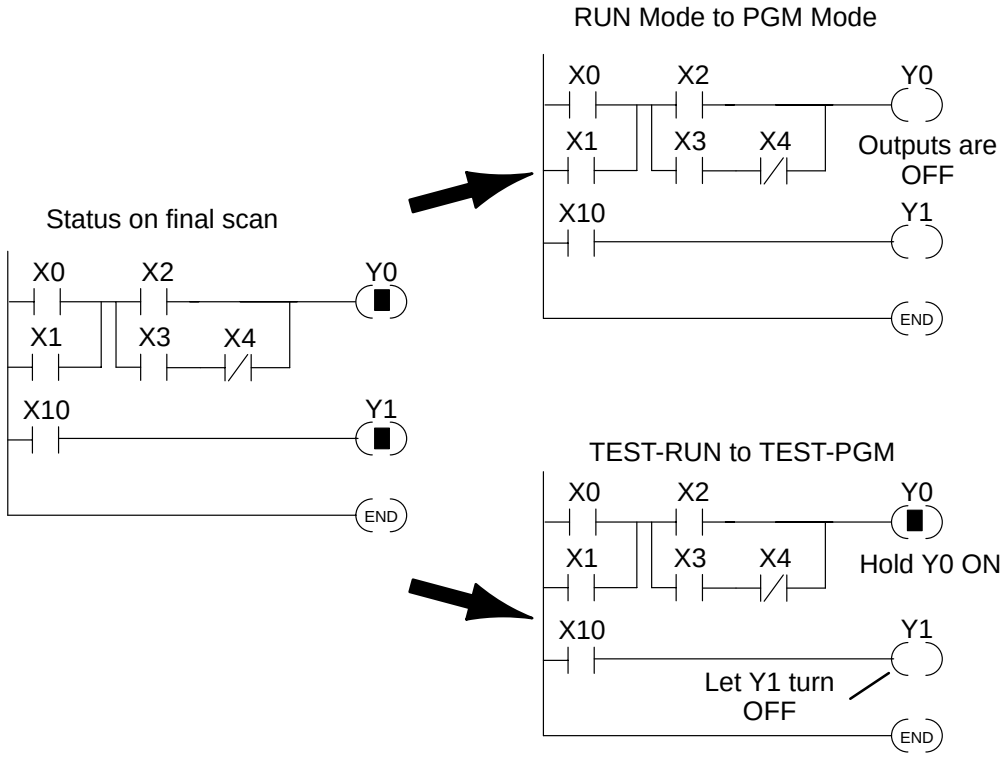
ENT (Note, the TEST LED on the DL205 Handheld indicates the CPU is in TEST Mode.)

```
*MODE CHANGE*
CPU T-PGM
```

Test Displays: With the Handheld Programmer you also have a more detailed display when you use TEST Mode. For some instructions, the TEST-RUN mode display is more detailed than the status displays shown in RUN mode. The following diagram shows an example of a Timer instruction display during TEST-RUN mode.



Holding Output States: The ability to hold output states is very useful, because it allows you to maintain key system I/O points. In some cases you may need to modify the program, but you do not want certain operations to stop. In normal Run Mode, the outputs are turned off when you return to Program Mode. In TEST-RUN mode you can set each individual output to either turn off, or, to hold its last output state on the transition to TEST-PGM mode. This feature is available via a menu option within ***Direct***SOFT™. The following diagram shows the differences between RUN and TEST-RUN modes.



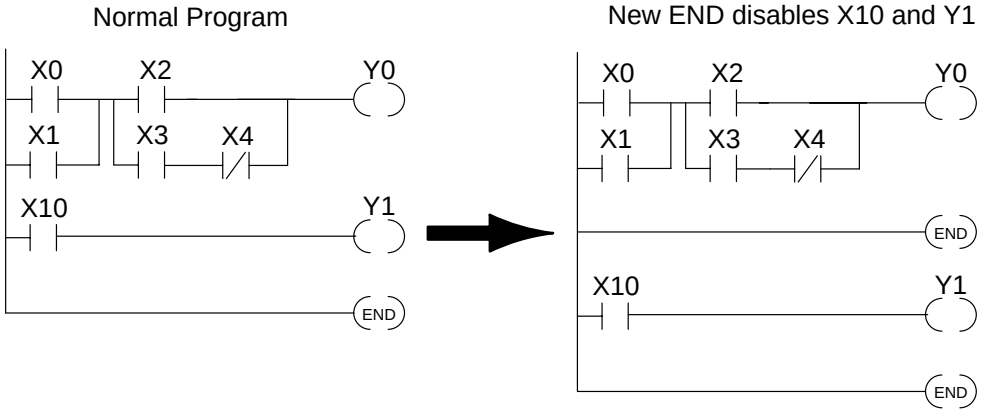
Before you decide that Test Mode is the perfect choice, remember the DL350 CPU also allows you to edit the program during Run Mode. The primary difference between the Test Modes and the Run Time Edit feature is you do not have to configure each individual I/O point to hold the output status. When you use Run Time Edits, the CPU automatically maintains all outputs in their current states while the program is being updated.

Special Instructions

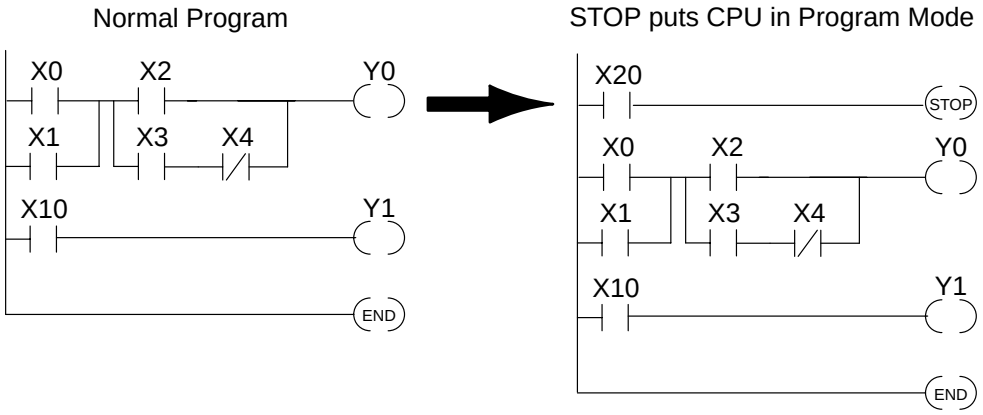
There are several instructions that can be used to help you debug your program during machine startup operations.

- END
- PAUSE
- STOP

END Instruction: If you need a way to quickly disable part of the program, insert an END statement prior to the portion that should be disabled. When the CPU encounters the END statement, it assumes it is the end of the program. The following diagram shows an example.



STOP Instruction: Sometimes during machine startup you need a way to quickly turn off all the outputs and return to Program Mode. In addition to using the Test Modes, you can also use the STOP instruction. When this instruction is executed the CPU automatically exits Run Mode and enters Program Mode. Remember, all outputs are turned off during Program Mode. The following diagram shows an example of a condition that returns the CPU to Program Mode.



In the example shown above, you could trigger X20 which would execute the STOP instruction. The CPU would enter Program Mode and all outputs would be turned off.

Run Time Edits

The DL350 CPU allows you to make changes to the application program during Run Mode. These edits are not “bumpless.” Instead, CPU scan is momentarily interrupted (and the outputs are maintained in their current state) until the program change is complete. This means if the output is off, it will remain off until the program change is complete. If the output is on, it will remain on.

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. There are some important operations sequence changes during Run Time Edits.

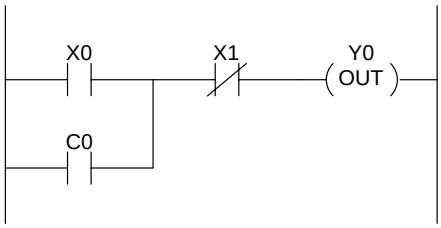
1. If there is a syntax error in the new instruction, the CPU *will not* enter the Run Mode.
2. If you delete an output coil reference and the output was on at the time, the output will remain on until it is forced off with a programming device.
3. Input point changes are not acknowledged during Run Time Edits. So, if you’re using a high-speed operation and a critical input comes on, the CPU may not see the change.

Not all instructions can be edited during a Run Time Edit session. The following list shows the instructions that can be edited.

Mnemonic	Description
TMR	Timer
TMRF	Fast timer
TMRA	Accumulating timer
TMRAF	Accumulating fast timer
CNT	Counter
UDC	Up / Down counter
SGCNT	Stage counter
STR, STRN	Store, Store not
AND, ANDN	And, And not
OR, ORN	Or, Or not
STRE, STRNE	Store equal, Store not equal
ANDE, ANDNE	And equal, And not equal
ORE, ORNE	Or equal, Or not equal
STR, STRN	Store greater than or equal Store less than
AND, ANDN	And greater than or equal And less than

Mnemonic	Description
OR, ORN	Or greater than or equal Or less than
LD	Load data (constant)
LDD	Load data double (constant)
ADDD	Add data double (constant)
SUBD	Subtract data double (constant)
MUL	Multiply (constant)
DIV	Divide (constant)
CMPD	Compare accumulator (constant)
ANDD	And accumulator (constant)
ORD	Or accumulator (constant)
XORD	Exclusive or accumulator (constant)
LDF	Load discrete points to accumulator
OUTF	Output accumulator to discrete points
SHFR	Shift accumulator right
SHFL	Shift accumulator left
NCON	Numeric constant

Use the program logic shown to describe how this process works. In the example, change X0 to C10. Note, the example assumes you have already placed the CPU in Run Mode.



Use the MODE key to select Run Time Edits

MODE NEXT NEXT ENT

```
*MODE CHANGE*
RUN TIME EDIT?
```

Press ENT to confirm the Run Time Edits

ENT (Note, the RUN LED on the DL205 Handheld starts flashing to indicate Run Time Edits are enabled.)

```
*MODE CHANGE*
RUNTIME EDITS
```

Find the instruction you want to change (X0)

SHFT X SET A 0 SHFT FD REF FIND

```
$00000 STR X0
```

Press the arrow key to move to the X. Then enter the new contact (C10).

→ → SHFT C 2 B 1 A 0 ENT

```
RUNTIME EDIT?
STR C10
```

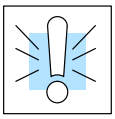
Press ENT to confirm the change

ENT (Note, once you press ENT, the next address is displayed.

```
OR C0
```

Forcing I/O Points

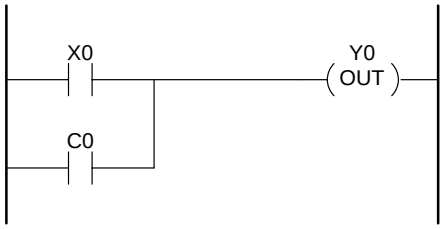
There are many times, especially during machine startup and troubleshooting, where you need the capability to force an I/O point to be either on or off. Before you use a programming device to force any data type, it is important to understand how the DL350 CPU processes the forcing requests.



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.

- **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.

The following diagrams show a brief example of how you could use the Handheld Programmer to force an I/O point. The image register will not be updated with the status from the input module. Also, the solution from the application program will not be used to update the output image register. The example assumes you have already placed the CPU into Run Mode.



From a clear display, use the following keystrokes

STAT ENT

```
16P STATUS
BIT REF   X
```

Use the PREV or NEXT keys to select the Y data type. (Once the Y appears, press 0 to start at Y0.)

NEXT A 0 ENT

```
   Y   10          Y    0
□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
```

Use arrow keys to select point, then use ON and OFF to change the status

← ← SHFT ON INS

Y2 is now on

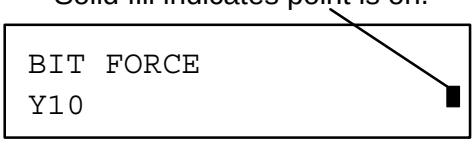
```
   Y   10          Y    0
□ □ □ □ □ □ □ □ □ □ □ □ □ ■ □ □
```

Regular Forcing with Direct Access

From a clear display, use the following keystrokes to force Y10 ON

SHFT | Y MLS | B 1 | A 0 | SHFT | ON INS

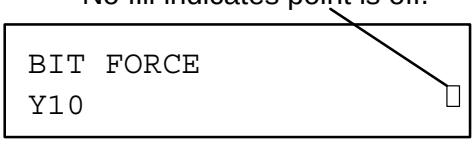
Solid fill indicates point is on.



From a clear display, use the following keystrokes to force Y10 OFF

SHFT | Y MLS | B 1 | A 0 | SHFT | OFF DEL

No fill indicates point is off.



Auxiliary Functions

A

In This Appendix. . . .

Introduction

What are 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. You can access the AUX Functions from ***Direct***SOFT™ or from the DL205 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. Even though this Appendix provides many examples of how the AUX functions operate, you should supplement this information with the documentation for your choice of programming device. Note, the Handheld Programmer may have additional AUX functions that are not supported with the DL350 CPU.

AUX Function and Description		350	HPP
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	×	–
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 CPU Network Address	✓	–
57	Set Retentive Ranges	✓	–
58	Test Operations	×	–
59	Bit Override	×	–
5B	Counter Interface Config.	×	–
5C	Display Error History	✓	–

AUX Function and Description		350	HPP
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	✓	–

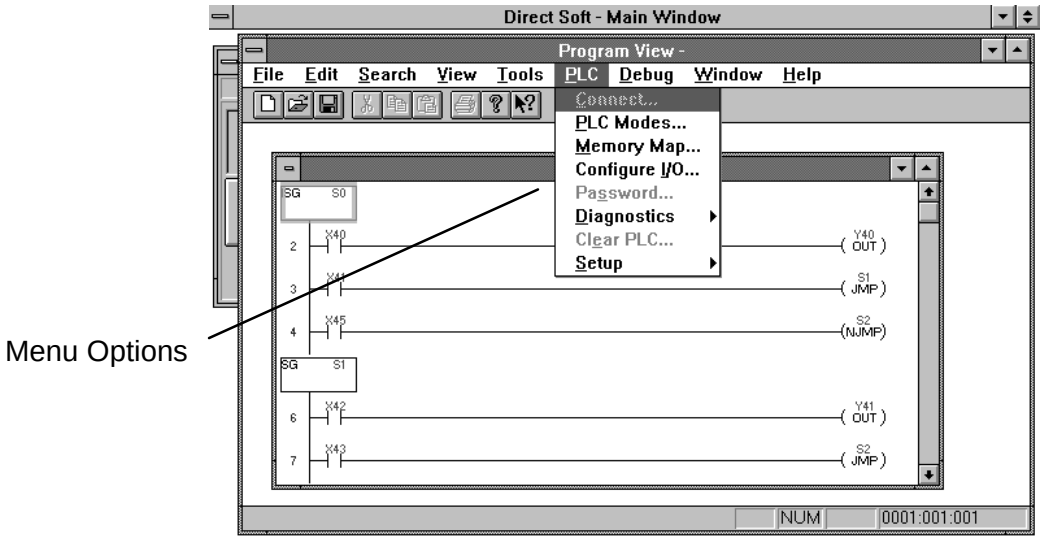
✓ supported

× not supported

– not applicable

Accessing AUX Functions via *Direct*SOFT

DirectSOFT™ provides various menu options during both online and offline programming. Some of the AUX functions are only available during online programming, some only during offline programming, and some during both online and offline programming. The following diagram shows and example of the PLC operations menu available within ***Direct***SOFT.



Accessing AUX Functions via the Handheld Programmer

You can also access the AUX functions by using a Handheld Programmer. Plus, remember some of the AUX functions are only available from the Handheld. Sometimes the AUX name or description cannot fit on one display. If you want to see the complete description, press the arrow keys to scroll left and right. Also, depending on the current display, you may have to press CLR more than once.

CLR AUX

```
AUX FUNCTION SELECTION
AUX 2* RLL OPERATIONS
```

Use NXT or PREV to cycle through the menus

NEXT

```
AUX FUNCTION SELECTION
AUX 3* V OPERATIONS
```

Press ENT to select sub-menus

ENT

```
AUX 3* V OPERATIONS
AUX 31 CLR V MEMORY
```

You can also enter the exact AUX number to go straight to the sub-menu.

Enter the AUX number directly

CLR D 3 B 1 AUX

```
AUX 3* V OPERATIONS
AUX 31 CLR V MEMORY
```

AUX 2* — RLL Operations

AUX 21, 22, 23 and 24

There are four AUX functions available that you can use to perform various operations on the control program.

- AUX 21 — Check Program
- AUX 22 — Change Reference
- AUX 23 — Clear Ladder Range
- AUX 24 — Clear Ladders

AUX 21 Check Program

Both the Handheld and ***Direct***SOFT™ automatically check for errors during program entry. However, there may be occasions when you want to check a program that has already been in the CPU. There are two types of checks available:

- Syntax
- Duplicate References

The Syntax check will find a wide variety of programming errors, such as missing END statements, incomplete FOR/NEXT loops, etc. If you perform this check and get an error, see Appendix B for a complete listing of programming error codes. Correct the problem and then continue running the Syntax check until the message "NO SYNTAX ERROR" appears.

Use the Duplicate Reference check to verify you have not used the same output coil reference more than once. Note, this AUX function will also find the same outputs even if they have been used with the OROUT instruction, which is perfectly acceptable.

This AUX function is available on the PLC Diagnostics sub-menu from within ***Direct***SOFT.

AUX 22 Change Reference

There will be times when you need to change an I/O address reference or control relay reference. AUX 22 allows you to quickly and easily change all occurrences, (within an address range), of a specific instruction. For example, you can replace every instance of X5 with X10.

AUX 23 Clear Ladder Range

There have been many times when you take existing programs and add or remove certain portions to solve new application problems. By using AUX 23 you can select and delete a portion of the program. ***Direct***SOFT does not have a menu option for this AUX function, but you can select the appropriate portion of the program and cut it with the editing tools.

AUX 24 Clear Ladders

AUX 24 clears the entire program from CPU memory. Before you enter a new program, you should always clear ladder memory. This AUX function is available on the PLC/Clear PLC sub-menu within ***Direct***SOFT.

AUX 3* — V-memory Operations

- AUX 31 — Clear V memory

AUX 31 Clear V Memory

AUX 31 clears all the information from the V-memory locations available for general use. This AUX function is available on the PLC/Clear PLC sub-menu within ***Direct***SOFT.

AUX 4* — I/O Configuration

AUX 41
Show I/O Configuration

This AUX function allows you to display the current I/O configuration. With the Handheld Programmer, you can scroll through each base and I/O slot to view the complete configuration. The configuration shows the type of module installed in each slot. ***Direct***SOFT™ provides the same information, but it is much easier to view because you can view a complete base on one screen.

AUX 5* — CPU Configuration

AUX 51 – 58

There are several AUX functions available that you can use to setup, view, or change the CPU configuration.

- AUX 51 — Modify Program Name
- AUX 52 — Display / Change Calendar
- AUX 53 — Display Scan Time
- AUX 54 — Initialize Scratchpad
- AUX 55 — Set Watchdog Timer
- AUX 56 — Configure Comm Port
- AUX 57 — Set Retentive Ranges
- AUX 5C — Display Error / Message History

AUX 51
Modify Program Name

The DL305 products can use a program name for the CPU program or a program stored on EEPROM in the Handheld Programmer. Note, you cannot have multiple programs stored on the EEPROM. The program name can be up to eight characters in length and can use any of the available characters (A–Z, 0–9). AUX 51 allows you to enter a program name. You can also perform this operation from within ***Direct***SOFT™ by using the PLC/Setup sub-menu. Once you've entered a program name, you can only clear the name by using AUX 54 to reset the system memory. Make sure you understand the possible ramifications of AUX 54 before you use it!

AUX 52
Display /Change Calendar

The DL350 CPU has a clock and calendar feature. If you are using this, you can use the Handheld and AUX 52 to set the time and date. The following format is used.

- Date — Year, Month, Date, Day of week (0 – 6, Sunday thru Saturday)
- Time — 24 hour format, Hours, Minutes, Seconds

You can use the AUX function to change any component of the date or time. However, the CPU will not automatically correct any discrepancy between the date and the day of the week. For example, if you change the date to the 15th of the month and the 15th is on a Thursday, you will also have to change the day of the week (unless the CPU already shows the date as Thursday).

You can also perform this operation from within ***Direct***SOFT by using the PLC/Setup sub-menu.

AUX 53
Display Scan Time

AUX 53 displays the current, minimum, and maximum scan times. The minimum and maximum times are the ones that have occurred since the last Program Mode to Run Mode transition. You can also perform this operation from within ***Direct***SOFT by using the PLC/Diagnostics sub-menu.

AUX 54
Initialize Scratchpad

The DL350 CPU 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.

NOTE: You may never have to use this feature unless you have made changes that affect 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 change from program to program without ever initializing system memory.

AUX 54 resets the system memory to the default values. You can also perform this operation from within ***Direct***SOFT™ by using the PLC/Setup sub-menu.

AUX 55
Set Watchdog Timer

The DL350 CPU has a “watchdog” timer that is used to monitor the scan time. The default value set from the factory is 200 ms. If the scan time exceeds the watchdog time limit, the CPU automatically leaves RUN mode and enters PGM mode. The Handheld displays the following message E003 S/W TIMEOUT when the scan overrun occurs.

Use AUX 55 to increase or decrease the watchdog timer value. You can also perform this operation from within ***Direct***SOFT by using the PLC/Setup sub-menu.

AUX 56
CPU Network Address

Since the DL350 CPU has an additional communication port, you can use the Handheld to set the network address for the port and the port communication parameters. The default settings are:

- Station address 1
- HEX mode
- Odd parity

You can use this port with either the Handheld Programmer, ***Direct***SOFT, or, as a ***Direct***NET communication port. The ***Direct***NET Manual provides additional information about communication settings required for network operation.

NOTE: You will only need to use this procedure if you have the bottom port connected to a network. Otherwise, the default settings will work fine.

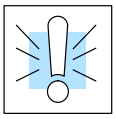
Use AUX 56 to set the network address and communication parameters. You can also perform this operation from within ***Direct***SOFT by using the PLC/Setup sub-menu.

AUX 57 Set Retentive Ranges

The DL350 CPU provides 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	DL350	
	Default Range	Avail. Range
Control Relays	C1000 – C1777	C0 – C1777
V Memory	V1400 – V37777	V0 – V37777
Timers	None by default	T0 – T377
Counters	CT0 – CT177	CT0 – CT177
Stages	None by default	S0 – S1777

Use AUX 57 to change the retentive ranges. You can also perform this operation from within ***Direct***SOFT™ by using the PLC/Setup sub-menu.



WARNING: The DL350 CPUs do not come with a battery. The super capacitor will retain the values in the event of a power loss, but only up to 1 week. The retention time may be less in some conditions. If the retentive ranges are important for your application, make sure you obtain the optional battery.

AUX 5C Display Error History

The DL350 CPU will automatically log any system error codes and custom messages created with the FAULT instructions. The CPU logs the error code, date, and time the error occurred. There are two separate tables that store this information.

- Error Code Table – the system logs up to 32 errors in the table. When an error occurs, the errors already on the table are pushed down and the most recent error is loaded into the top slot. If the table is full when an error occurs, the oldest error is pushed out (erased) of the table.
- Message Table – the system logs up to 16 messages in this table. When a message is triggered, the messages already stored in the table are pushed down and the most recent message is loaded into the top slot. If the table is full when an error occurs, the oldest message is pushed out (erased) of the table.

The following diagram shows an example of an error table for messages.

Date	Time	Message
1997–05–26	08:41:51:11	* Conveyor–2 stopped
1997–04–30	17:01:11:56	* Conveyor–1 stopped
1997–04–30	17:01:11:12	* Limit SW1 failed
1997–04–28	03:25:14:31	* Saw Jam Detect

You can use AUX Function 5C to show the error codes or messages. You can also view the errors and messages from within ***Direct***SOFT™ by using the PLC/Diagnostics sub-menu.

AUX 6* — Handheld Programmer Configuration

AUX 61 Show Revision Numbers

As with most industrial control products, there are cases when additional features and enhancements are made. Sometimes these new features only work with certain releases of firmware. By using AUX 61 you can quickly view the CPU and Handheld Programmer firmware revision numbers. This information (for the CPU) is also available from within ***Direct***SOFT from the PLC/Diagnostics sub-menu.

AUX 7* — EEPROM Operations

AUX 71 – 76

There are several AUX functions available you can use to move programs between the CPU memory and an optional EEPROM installed in the Handheld Programmer.

- AUX 71 — Read from CPU memory to HPP EEPROM
- AUX 72 — Write HPP EEPROM to CPU
- AUX 73 — Compare CPU to HPP EEPROM
- AUX 74 — Blank Check (HPP EEPROM)
- AUX 75 — Erase HPP EEPROM
- AUX 76 — Show EEPROM Type (CPU and HPP)

AUX 71 CPU to HPP EEPROM

AUX 71 copies information from the CPU memory to an EEPROM installed in the Handheld Programmer.

You can copy different portions of EEPROM (HP) memory to the CPU memory as shown in the previous table. The amount of data you can copy depends on the CPU.

AUX 72 HPP EEPROM to CPU

AUX 72 copies information from an EEPROM installed in the Handheld Programmer to the CPU. You can copy different types of information from CPU memory as shown in the previous table.

AUX 73 Compare HPP EEPROM to CPU

AUX 73 compares the program in the Handheld programmer (EEPROM) with the CPU program. You can compare different types of information as shown previously. There is also an option called “etc.” that allows you to check all of the areas sequentially without re-executing the AUX function every time.

AUX 74 HPP EEPROM Blank Check

AUX 74 allows you to check the EEPROM in the handheld programmer to make sure it is blank. It’s a good idea to use this function anytime you start to copy an entire program to an EEPROM in the handheld programmer.

AUX 75 Erase HPP EEPROM

AUX 75 allows you to clear all data in the EEPROM in the handheld programmer. You should use this AUX function before you copy a program from the CPU.

AUX 76 Show EEPROM Type

You can use AUX 76 to quickly determine what size EEPROM is installed in the Handheld Programmer.

AUX 8* — Password Operations

AUX 81 – 83

There are several AUX functions available that you can use to modify or enable the CPU password. You can use these features during on-line communications with the CPU, or, you can also use them with an EEPROM installed in the Handheld Programmer during off-line operation. This will allow you to develop a program in the Handheld Programmer and include password protection.

- AUX 81 — Modify Password
- AUX 82 — Unlock CPU
- AUX 83 — Lock CPU

AUX 81
Modify Password

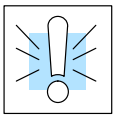
You can use AUX 81 to provide an extra measure of protection by entering a password that prevents unauthorized machine operations. If you are using the standard level password, it must be an eight-character numeric (0–9) code. Once you've entered a password, you can remove it by entering all zeros (00000000). This is the default from the factory.

The DL350 also features a multi–level password that you select by entering the character "A" and seven numeric characters. This level of protection differs from the standard in that it allows an operator interface device to access and change V–memory data (i.e., presets). However, it also does not allow a ladder program edit.

Once you've entered a password, you can lock the CPU against access. There are two ways to lock the CPU with the Handheld Programmer.

- The CPU is always locked after a power cycle (if a password is present).
- You can use AUX 83 and AUX 84 to lock and unlock the CPU.

You can also enter or modify a password from within ***Direct***SOFT™ by using the PLC/Password sub-menu. This feature works slightly differently in ***Direct***SOFT. Once you've entered a password, the CPU is automatically locked when you exit the software package. It will also be locked if the CPU is power cycled.



WARNING: Make *sure* you remember the password *before* you lock the CPU. Once the CPU is locked you cannot view, change, or erase the password. If you do not remember the password, you have to return the CPU to the factory for password removal.

NOTE: The D3–350 CPU supports 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).

AUX 82
Unlock CPU

AUX 81 can be used to unlock a CPU that has been password protected. ***Direct***SOFT will automatically ask you to enter the password if you attempt to communicate with a CPU that contains a password.

AUX 83
Lock CPU

AUX 83 can be used to lock a CPU that contains a password. Once the CPU is locked, you will have to enter a password to gain access. Remember, this is not necessary with ***Direct***SOFT since the CPU is automatically locked whenever you exit the software package.

DL305 Error Codes

B

In This Appendix. . . .

— Error Code Table

DL305 Error Code	Description
E003 SOFTWARE TIME-OUT	If the program scan time exceeds the time allotted to the watchdog timer, this error will occur. SP51 will be on and the error code will be stored in V7755. To correct this problem add RSTWT instructions in FOR NEXT loops and subroutines or use AUX 55 to extend the time allotted to the watchdog timer.
E041 CPU BATTERY LOW	The CPU battery is low and should be replaced. SP43 will be on and the error code will be stored in V7757.
E099 PROGRAM MEMORY EXCEEDED	If the compiled program length exceeds the amount of available CPU RAM this error will occur. SP52 will be on and the error code will be stored in V7755. Reduce the size of the application program.
E104 WRITE FAILED	A write to the CPU was not successful. Disconnect the power, remove the CPU, and make sure the EEPROM is not write protected. If the EEPROM is not write protected, make sure the EEPROM is installed correctly. If both conditions are OK, replace the CPU.
E151 BAD COMMAND	A parity error has occurred in the application program. SP44 will be on and the error code will be stored in V7755 .This problem may possibly be due to electrical noise .Clear the memory and download the program again. Correct any grounding problems .If the error returns replace the EEPROM or the CPU.
E155 RAM FAILURE	A checksum error has occurred in the system RAM. SP44 will be on and the error code will be stored in V7755. This problem may possibly be due to a low battery, electrical noise or a CPU RAM failure. Clear the memory and download the program again. Correct any grounding problems. If the error returns replace the CPU.
E202 MISSING I/O MODULE	An I/O module has failed to communicate with the CPU or is missing from the base. SP45 will be on and the error code will be stored in V7756. Run AUX42 to determine the slot and base location of the module reporting the error.
E210 POWER FAULT	A short duration power drop-out occurred on the main power line supplying power to the base.
E250 COMMUNICATION FAILURE IN THE I/O CHAIN	A failure has occurred in the local I/O system. The problem could be in the base I/O bus or the base power supply. SP45 will be on and the error code will be stored in V7755. Run AUX42 to determine the base location reporting the error.
E252 NEW I/O CFG	This error occurs when the auto configuration check is turned on in the CPU and the actual I/O configuration has changed either by moving modules in a base or changing types of modules in a base. You can return the modules to the original position/types or run AUX45 to accept the new configuration. SP47 will be on and the error code will be stored in V7755.
E262 I/O OUT OF RANGE	An out of range I/O address has been encountered in the application program. Correct the invalid address in the program. SP45 will be on and the error code will be stored in V7755.

DL305 Error Code	Description
E312 HP COMM ERROR 2	A data error was encountered during communications with the CPU. Clear the error and retry the request. If the error continues check the cabling between the two devices, replace the handheld programmer, then if necessary replace the CPU. SP46 will be on and the error code will be stored in V7756.
E313 HP COMM ERROR 3	An address error was encountered during communications with the CPU. Clear the error and retry the request. If the error continues check the cabling between the two devices, replace the handheld programmer, then if necessary replace the CPU. SP46 will be on and the error code will be stored in V7756.
E316 HP COMM ERROR 6	A mode error was encountered during communications with the CPU. Clear the error and retry the request. If the error continues replace the handheld programmer, then if necessary replace the CPU. SP46 will be on and the error code will be stored in V7756.
E320 HP COMM TIME-OUT	The CPU did not respond to the handheld programmer communication request. Check to insure cabling is correct and not defective. Power cycle the system if the error continues replace the CPU first and then the handheld programmer if necessary.
E321 COMM ERROR	A data error was encountered during communication with the CPU. Check to insure cabling is correct and not defective. Power cycle the system and if the error continues replace the CPU first and then the handheld programmer if necessary.
E4** NO PROGRAM	A syntax error exists in the application program. The most common is a missing END statement. Run AUX21 to determine which one of the E4** series of errors is being flagged. SP52 will be on and the error code will be stored in V7755.
E401 MISSING END STATEMENT	All application programs must terminate with an END statement. Enter the END statement in appropriate location in your program. SP52 will be on and the error code will be stored in V7755.
E402 MISSING LBL	A GOTO, GTS, MOVMC or LDLBL instruction was used without the appropriate label. Refer to the programming manual for details on these instructions. SP52 will be on and the error code will be stored in V7755.
E403 MISSING RET	A subroutine in the program does not end with the RET instruction. SP52 will be on and the error code will be stored in V7755.
E404 MISSING FOR	A NEXT instruction does not have the corresponding FOR instruction. SP52 will be on and the error code will be stored in V7755.

DL305 Error Code	Description
E405 MISSING NEXT	A FOR instruction does not have the corresponding NEXT instruction. SP52 will be on and the error code will be stored in V7755.
E406 MISSING IRT	An interrupt routine in the program does not end with the IRT instruction. SP52 will be on and the error code will be stored in V7755.
E412 SBR/LBL>64	There is greater than 64 SBR, LBL or DLBL instructions in the program. This error is also returned if there is greater than 128 GTS or GOTO instructions used in the program. SP52 will be on and the error code will be stored in V7755.
E413 FOR/NEXT>64	There is greater than 64 FOR/NEXT loops in the application program. SP52 will be on and the error code will be stored in V7755.
E421 DUPLICATE STAGE REFERENCE	Two or more SG or ISG labels exist in the application program with the same number. A unique number must be allowed for each Stage and Initial Stage. SP52 will be on and the error code will be stored in V7755.
E422 DUPLICATE SBR/LBL REFERENCE	Two or more SBR or LBL instructions exist in the application program with the same number. A unique number must be allowed for each Subroutine and Label. SP52 will be on and the error code will be stored in V7755.
E423 NESTED LOOPS	Nested loops (programming one FOR/NEXT loop inside of another) is not allowed in the DL350 series. SP52 will be on and the error code will be stored in V7755.
E431 INVALID ISG/SG ADDRESS	An ISG or SG must not be programmed after the end statement such as in a subroutine. SP52 will be on and the error code will be stored in V7755.
E432 INVALID JUMP (GOTO) ADDRESS	A LBL that corresponds to a GOTO instruction must not be programmed after the end statement such as in a subroutine. SP52 will be on and the error code will be stored in V7755.
E433 INVALID SBR ADDRESS	A SBR must be programmed after the end statement, not in the main body of the program or in an interrupt routine. SP52 will be on and the error code will be stored in V7755.
E435 INVALID RT ADDRESS	A RT must be programmed after the end statement, not in the main body of the program or in an interrupt routine. SP52 will be on and the error code will be stored in V7755.

DL305 Error Code	Description
E436 INVALID INT ADDRESS	An INT must be programmed after the end statement, not in the main body of the program. SP52 will be on and the error code will be stored in V7755.
E438 INVALID IRT ADDRESS	An IRT must be programmed after the end statement, not in the main body of the program. SP52 will be on and the error code will be stored in V7755.
E440 INVALID DATA ADDRESS	Either the DLBL instruction has been programmed in the main program area (not after the END statement), or the DLBL instruction is on a rung containing input contact(s).
E441 ACON/NCON	An ACON or NCON must be programmed after the end statement, not in the main body of the program. SP52 will be on and the error code will be stored in V7755.
E451 BAD MLS/MLR	MLS instructions must be numbered in ascending order from top to bottom.
E452 X AS COIL	An X data type is being used as a coil output.
E453 MISSING T/C	A timer or counter contact is being used where the associated timer or counter does not exist.
E454 BAD TMRA	One of the contacts is missing from a TMRA instruction.
E455 BAD CNT	One of the contacts is missing from a CNT or UDC instruction.
E456 BAD SR	One of the contacts is missing from the SR instruction.
E461 STACK OVERFLOW	More than nine levels of logic have been stored on the stack. Check the use of OR STR and AND STR instructions.
E462 STACK UNDERFLOW	An unmatched number of logic levels have been stored on the stack. Insure the number of AND STR and OR STR instructions match the number of STR instructions.
E463 LOGIC ERROR	A STR instruction was not used to begin a rung of ladder logic.
E464 MISSING CKT	A rung of ladder logic is not terminated properly.
E471 DUPLICATE COIL REFERENCE	Two or more OUT instructions reference the same I/O point.
E472 DUPLICATE TMR REFERENCE	Two or more TMR instructions reference the same number.

DL305 Error Code	Description
E473 DUPLICATE CNT REFERENCE	Two or more CNT instructions reference the same number.
E480 INVALID CV ADDRESS	The CV instruction is used in a subroutine or program interrupt routine. The CV instruction may only be used in the main program area (before the END statement).
E481 CONFLICTING INSTRUCTIONS	An instruction exists between convergence stages.
E482 MAX. CV INSTRUCTIONS EXCEEDED	Number of CV instructions exceeds 17.
E483 INVALID CVJMP ADDRESS	CVJMP has been used in a subroutine or a program interrupt routine.
E484 MISSING CV INSTRUCTION	CVJMP is not preceded by the CV instruction. A CVJMP must immediately follow the CV instruction.
E485 NO CVJMP	A CVJMP instruction is not placed between the CV and the SG, ISG, BLK, BEND, END instruction.
E486 INVALID BCALL ADDRESS	A BCALL is used in a subroutine or a program interrupt routine. The BCALL instruction may only be used in the main program area (before the END statement).
E487 MISSING BLK INSTRUCTION	The BCALL instruction is not followed by a BLK instruction.
E488 INVALID BLK ADDRESS	The BLK instruction is used in a subroutine or a program interrupt. Another BLK instruction is used between the BCALL and the BEND instructions.
E489 DUPLICATED CR REFERENCE	The control relay used for the BLK instruction is being used as an output elsewhere.

DL305 Error Code	Description
E490 MISSING SG INSTRUCTION	The BLK instruction is not immediately followed by the SG instruction.
E491 INVALID ISG INSTRUCTION ADDRESS	There is an ISG instruction between the BLK and BEND instructions.
E492 INVALID BEND ADDRESS	The BEND instruction is used in a subroutine or a program interrupt routine. The BEND instruction is not followed by a BLK instruction.
E493 MISSING REQUIRED INSTRUCTION	A [CV, SG, ISG, BLK, BEND] instruction must immediately follow the BEND instruction.
E494 MISSING BEND INSTRUCTION	The BLK instruction is not followed by a BEND instruction.
E499 PRINT INSTRUCTION	Invalid PRINT instruct usage. Quotations and/or spaces were not entered or entered incorrectly.
E501 BAD ENTRY	An invalid keystroke or series of keystrokes was entered into the handheld programmer.
E502 BAD ADDRESS	An invalid or out of range address was entered into the handheld programmer.
E503 BAD COMMAND	An invalid instruction was entered into the handheld programmer.
E504 BAD REF/VAL	An invalid value or reference number was entered with an instruction.
E505 INVALID INSTRUCTION	An invalid instruction was entered into the handheld programmer.
E506 INVALID OPERATION	An invalid operation was attempted by the handheld programmer.
E520 BAD OP–RUN	An operation which is invalid in the RUN mode was attempted by the handheld programmer.
E521 BAD OP–TRUN	An operation which is invalid in the TEST RUN mode was attempted by the handheld programmer.
E523 BAD OP–TPGM	An operation which is invalid in the TEST PROGRAM mode was attempted by the handheld programmer.
E524 BAD OP–PGM	An operation which is invalid in the PROGRAM mode was attempted by the handheld programmer.

DL305 Error Code	Description
E525 MODE SWITCH	An operation was attempted by the handheld programmer while the CPU mode switch was in a position other than the TERM position.
E526 OFF LINE	The handheld programmer is in the OFFLINE mode. To change to the ONLINE mode use the MODE key.
E527 ON LINE	The handheld programmer is in the ON LINE mode. To change to the OFF LINE mode use the MODE the key.
E528 CPU MODE	The operation attempted is not allowed during a Run Time Edit.
E540 CPU LOCKED	The CPU has been password locked. To unlock the CPU use AUX82 with the password.
E541 WRONG PASSWORD	The password used to unlock the CPU with AUX82 was incorrect.
E542 PASSWORD RESET	The CPU powered up with an invalid password and reset the password to 00000000. A password may be re-entered using AUX81.
E601 MEMORY FULL	Attempted to enter an instruction which required more memory than is available in the CPU.
E602 INSTRUCTION MISSING	A search function was performed and the instruction was not found.
E604 REFERENCE MISSING	A search function was performed and the reference was not found.
E610 BAD I/O TYPE	The application program has referenced an I/O module as the incorrect type of module.
E620 OUT OF MEMORY	Incorrect structure of LDLBL, MOV, or MOVMC command. An attempt to transfer more data between the CPU and handheld programmer than the receiving device can hold.
E621 EEPROM NOT BLANK	An attempt to write to a non-blank EEPROM was made. Erase the EEPROM and then retry the write.
E622 NO HPP EEPROM	A data transfer was attempted with no EEPROM (or possibly a faulty EEPROM) installed in the handheld programmer.
E623 SYSTEM EEPROM	A function was requested with an EEPROM which contains system information only.
E624 V-MEMORY ONLY	A function was requested with an EEPROM which contains V-memory data only.
E625 PROGRAM ONLY	A function was requested with an EEPROM which contains program data only.

DL305 Error Code	Description
E627 BAD WRITE	An attempt to write to a write protected or faulty EEPROM was made. Check the write protect jumper and replace the EEPROM if necessary.
E640 COMPARE ERROR	A compare between the EEPROM and the CPU was found to be in error.
E650 HPP SYSTEM ERROR	A system error has occurred in the handheld programmer. Power cycle the handheld programmer. If the error returns replace the handheld programmer.
E651 HPP ROM ERROR	A ROM error has occurred in the handheld programmer. Power cycle the handheld programmer. If the error returns replace the handheld programmer.
E652 HPP RAM ERROR	A RAM error has occurred in the handheld programmer. Power cycle the handheld programmer. If the error returns replace the handheld programmer.

C

Instruction Execution Times

In This Appendix. . . .

Introduction

This appendix contains several tables that provide the instruction execution times for the DL350 CPU. You will notice is that many of the execution times depend on the type of data being used with the instruction. For example, a few of the instructions that use V-memory locations are further defined by the following items.

- Data Registers
- Bit Registers

V-Memory Data Registers

Some V-memory locations are considered data registers. For example, the V-memory locations that store the timer or counter current values, or just regular user V memory would be considered as a V-memory data register. Don't think that you cannot load a bit pattern into these types of registers, you can. It's just that their primary use is as a data register. The following locations are considered as data registers.

Data Registers	DL350
Timer Current Values	V0 – V377
Counter Current Values	V1000 – V1177
User Data Words	V1400 – V7377 V10000 – V17777

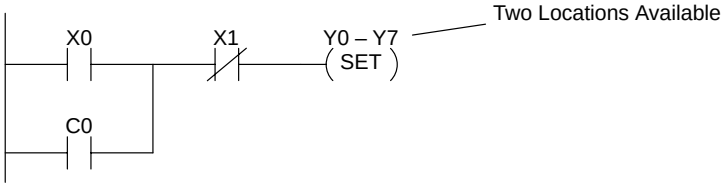
V-Memory Bit Registers

You may recall that some of the discrete points such as X, Y, C, etc. are automatically mapped into V memory. The following locations that contain this data are considered bit registers.

Bit Registers	DL350
Input Points (X)	V40400 – V 40437
Output Points (Y)	V40500 – V40537
Control Relays (C)	V40600 – V40677
Timer Status Bits	V41100 – V41117
Counter Status Bits	V41040 – V41147
Stages	V41000 – V41077

How to Read the Tables

Some of the instructions can have more than one parameter so the table shows execution times that depend on the amount and type of parameters. For example, the SET instruction can be used to set a single point or a range of points. If you examine the execution table you'll notice the available data types and execution times for both situations. The following diagram shows an example.



Execution depends on numbers of locations and types of data used

SET	1st #:	X, Y, C, S	17.4 μs
	2nd #:	X, Y, C, S, (N pt)	12.0μs+5.4μsxN
RST	1st #:	X, Y, C, S	19.5 μs
	2nd #:	X, Y, C, S, (N pt)	10.5μs+5.2μsxN

Boolean Instructions

Boolean Instructions		DL350	
Instruction	**Legal Data Types**	**Execute**	**Not Exec**
STR	X, Y, C, T, CT,S, SP	.74 µs	.74 µs
STRN	X, Y, C, T, CT,S, SP	0.68 µs	0.74 µs
OR	X, Y, C, T, CT, S, SP	0.56 µs	0.56 µs
ORN	X, Y, C, T, CT,S, SP	0.6 µs	0.6 µs
AND	X, Y, C, T, CT, S, SP	0.46 µs	0.46 µs
ANDN	X, Y, C, T, CT, S, SP	0.56 µs	0.56 µs
ANDSTR	None	0.4 µs	0.4 µs
ORSTR	None	0.4 µs	0.4 µs
OUT	X, Y, C	2.0 µs	2.0 µs
OUTH	X, Y, C	1.1 µs	1.1 µs
OROUT	X, Y, C	2.4 µs	2.4 µs
PD	X, Y, C	16.6 µs	16.6 µs
SET	1st #: X, Y, C, S 2nd #: X, Y, C, S (N pt)	10.6 µs 11.4µs+ 0.9µsxN	1.1 µs 1.1 µs
RST	1st #: X, Y, C, S 2nd #: X, Y, C, S (N pt)	10.6 µs 11.4µs+ 0.9µsxN	1.1 µs 1.1 µs
	1st #: T, CT 2nd #: T, CT (N pt)	10.6 µs 11.4µs+ 0.9µsxN	1.1 µs 1.1 µs

Comparative Boolean

Comparative Boolean Instructions			DL350	
Instruction	**Legal Data Types**		**Execute**	**Not Exec**
STRE	1st	2nd		
	V: Data Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	V: Bit Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	P:Indir. (Data)	V:Data Reg.	—	
		V:Bit Reg.	35.6µs	35.6µs
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		
	P:Indir. (Bit)	V:Data Reg.	35.6µs	35.6µs
		V:Bit Reg.		—
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		
STRNE	1st	2nd		
	V: Data Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	V: Bit Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	P:Indir. (Data)	V:Data Reg.	—	—
		V:Bit Reg.	35.6µs	35.6µs
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		
	P:Indir. (Bit)	V:Data Reg.	35.6µs	35.6µs
		V:Bit Reg.		
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		

Comparative Boolean (cont.)			DL350	
Instruct	**Legal Data Types**		**Execute**	**Not Exec**
ORE	1st	2nd		
	V: Data Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	V: Bit Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	P:Indir. (Data)	V:Data Reg.	—	—
		V:Bit Reg.	35.6µs	35.6µs
		K:Constant		
		P:Indir. (Data)	32.6µs	32.6µs
		P:Indir. (Bit)	60.7µs	60.7µs
	P:Indir. (Bit)	V:Data Reg.	35.6µs	35.6µs
		V:Bit Reg.		
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		
ORNE	1st	2nd		
	V: Data Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	V: Bit Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	P:Indir. (Data)	V:Data Reg.	—	—
		V:Bit Reg.	35.6µs	35.6µs
		K:Constant		
		P:Indir. (Data)	32.6µs	32.6µs
		P:Indir. (Bit)	60.7µs	60.7µs
	P:Indir. (Bit)	V:Data Reg.	35.6µs	35.6µs
		V:Bit Reg.		
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		

Comparative Boolean (cont.)			DL350	
Instruct	**Legal Data Types**		**Execute**	**Not Exec**
ANDE	1st	2nd		
	V: Data Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	V: Bit Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	P:Indir. (Data)	V:Data Reg.	—	—
		V:Bit Reg.	35.6µs	35.6µs
		K:Constant		
		P:Indir. (Data)	32.6µs	32.6µs
		P:Indir. (Bit)	60.7µs	60.7µs
	P:Indir. (Bit)	V:Data Reg.	35.6µs	35.6µs
		V:Bit Reg.		
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		
ANDNE	1st	2nd		
	V: Data Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	V: Bit Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	P:Indir. (Data)	V:Data Reg.	—	—
		V:Bit Reg.	35.6µs	35.6µs
		K:Constant		
		P:Indir. (Data)	32.6µs	32.6µs
		P:Indir. (Bit)	60.7µs	60.7µs
			35.6µs	35.6µs
	P:Indir. (Bit)	V:Data Reg.		
		V:Bit Reg.	32.6µs	32.6µs
		K:Constant		
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		

Comparative Boolean (cont.)			DL350	
Instruc	**Legal Data Types**		**Execute**	**Not Exec**
STR	1st	2nd		
	T, CT	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	8.7µs 5.5µs 35.9µs	8.7µs 5.5µs 35.9µs
	1st	2nd		
	V: Data Reg.	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	8.7µs 5.5µs 35.9µs	8.7µs 5.5µs 35.9µs
	V: Bit Reg.	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	8.7µs 5.5µs 35.9µs	8.7µs 5.5µs 35.9µs
	P:Indir. (Data)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	— 35.6µs 32.6µs 60.7µs	— 35.6µs 32.6µs 60.7µs
	P:Indir. (Bit)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	35.6µs 32.6µs 60.7µs	35.6µs 32.6µs 60.7µs
STRN	1st	2nd		
	T, CT	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	8.7µs 5.5µs 35.9µs	8.7µs 5.5µs 35.9µs
	1st	2nd		
	V: Data Reg.	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	8.7µs 5.5µs 35.9µs	8.7µs 5.5µs 35.9µs
	V: Bit Reg.	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	8.7µs 5.5µs 35.9µs	8.7µs 5.5µs 35.9µs
	P:Indir. (Data)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	— 35.6µs 32.6µs 60.7µs	— 35.6µs 32.6µs 60.7µs
	P:Indir. (Bit)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	35.6µs 32.6µs 60.7µs	35.6µs 32.6µs 60.7µs

Comparative Boolean (cont.)			DL350	
Instruc	**Legal Data Types**		**Execute**	**Not Exec**
OR	1st	2nd		
	T, CT	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	1st	2nd		
	V: Data Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	V: Bit Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	P:Indir. (Data)	V:Data Reg.	—	—
		V:Bit Reg.	35.6µs	35.6µs
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		
	P:Indir. (Bit)	V:Data Reg.	35.6µs	35.6µs
		V:Bit Reg.		
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		
ORN	1st	2nd		
	T, CT	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	1st	2nd		
	V: Data Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	V: Bit Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	P:Indir. (Data)	V:Data Reg.	—	—
		V:Bit Reg.	35.6µs	35.6µs
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		
	P:Indir. (Bit)	V:Data Reg.	35.6µs	35.6µs
		V:Bit Reg.		
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		

Comparative Boolean (cont.)			DL350	
Instruc	**Legal Data Types**		**Execute**	**Not Exec**
AND	1st	2nd		
	T, CT	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	1st	2nd		
	V: Data Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	V: Bit Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	P:Indir. (Data)	V:Data Reg.	—	—
		V:Bit Reg.	35.6µs	35.6µs
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		
	P:Indir. (Bit)	V:Data Reg.	35.6µs	35.6µs
		V:Bit Reg.		
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		
ANDN	1st	2nd		
	T, CT	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	1st	2nd		
	V: Data Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	V: Bit Reg.	V:Data Reg.	8.7µs	8.7µs
		V:Bit Reg.		
		K:Constant	5.5µs	5.5µs
		P:Indir. (Data)	35.9µs	35.9µs
		P:Indir. (Bit)		
	P:Indir. (Data)	V:Data Reg.	—	—
		V:Bit Reg.	35.6µs	35.6µs
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		
	P:Indir. (Bit)	V:Data Reg.	35.6µs	35.6µs
		V:Bit Reg.		
		K:Constant	32.6µs	32.6µs
		P:Indir. (Data)	60.7µs	60.7µs
		P:Indir. (Bit)		

Immediate Instructions

Immediate Instructions		DL350	
Instruc	**Legal Data Types**	**Execute**	**Not Exec**
STRI	X	78.6 µs	78.6 µs
STRNI	X	78.6 µs	78.6 µs
ORI	X	78.6 µs	78.6 µs
ORNI	X	78.6 µs	78.6 µs
ANDI	X	78.6 µs	78.6 µs
ANDNI	X	78.6 µs	78.6 µs
OUTI	Y	91.0 µs	91.0 µs
OROUTI	Y	94.0 µs	94.0 µs
SETI	1st #: Y 2nd #: Y (N pt)	87.6 µs 97.5µs+ 16.25xN	1.1 µs 1.1 µs
RSTI	1st #: Y 2nd #: Y (N pt)	87.6 µs 97.5µs+ 16.25xN	1.1 µs

Timer, Counter, Shift Register Instructions

Timer, Counter, Shift Register Instructions			DL350	
Instruc	**Legal Data Types**		**Execute**	**Not Exec**
TMR	1st	2nd		
	T	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	38.6µs 23.0µs 54.3µs	24.6µs 24.6µs 52.0µs
TMRF	1st	2nd		
	T	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	61.2µs 57.6µs 90.4µs	23.0µs 19.4µs 37.5µs
TMRA	1st	2nd		
	T	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	58.2µs 53.6µs 90.4µs	27.1µs 22.4µs 59.2µs
TMRAF	1st	2nd		
	T	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	64.5µs 59.9µs 96.7µs	27.6µs 22.4µs 59.2µs
CNT	1st	2nd		
	CT	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	36.1µs 32.5µs 97.1µs	24.6µs 21.0µs 56.8µs
SGCNT	1st	2nd		
	CT	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	35.2µs 33.7µs 67.4µs	27.7µs 27.1µs 57.9µs
UDC	1st	2nd		
	CT	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	47.4µs 42.7µs 81.7µs	40.0µs 35.3µs 72.1µs
SR	C (N points to shift)		17.8µs+ 1.0µsxN	12.6 µs

Accumulator Data Instructions

Accumulator / Stack Load and Output Data Instructions		DL350	
Instruc	**Legal Data Types**	**Execute**	**Not Exec**
LD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	13.6µs 10.4µs 40.4µs	1.1µs 1.1µs 1.1µs
LDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	14.0µs 10.4µs 45.0µs	1.1µs 1.1µs 1.3µs
LDF	1st: X, Y, C, S, T, CT, SP 2nd: K:Constant	10.5µs+ 3.45µs x N	1.4µs
LDA	O: (Octal constant for address)	10.4 µs	1.1µs
LDSX	K: Constant	14.6 µs	1.5µs
OUT	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	10.7 µs 41.9 µs	1.1µs
OUTD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	11.7 µs 42.6 µs	1.1µs
OUTF	1st: X, Y, C 2nd: K:Constant	43.8µs+ 6.2µs x N	1.1µs
POP	None	7.8 µs	1.0µs

Logical Instructions

Logical (Accumulator) Instructions		DL350	
Instruc	**Legal Data Types**	**Execute**	**Not Exec**
AND	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	9.1µs 39.8µs	1.1µs 1.1µs
ANDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	10.2µs 6.5µs 40.9µs	1.1µs 1.1µs 1.1µs
OR	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	9.3µs 40.2µs	1.1µs 1.1µs
ORD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	10.4µs 6.7µs 41.1µs	1.1µs 1.1µs 1.1µs
XOR	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	9.2µs 40.0µs	1.1µs 1.1µs
XORD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	10.3µs 6.2µs 41.0µs	1.1µs 1.1µs 1.1µs
CMP	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	10.8µs 41.5µs	1.1µs 1.1µs
CMPD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	11.4µs 7.7µs 42.1µs	1.2µs 1.2µs 1.2µs
CMPS	None	—	—

Math Instructions

Math Instructions (Accumulator)		DL350	
Instruc	**Legal Data Types**	**Execute**	**Not Exec**
ADD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	93.3µs 129.8µs 	1.2µs 1.1µs
ADDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	99.2µs 80.6µs 129.8µs 	1.2µs 1.2µs 1.2µs
SUB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	92.1µs 121.9µs 	1.1µs 1.1µs
SUBD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	98.2µs 78.6µs 127.8µs 	1.1µs 1.1µs 1.1µs
MUL	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	341.1µs 367.8 371.8µs 	1.1µs 1.1µs 1.1µs
MULD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	1075.8µs 1106.5µs 	1.1µs 1.1µs
DIV	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	466.6µs 492.8µs 538.2µs 	1.1µs 1.1µs 1.1µs
DIVD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	510.6µs 501.1µs 	1.1µs 1.1µs
INCB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	15.2µs 45.9µs 	1.1µs 1.1µs
DECB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	15.2µs 45.2µs 	1.1µs 1.1µs

Bit Instructions

Bit Instructions (Accumulator)		DL350	
Instruc	**Legal Data Types**	**Execute**	**Not Exec**
SHFR	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	9.8µs+ 0.2 x N 7.9µs+0.2 x N	1.2 µs
SHFL	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	9.8µs+ 0.2 x N 7.9µs+ 0.2 x N	1.2 µs
ROTR	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	15.7 12.3	1.2 µs
ROTL	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	15.7µs 12.3µs	1.2 µs
ENCO	None	40.3 µs	1.0 µs
DECO	None	6.5 µs	1.0 µs

Number Conversion Instructions

Number Conversion Instructions (Accumulator)		DL350	
Instruc	**Legal Data Types**	**Execute**	**Not Exec**
BIN	None	128.4 µs	1.0 µs
BCD	None	122.0 µs	1.0 µs
INV	None	2.9 µs	1.0 µs
BCDCPL	None	74.5 µs	1.0 µs
ATH	None	29.2 µs	1.0 µs
HTA	None	29.2 µs	1.0 µs
SEG	None	12.6 µs	1.0 µs
GRAY	None	142.0 µs	1.0 µs
SFLDGT	None	26.6 µs	1.0 µs

Table Instructions

Table Instructions		DL350	
Instruc	**Legal Data Types**	**Execute**	**Not Exec**
MOV	Move V:data reg. to V:data reg . Move V:bit reg. to V:data reg. Move V:data reg to V:bit reg. Move V:bit reg. to V:bit reg. N= #of words	63µs+ 16xN	1.20 µs
MOVMC	Move V:Data Reg. to E^2 Move V:Bit Reg. to E^2 Move from E^2 to V:Data Reg. Move from E^2 to V:Bit Reg. N= #of words	50µs+ 15xN	1.2 µs
LDLBL	K	7.4µs	1.5 µs

CPU Control Instructions

CPU Control Instructions		DL350	
Instruct	**Legal Data Types**	**Execute**	**Not Exec**
NOP	None	0.6 µs	0.6 µs
END	None	14.7 µs	14.7 µs
STOP	None	4.1 µs	1.0 µs
RSTWT	None	5.4 µs	1.0 µs
NOT	None	1.0 µs	1.0 µs

Program Control Instructions

Program Control Instructions		DL350	
Instruct	**Legal Data Types**	**Execute**	**Not Exec**
GOTO	K	5.0 µs	4.9 µs
LBL	K	0.6 µs	0.6 µs
FOR	V, K	110 µs	7.9 µs
NEXT	None	48.4 µs	0 µs
GTS	K	12.5 µs	6.3 µs
SBR	K	0.5 µs	0 µs
RT	None	11.4 µs	11.4 µs
MLS	K (1–7)	4.2 µs	4.2 µs
MLR	K (0–7)	4.0 µs	4.0 µs

Interrupt Instructions

Interrupt Instructions		DL350	
Instruc	**Legal Data Types**	**Execute**	**Not Exec**
ENI	None	45.8 µs	1.1 µs
DISI	None	5.7 µs	1.1 µs
INT	0 (0–7)	0 µs	0 µs
IRT	None	1.5 µs	—
IRTC	None	0.5 µs	0.5

Network Instructions

Network Instructions		DL350	
Instruc	**Legal Data Types**	**Execute**	**Not Exec**
RX	X, Y, C, T, CT, SP, S V:Data Reg. V:Bit Reg.	2024.1 µs	1.4 µs
WX	X, Y, C, T, CT, SP, S V:Data Reg. V:Bit Reg.	2024.1 µs	1.4 µs

Message Instructions

Message Instructions		DL350	
Instruc	**Legal Data Types**	**Execute**	**Not Exec**
FAULT	V:Data Reg. V:Bit Reg. K:Constant	108.9 µs 108.9 µs 96.2 µs	1.4 µs 1.4 µs 1.4 µs
DLBL	K	0 µs	0 µs
NCON	K	0 µs	µs
ACON	K	0 µs	0 µs
PRINT		104.0 µs	1.4 µs

RLL*PLUS* Instructions

RLL*PLUS* Instructions		DL350	
Instruc	**Legal Data Types**	**Execute**	**Not Exec**
ISG	S	24.3 µs	21.5 µs
SG	S	24.3 µs	21.5 µs
JMP	S	24.4 µs	4.3 µs
NJMP	S	24.4 µs	4.6 µs
CV	S	13.9 µs	13.9 µs
CVJMP	S (N stages, 1 to 16)	12.6µs	12.6 µs
BCALL	C	17.1 µs	17.1 µs
BLK	C	22.1 µs	22.6 µs
BEND	None	8.7 µs	0 µs

Clock / Calendar Instructions

Clock / Calendar Instructions		DL350	
Instruction	**Legal Data Types**	**Execute**	**Not Exec**
DATE	V:Data Reg. V:Bit Reg.	21.3 µs 21.3 µs	1.9 µs 1.9 µs
TIME	V:Data Reg. V:Bit Reg.	13.2 µs 13.2 µs	1.9 µs 1.9 µs

Drum Instructions

Drum Instructions		DL350	
Instruction	**Legal Data Types**	**Execute**	**Not Exe.**
DRUM	CT	340.0 µs	62.6 µs
EDRUM	CT	243.0 µs	100.0 µs
MDRMD	CT	206.0 µs	142.00 µs
MDRMW	CT	150.0 µs	94.00 µs

Special Relays

D

In This Appendix. . . .

— DL350 CPU Special Relays

DL350 CPU Special Relays

Startup and Real-Time Relays

SP0	First scan	on for the first scan after a power cycle or program to run transition only. The relay is reset to off on the second scan. It is useful where a function needs to be performed only on program startup.
SP1	Always ON	provides a contact to insure an instruction is executed every scan.
SP2	Always OFF	provides a contact that is always off.
SP3	1 minute clock	on for 30 seconds and off for 30 seconds.
SP4	1 second clock	on for 0.5 second and off for 0.5 second.
SP5	100 ms clock	on for 50 ms. and off for 50 ms.
SP6	50 ms clock	on for 25 ms. and off for 25 ms.
SP7	Alternate scan	on every other scan.

CPU Status Relays

SP11	Forced run mode	on anytime the CPU switch is in the RUN position.
SP12	Terminal run mode	on when the CPU switch is in the TERM position and the CPU is in the RUN mode.
SP13	Test run mode	on when the CPU switch is in the TERM position and the CPU is in the test RUN mode.
SP14	Test hold mode	on when theCPU switch is in the TERM position and the CPU is in the TEST HOLD mode
SP15	Test program mode	on when the CPU is in the TERM position and the CPU is in the TEST PROGRAM MODE.
SP16	Terminal program mode	on when the CPU switch is in the TERM position and the CPU is in the PROGRAM MODE.
SP17	Forced stop mode relay	on anytime the CPU mode switch is in the STOP position.
SP20	Forced stop mode	on when the STOP instruction is executed.
SP21	Break Relay 2	on when the BREAK instructions is executed. It is OFF when the CPU mode is changed to RUN.
SP22	Interrupt enabled	on when interrupts have been enabled using the ENI instruction.
SP25	CPU battery disabled relay	on when the CPU battery is disabled by special V–memory.

System Monitoring Relays

SP40	Critical error	on when a critical error such as I/O communication loss has occurred.
SP41	Warning	on when a non-critical error such as a low battery has occurred.
SP43	Battery low	on when the CPU battery voltage is low.
SP44	Reserved	
SP45	Reserved	
SP46	Communications error	on when a communications error has occurred on any of the CPU ports.
SP47	I/O configuration error	on if an I/O configuration error has occurred. The CPU power-up I/O configuration check must be enabled before this relay will be functional.
SP50	Fault instruction	on when a Fault Instruction is executed.
SP51	Watch Dog timeout	on if the CPU Watch Dog timer times out.
SP52	Grammatical error	on if a grammatical error has occurred either while the CPU is running or if the syntax check is run. V7755 contains the exact error code.
SP53	Solve logic error	on if CPU cannot solve the logic.
SP54	Intelligent I/O error	on when communications with an intelligent module has occurred.

Accumulator Status Relays

SP60	Value less than	on when the accumulator value is less than the instruction value.
SP61	Value equal to	on when the accumulator value is equal to the instruction value.
SP62	Greater than	on when the accumulator value is greater than the instruction value.
SP63	Zero	on when the result of the instruction is zero (in the accumulator.)
SP64	Half borrow	on when the 16 bit subtraction instruction results in a borrow.
SP65	Borrow	on when the 32 bit subtraction instruction results in a borrow.
SP66	Half carry	on when the 16 bit addition instruction results in a carry.
SP67	Carry	when the 32 bit addition instruction results in a carry.
SP70	Sign	on anytime the value in the accumulator is negative.
SP71	Invalid octal number	on when an Invalid octal number was entered. This also occurs when the V-memory specified by a pointer (P) is not valid.
SP72	Invalid Real Number	On when an invalid real number is in the accumulator
SP73	Overflow	on if overflow occurs in the accumulator when a signed addition or subtraction results in a incorrect sign bit.
SP74	Underflow	On if real number underflow occurs in the accumulator (numbers are too close to 0.0)
SP75	Data error	on if a BCD number is expected and a non–BCD number is encountered.
SP76	Load zero	on when any instruction loads a value of zero into the accumulator.

Communications Monitoring Relays

SP116	DL350 CPU communication	on when port 2 is communicating with another device
SP117	Comm error port 2	on when Port 2 has encountered a communication error.
SP120	Module busy Slot 0	on when the communication module in slot 0 is busy transmitting or receiving. You must use this relay with the RX or WX instructions to prevent attempting to execute a RX or WX while the module is busy .
SP121	Com. error Slot 0	on when the communication module in slot 0 of the local base has encountered a communication error.
SP122	Module busy Slot 1	on when the communication module in slot 1 of the local base is busy transmitting or receiving. You must use this relay with the RX or WX instructions to prevent attempting to execute a RX or WX while the module is busy.
SP123	Com. error Slot 1	on when the communication module in slot 1 of the local base has encountered a communication error.
SP124	Module busy Slot 2	on when the communication module in slot 2 of the local base is busy transmitting or receiving. You must use this relay with the RX or WX instructions to prevent attempting to execute a RX or WX while the module is busy.
SP125	Com. error Slot 2	on when the communication module in slot 2 of the local base has encountered a communication error.
SP126	Module busy Slot 3	on when the communication module in slot 3 of the local base is busy transmitting or receiving. You must use this relay with the RX or WX instructions to prevent attempting to execute a RX or WX while the module is busy.
SP127	Com. error Slot 3	on when the communication module in slot 3 of the local base has encountered a communication error.
SP130	Module busy Slot 4	on when the communication module in slot 4 of the local base is busy transmitting or receiving. You must use this relay with the RX or WX instructions to prevent attempting to execute a RX or WX while the module is busy.
SP131	Com. error Slot 4	on when the communication module in slot 4 of the local base has encountered a communication error.
SP132	Module busy Slot 5	on when the communication module in slot 5 of the local base is busy transmitting or receiving. You must use this relay with the RX or WX instructions to prevent attempting to execute a RX or WX while the module is busy.
SP133	Com. error Slot 5	on when the communication module in slot 5 of the local base has encountered a communication error.
SP134	Module busy Slot 6	on when the communication module in slot 6 of the local base is busy transmitting or receiving. You must use this relay with the RX or WX instructions to prevent attempting to execute a RX or WX while the module is busy.
SP135	Com. error Slot 6	on when the communication module in slot 6 of the local base has encountered a communication error.
SP136	Module busy Slot 7	on when the communication module in slot 7 of the local base is busy transmitting or receiving. You must use this relay with the RX or WX instructions to prevent attempting to execute a RX or WX while the module is busy.
SP137	Com. error Slot 7	on when the communication module in slot 7 of the local base has encountered a communication error.

E

DL305
Product Weights

In This Appendix. . . .

Product Weight Table

CPUs	Weight
D3–330	6.3 oz. (178g)
D3–330P	6.3 oz. (178g)
D3–340	5.2 oz. (146g)
D3–350	4.9 oz. (140g)
Specialty CPUs	
F3–OMUX–1	6.4 oz. (182g)
F3–OMUX–2	6.4 oz. (182g)
F3–PMUX	3.7 oz. (104g)
F3–RTU	6.7 oz. (190g)
Bases	
D3–05B–1	37.0 oz. (1050g)
D3–05BDC–1	37.0 oz.(1050g)
D3–08B–1	44.1 oz.(1250g)
D3–10B–1	51.1 oz.(1450g)
D3–05B	34.0 oz. (964g)
D3–05BDC	34.0 oz.(964g)
D3–08B	44.2 oz.(1253g)
D3–10B	50.5 oz.(1432g)
DC Input Modules	
D3–08ND2	4.2 oz. (120g)
D3–16ND2–1	6.3 oz. (180g)
D3–16ND2–2	5.3 oz. (150g)
D3–16ND2F	6.3 oz. (180g)
F3–16ND3F	5.4 oz. (153g)
AC Input Modules	
D3–08NA–1	5 oz. (140g)
D3–08NA–2	5 oz. (140g)
D3–16NA	6.4 oz. (180g)
AC/DC Input Modules	
D3–08NE3	4.2 oz. (120g)
D3–16NE3	6 oz. (170g)

DC Output Modules	Weight
D3–08TD1	4.2 oz. (120g)
D3–08TD2	4.2 oz. (120g)
D3–16TD1–1	5.6 oz. (160g)
D3–16TD1–2	5.6 oz. (160g)
D3–16TD2	7.1 oz. (210g)
AC Output Modules	
D3–04TAS	6.4 oz. (180g)
D3–08TAS	N/A at press time
D3–08TA–1	7.4 oz. (210g)
D3–08TA–2	6.4 oz. (180g)
F3–16TA–1	6.2 oz. (176g)
D3–16TA–2	7.2 oz. (210g)
Relay Output Modules	
D3–08TR	7 oz. (200g)
F3–08TRS–1	8.9 oz. (252g)
F3–08TRS–2	9 oz. (255g)
D3–16TR	8.5 oz. (248g)
Analog Modules	
D3–04AD	7 oz. (200g)
F3–04ADS	6.9 oz. (195g)
F3–08AD	5.5 oz. (154g)
F3–08TEMP	5.2 oz. (147g)
F3–08THM–n	6 oz. (170g)
F3–16AD	5.4 oz. (152g)
D3–02DA	7 oz. (200g)
F3–04DA–1	6.3 oz. (180g)
F3–04DA–2	6.3 oz. (180g)
F3–04DAS	7 oz. (200g)

Communications and Networking	Weight
D3–232–DCU	15.0 oz. (427g)
D3–422–DCU	14.8 oz. (419g)
ASCII BASIC Modules	
F3–AB128–R	5.1 oz. (146g)
F3–AB128–T	6.2 oz. (175g)
F3–AB128	5.4 oz. (154g)
Specialty Modules	
D3–08SIM	3.0 oz. (85g)
D3–HSC	5.2 oz. (147g)
D3–PWU	13.0 oz. (368g)
D3–FILL	1oz. (30g)
Programming	
D3–HP	7.1 oz. (202g)
D3–HPP	7.2 oz. (204g)
D2–HPP	7.7 oz. (220g)

F

I/O Addressing Conventional Method

In This Appendix. . . .

- — Understanding Conventional I/O Numbering
- — Conventional Base Specifications
- — Local and Expansion I/O Systems
- — Setting the Base Switches
- — Example I/O Configurations

Understanding Conventional I/O Numbering

This Appendix covers the information needed when installing a DL350 CPU in an conventional base or when the DL350 is in a new base in a mixed system. Since the DL350 can be used in either scenario, both 16 bit and 8 bit addressing needs to be addressed. Chapter 4 provides the information on the xxxx–1 bases and the 16 bit addressing scheme. The DL350 CPU will revert to the DL340 CPU I/O scheme when it is configured for either of these scenarios.

DL305 I/O Configuration History

The conventional DL305 product family has had several enhancements over the years. Each time the product family has grown or has been enhanced, compatibility with the earlier products has been of the utmost concern. Some of these enhancements such as increasing the I/O count and supporting 16 point modules have impacted the numbering system. To help you understand the numbering scheme, the following account of how the numbering system has been affected is provided.

- When the 16 point I/O modules were introduced to the standard line of 8 point modules, the I/O numbering system was not modified to count in 16 consecutive units. This was done to maintain compatibility with the 8 point systems. This means each 16 point module uses two groups of eight consecutive numbers such as 000 through 007 and 100 through 107.
- When the I/O count was increased from the original 112 maximum to 176 maximum (DL330/DL330P CPU) to 184 maximum (DL340/DL350 CPU), most of the new I/O addresses were not set up to be consecutive with the the original 112 I/O. This means you will see a large jump in the I/O number ranges.

Octal Numbering System

The conventional DL305 I/O points are numbered in octal (base 8.) The octal numbering system does not include the numbers 8 and 9. The following table lists the first few octal numbers with the equivalent decimal numbers so you can see the numbering pattern.

Octal Numbers	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	21	22	23	24	...
Decimal Numbers	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...

Fixed I/O Numbering

The DL305 base I/O numbering is fixed, you cannot choose the I/O address of specific points since the system allocates the addresses for each slot. The I/O number ranges are 0–177 and 700–767. The I/O numbering for each slot in the base depends on two things:

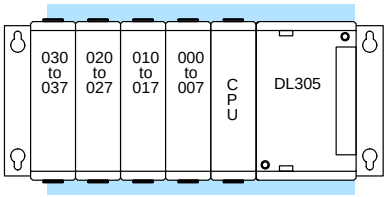
1. The base configuration, which is determined by the size of the base and whether you are using an expansion base.
2. The number of I/O points per module and the location of the I/O modules in the base.

I/O Numbering Guidelines

I/O numbering begins with address “000” which is the slot adjacent to the CPU. Each module uses increments of eight I/O points. For 8 point modules the I/O addresses are made up of eight contiguous points for each module. For 16 point modules the I/O addresses are made up of two groups of eight contiguous points, the first group follows the same scheme as the 8 point module and the second group adds 100 to the values of the first group.

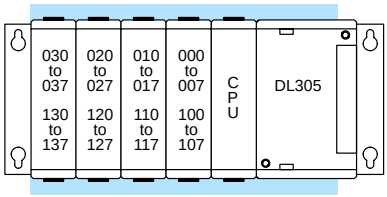
The examples below show the I/O numbering for a 5 slot local CPU base with 8 point I/O and a 5 slot local CPU base with 16 point I/O.

5 Slot Base Using 8 Point I/O Modules



Slot Number: 3—2—1—0

5 Slot Base Using 16 Point I/O Modules



Slot Number: 3—2—1—0

Number of I/O Points Required for Each Module

DC Input Modules	
D3–08ND2	8
D3–16ND2–1	16
D3–16ND2–2	16
D3–16ND2F	16
F3–16ND3	16
AC Input Modules	
D3–08NA–1	8
D3–08NA–2	8
D3–16NA	16
AC/DC Input Modules	
D3–08NE3	8
D3–16NE3	16

DC Output Modules	
D3–08TD1	8
D3–08TD2	8
D3–16TD1–1	16
D3–16TD1–2	16
D3–16TD2	16
AC Output Modules	
D3–04TAS	8*
F3–08TAS	8
D3–08TA–1	8
D3–08TA–2	8
F3–16TA–1	16
D3–16TA–2	16

Relay Output Modules	
D3–08TR	8
F3–08TRS–1	8
F3–08TRS–2	8
D3–16TR	16
Analog Modules	
D3–04AD	16
F3–04ADS	16
F3–08AD	16
F3–08TEMP	16
F3–08THM–n	16
F3–16AD	16
D3–02DA	16

Analog Modules (cont.)	
F3–04DA–1	16
F3–04DA–2	16
F3–04DAS	16
ASCII BASIC Modules	
F3–AB128–R	16
F3–AB128–T	16
F3–AB128	16
F3–AB64	16
Specialty Modules	
D3–08SIM	8
D3–HSC	16

* This is a 4-point module, but each slot is assigned a minimum of 8 I/O points.

I/O Module Placement Rules

There are some limitations that determine where you can place certain types of modules. Some modules require certain locations and may limit the number or placement of other modules. We have tried to give clearly written explanations of the rules governing module placement, but we realize a picture can sometimes be worth a thousand words. If you have difficulty with some of our explanations, please look ahead to the illustrations in this chapter. They should clear up any gray areas in the explanation and you will probably find the configuration you intend to use in your installation.

In all of the configurations mentioned the number of slots from the CPU that are to be used can roll over into an expansion base if necessary. For example if a rule states a module must reside in one of the six slots adjacent to the CPU, and the system configuration is comprised of two 5 slot bases, slots 1 and 2 of the expansion base are valid locations.

The following table provides the general placement rules for the DL305 components.

Module	Restriction
CPU	The CPU must reside in the first slot of the local CPU base. The first slot is the closest slot to the power supply.
16 Point I/O Modules	There can be a maximum of eight 16 point modules installed in a system depending on the CPU type and I/O modules used. The 16 point modules must be in the first 8 slots adjacent to the CPU rolling over into an expansion base if necessary. If any of the eight slots adjacent to the CPU are not used for 16 point modules, they can be used for 8 point modules.
Analog Modules	Analog modules must reside in any valid 16 point I/O slot.
ASCII Basic Modules	ASCII Basic modules must reside in any valid 16 point I/O slot.
High Speed Counter	High Speed Counters may be used in one of the first 4 slots in the local CPU base.

Conventional Base Specifications

The table below provides the specifications for the conventional DL305 bases. The xxxx–1 bases are covered in Chapter 2, Installation, Wiring, and Specifications.

	D3–05B	D3–05BDC	D3–08B	D3–10B
Number of Slots	5	5	8	10
Local CPU Base	Yes	Yes	Yes	Yes
Expansion Base	Yes	Yes	No	Yes
Input Voltage Range	97–132 VAC 194–264 VAC 47–63Hz	20.5–30 VDC <10% ripple	97–132 VAC 194–264 VAC 47–63Hz	97–132 VAC 194–264 VAC 47–63Hz
Base Power Consumption	70 VA max (46W)	48 Watts	70 VA max (57W)	70 VA max (57W)
Inrush Current max.	30A	30A	30A	30A
Dielectric Strength	1500VAC for 1 minute between terminals of AC P/S, Run output, Common, 24VDC	1500VAC for 1 minute between 24VDC input terminals and Run output	1500VAC for 1 minute between terminals of AC P/S, Run output, Common, 24VDC	2000VAC for 1 minute between terminals of AC P/S, Run output, Common, 24VDC
Insulation Resistance	>10MΩ at 500VDC	>10MΩ at 500VDC	>10MΩ at 500VDC	>10MΩ at 500VDC
Power Supply Output (Voltage Ranges and Ripple)	(5VDC) 4.75–5.25V less than 0.1V p–p (9VDC) 8.5–13.5V less than 0.2V p–p (24VDC) 20–28V less than 1.2V p–p	(5VDC) 4.75–5.25V less than 0.1V p–p (9VDC) 8.5–13.5V less than 0.2V p–p (24VDC) 20–28V less than 1.2V p–p	(5VDC) 4.75–5.25V less than 0.1V p–p (9VDC) 8.0–12.0V less than 0.2V p–p (24VDC) 20–28V less than 1.2V p–p	(5VDC) 4.75–5.25V less than 0.1V p–p (9VDC) 8.0–12.0V less than 0.2V p–p (24VDC) 20–28V less than 1.2V p–p
5 VDC current available	1.4A *	1.4A	1.4A @ 122° F (50° C) 1.0A @ 140° F (60° C)	1.4A @ 122° F (50° C) 1.0A @ 140° F (60° C)
9 VDC current available	0.8A *	0.8A	1.7A @ 122° F (50° C) 1.4A @ 140° F (60° C)	1.7A @ 122° F (50° C) 1.4A @ 140° F (60° C)
24 VDC current available	0.5A *	0.5A	0.6A	0.6A
Auxiliary 24 VDC Output	100mA max	None	100mA max	100mA max
Run Relay	250 VAC, 4A (resistive load)	250 VAC, 4A (resistive load)	250 VAC, 4A (resistive load)	250 VAC, 4A (resistive load)
Fuses	2A (250V) User replaceable	4A (250V) User replaceable	2A (250V) User replaceable	2A (250V) User replaceable
Dimensions WxHxD	11.42x4.85x4.41 in. (290x123x112 mm)	11.42x4.85x4.41 in (290x123x112 mm)	15.55x4.85x4.41 in (395x123x112 mm)	18.3x4.85x4.41 in. (465x123x112 mm)
Weight	34 oz. (964g)	34 oz. (964g)	44.2 oz. (1253g)	50.5 oz. (1432g)

* The total current for the D3–05B must not exceed 2.3A.

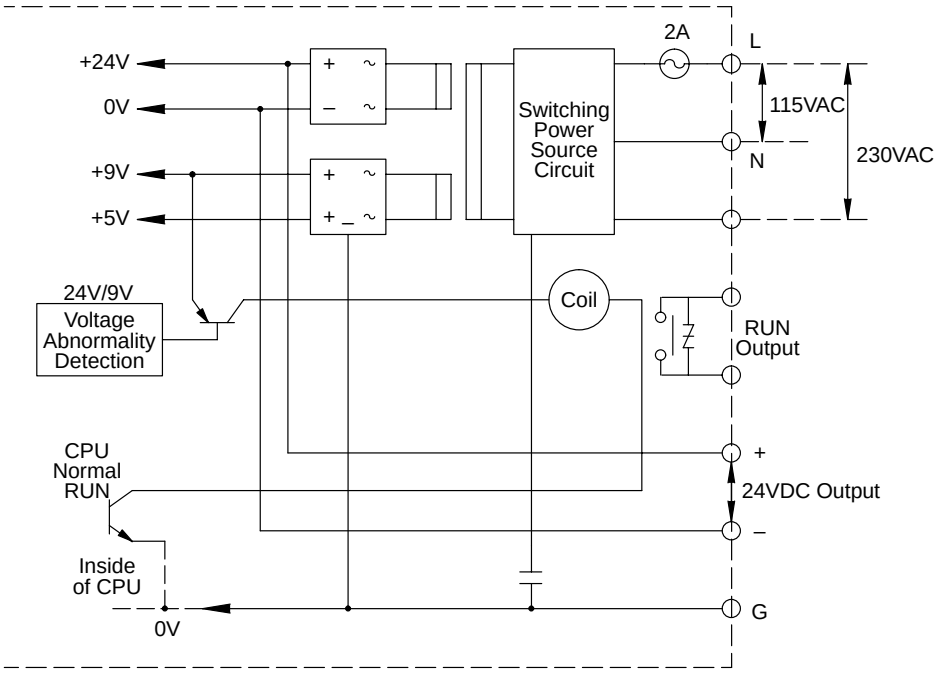
Auxiliary 24VDC Output at Base Terminal

There is 24 VDC available from the 24 VDC output terminals on all bases except the 5 slot DC version (D3–05BDC). The 24 VDC supply can be used to power external devices or DL305 modules that require external 24 VDC. The power used from the this 24 VDC output reduces the internal system 24 VDC available to the modules by an equal amount.

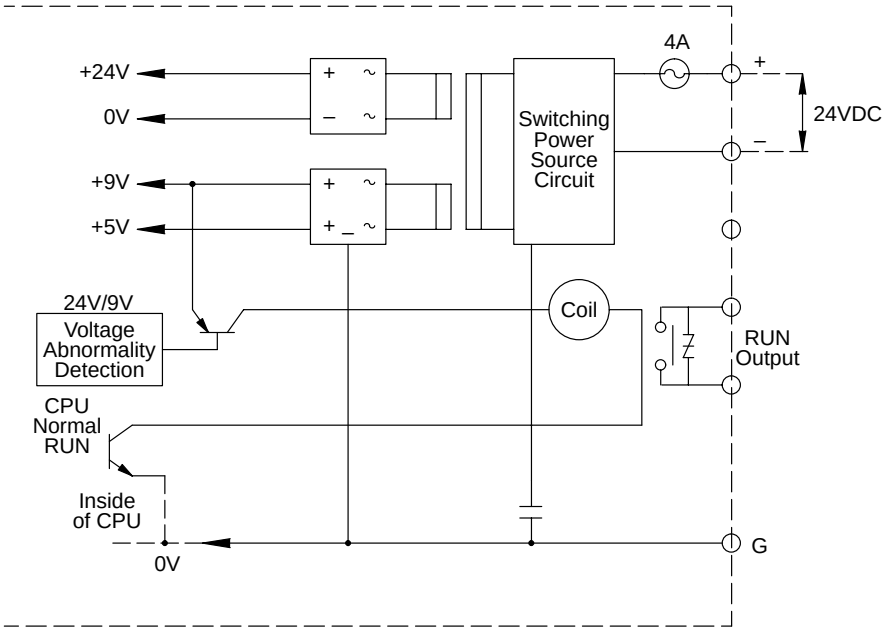
Power Supply Schematics

The following diagram shows the details of how the DL305 base provides many of the specifications listed on the previous page.

Schematic for D3–05B, D3–08B, D3–10B



Schematic for D3–05BDC

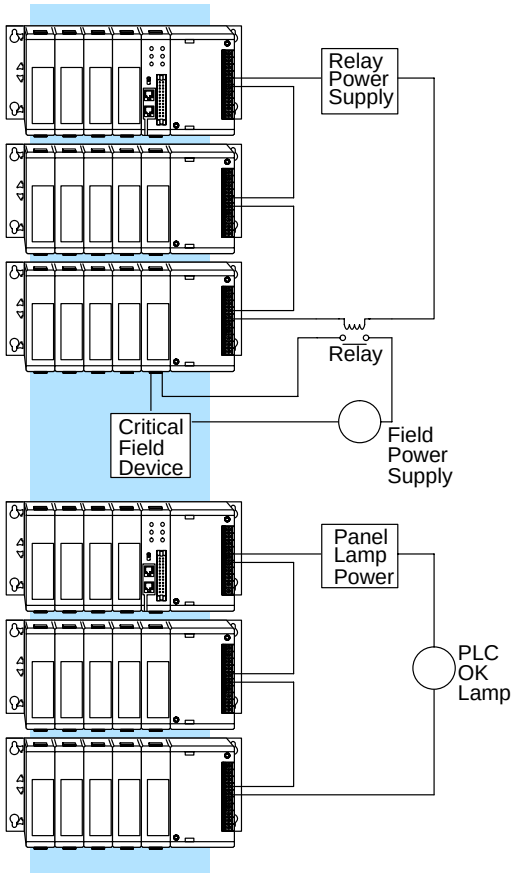


Using the Run Relay on the Base Power Supply

The RUN relay output, located on the DL305 base power supply, can be used to detect an undesired failure on the local CPU base or an expansion base. The following table shows the operating characteristics of the RUN relay for a local CPU base or an expansion base.

Event	Local CPU Base RUN Relay Would:	Expansion Base RUN Relay Would:
PROGRAM to RUN mode Transition	Energize	Not change
The CPU detects a fatal error	De–energize	Not change
CPU Local Base is Removed Form the RUN Mode	De–energize	Not change
Power Source to the Power Supply is Turned OFF	De–energize	De–energize
9 VDC or 24 VDC Failure on the Power Supply	De–energize	De–energize

The following example demonstrates possible uses for the RUN relay on the DL305 bases.



Use of the RUN relay to shutdown critical field devices upon error detection

Use of the RUN relay to monitor system operation

Local or Expansion I/O Systems

Base Uses Table

It is helpful to understand how you can use the various DL305 bases in your control system. The following table shows how the bases can be used.

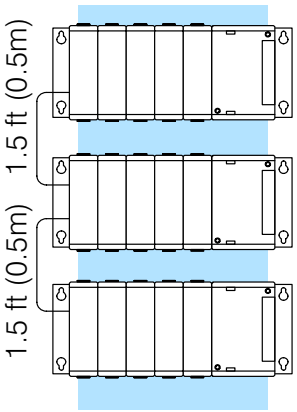
Base Part #	Number of Slots	Can Be Used As A Local CPU Base	Can Be Used As An Expansion Base
D3–05B	5	Yes	Yes
D3–05BDC	5	Yes	Yes
D3–08B	8	Yes	No
D3–10B	10	Yes	Yes

Local/Expansion Connectivity

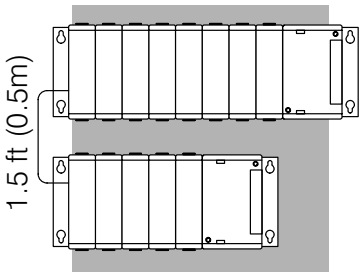
The configurations below show the valid combinations of local CPU bases and expansion bases.

NOTE: You should use one of the configurations listed below when designing an expansion system. If you use a configuration not listed below the system will not function properly.

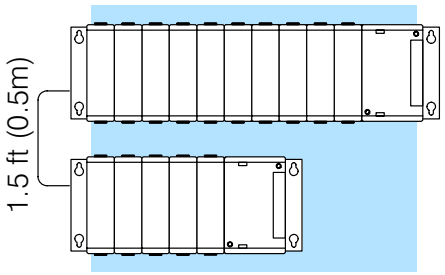
5 slot local CPU base with a maximum of two 5 slot expansion bases



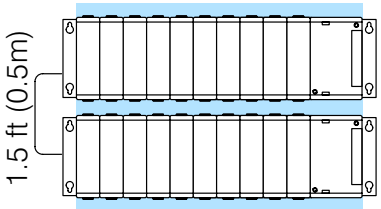
8 slot local CPU base with a 5 slot expansion base



10 slot local CPU base with a 5 slot expansion base



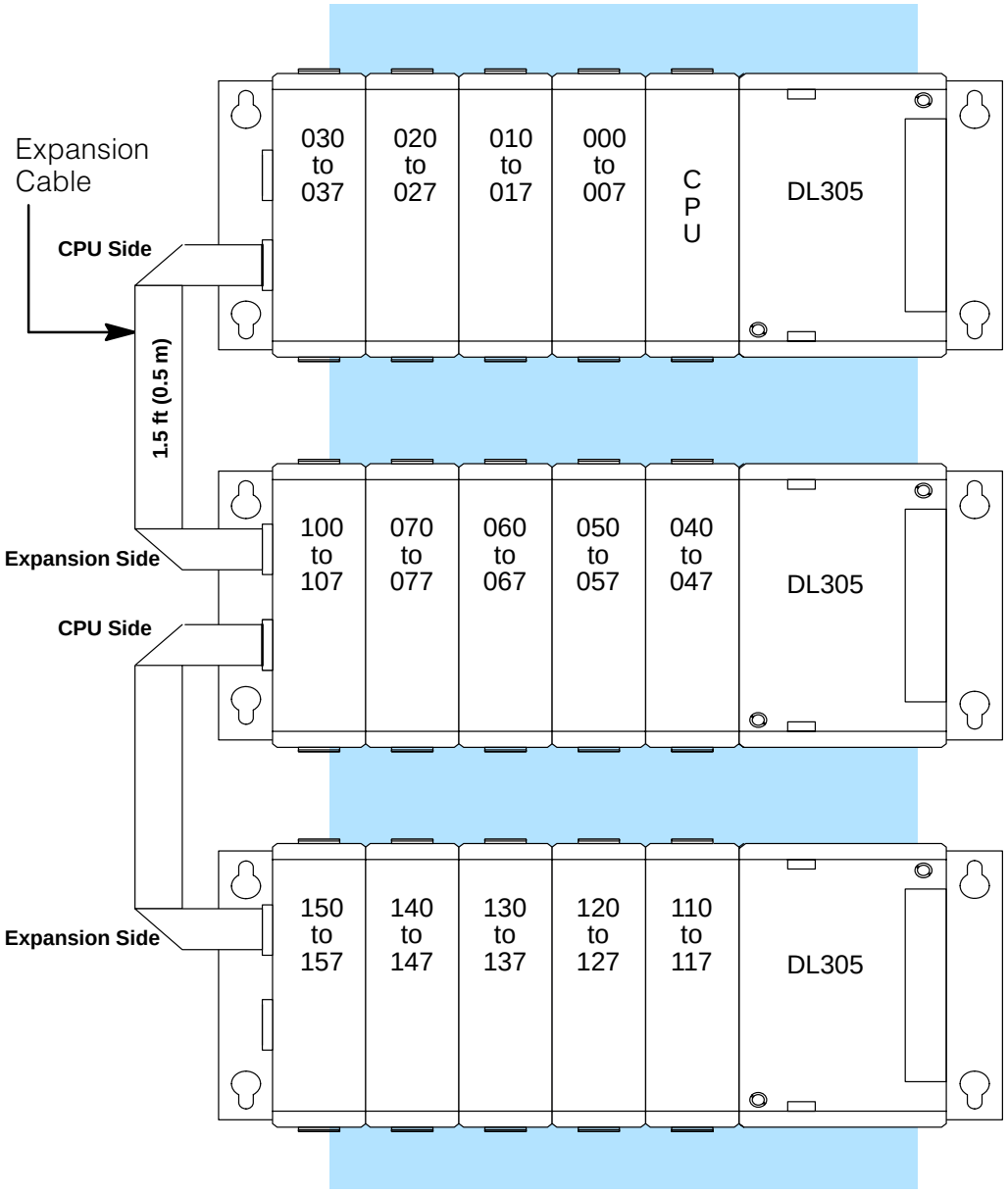
10 slot local CPU base with a 10 slot expansion base



Connecting Expansion Bases

The local CPU base is connected to the expansion base using a 1.5 ft. cable (D3–EXCBL). The base must be connected as shown in the diagram below.

The top expansion connector on the base is the input from a previous base. The bottom expansion connector on the base is the output to an expansion base. The expansion cable is marked with “CPU Side” and “Expansion Side”. The“ CPU Side” of the cable is connected to the bottom port of the base and the “Expansion Side” of the cable is connected to the top port of the next base.



Note: Avoid placing the expansion cable in the same wiring tray as the I/O and power source wiring.

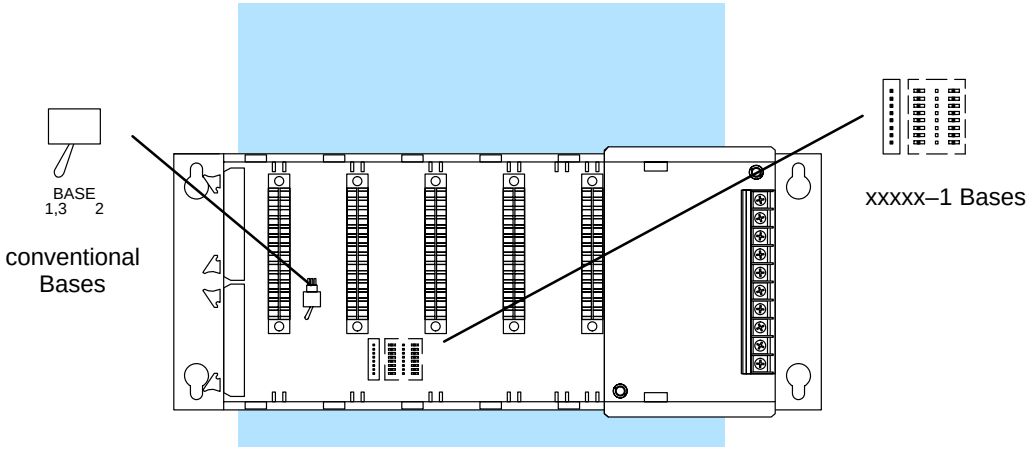
Setting the Base Switches

The conventional 5 slot and 10 slot bases have jumper switches that need to be set depending on which system configuration is used. The 8 slot base does not have any switches. All of the xxxxx–1 bases have a jumper switch and the 10 slot has two.

5 Slot Bases

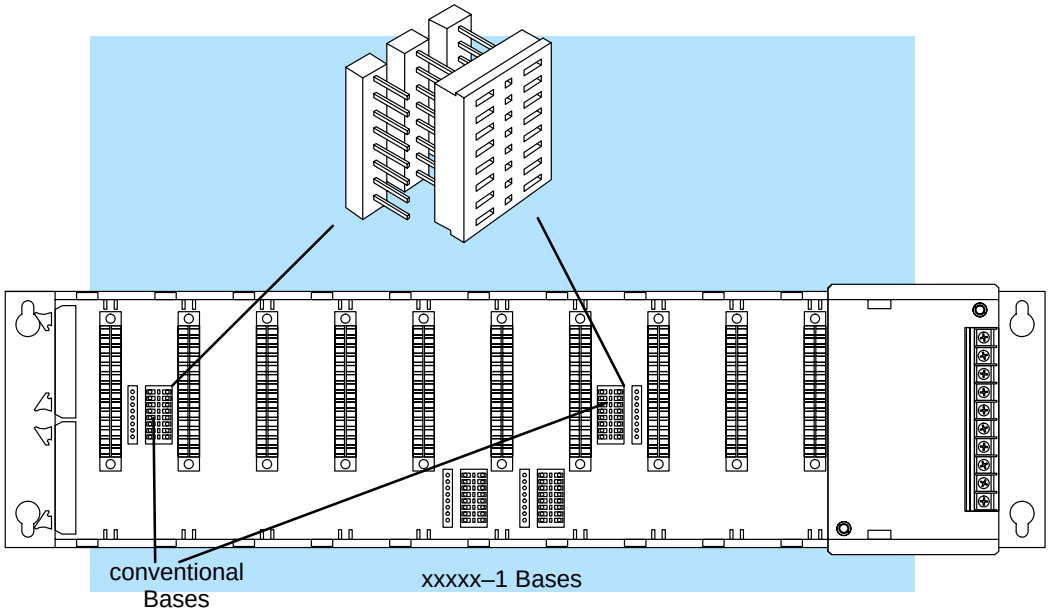
The conventional 5 slot bases have a two position toggle switch which is used to set the base as the CPU local base, the first expansion base, or the second (last) expansion base. The xxxxx–1 bases have a jumper switch between slots 3 and 4.

The switch is set to the “1,3” position if the base is the local CPU base or the third base in the system. The switch is set to the “2” position if the base is the 2nd base in the system. If the 5 slot base is used as an expansion base for a 10 slot local CPU base the switch is set in the “1,3” position.



10 Slot Base

The 10 slot base has a jumper switch between slot 3 and 4 used to set the base to local CPU base or expansion base. There is also a jumper switch between slot 9 and 10 (4 and 5 on the xxxxx–1 bases) that sets slot 10 to the 100–107 I/O address range or to the 700–707 I/O address range.

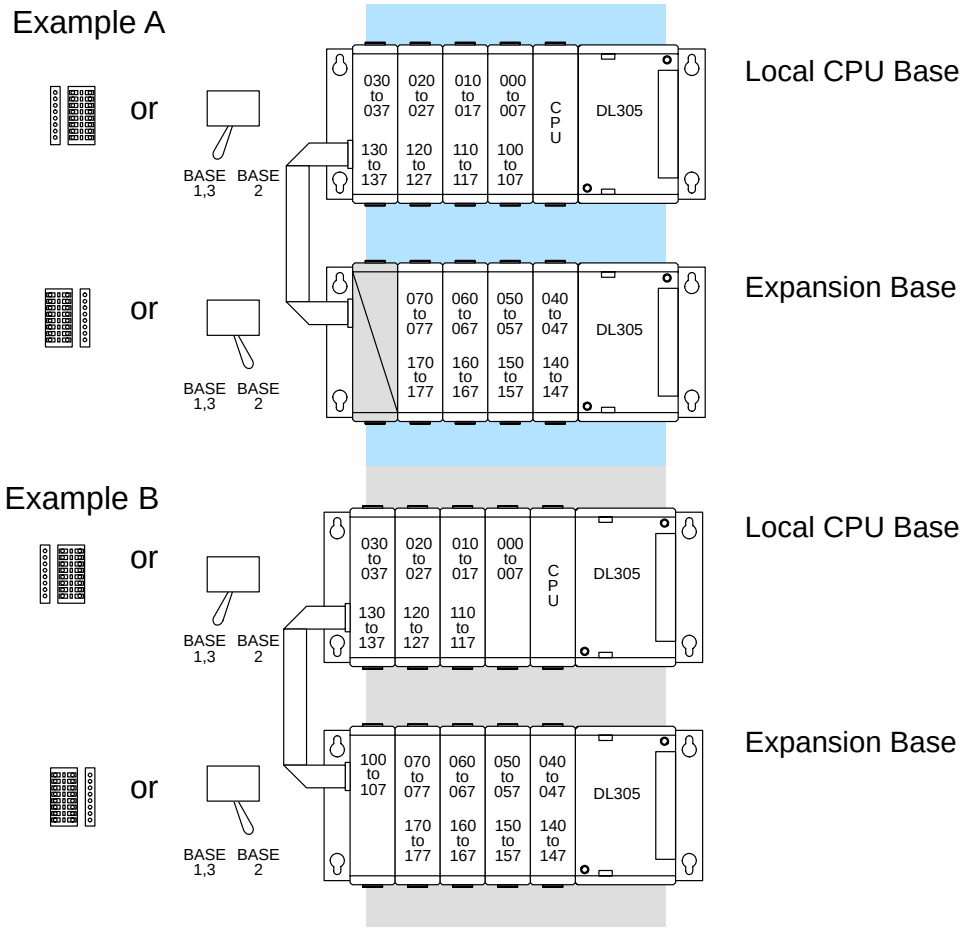


Example I/O Configurations

The following system configurations will allow you to quickly configure your system by using examples. These system configurations show the I/O numbering and the base switch settings for every valid base configuration for a DL305 system.

16 Point I/O Allocation Example

When a 16 point I/O module is used the last 8 I/O addresses of each 16 point module could have been used in another base slot. In the illustration below Example A shows a 16 point module in the slot next to the CPU using address 000–007 and 100–107. The expansion I/O cannot use the last slot of the expansion base since it is assigned addresses 100–107 and the 16 point module next to the CPU has already used these addresses. Example B shows an 8 point module in the slot next to the CPU and an 8 point module in the last slot of the expansion base. Both examples are valid configurations .



Examples Show Maximum I/O Points Available

For the following examples the configurations using 16 point I/O modules are shown with the maximum I/O points supported so you can always reduce the I/O count in one of our examples and the configuration will still be valid. Substitution of 8 point I/O modules can be made in place of any of the 16 point modules without affecting the I/O numbering for any of the other I/O modules. When a 16 point module is replaced with a 8 point I/O module the last 8 I/O addresses of that 16 point module may or may not be useable in another slot location, depending on the system configuration used

I/O Configurations with a 5 Slot Local CPU Base

The configurations below and on the next few pages show a 5 slot base with 8 point I/O modules, 16 point modules, one expansion base and two expansion bases.

Switch settings

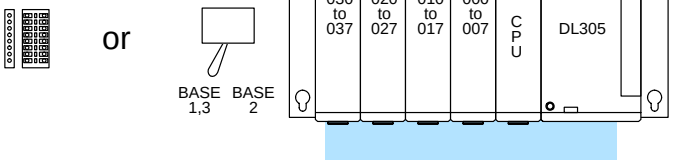
The 5 slot base has a toggle switch or jumper on the inside of the base which allows you to select:

Type of Base	**Switch Position** convent. bases	**Jumper Position** xxxxx–1 bases
Local CPU	Base 1,3	right pins bridged
First Expansion	Base 2*	left pins bridged
Last Expansion	Base 1,3	right pins bridged

*used only with a 5 slot local CPU base

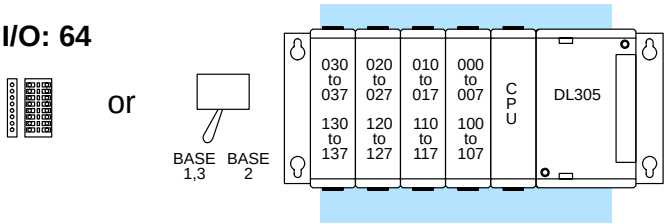
5 Slot Base with 8 Point I/O

Total I/O: 32



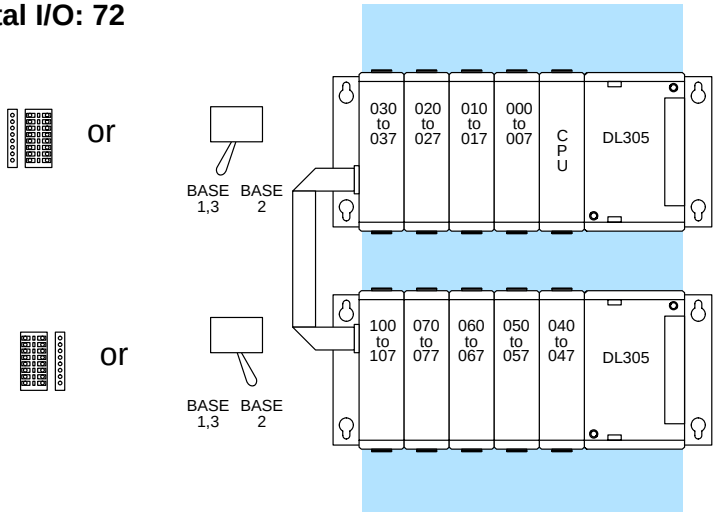
5 Slot Base with 16 Point I/O

Total I/O: 64



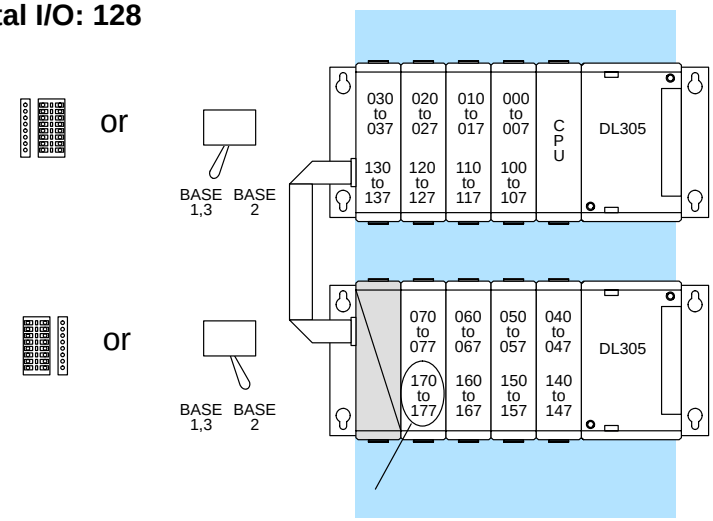
5 Slot Base and 5 Slot Expansion Base with 8 Point I/O

Total I/O: 72



5 Slot Base and 5 Slot Expansion Base with 16 Point I/O

Total I/O: 128

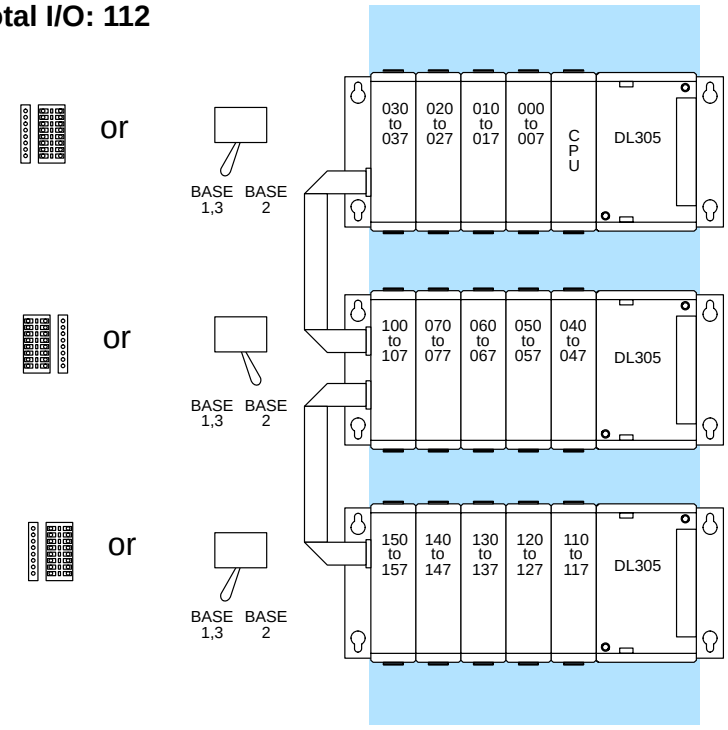


DL340 and DL350

NOTE: If a 16pt module is used in the last two available slots of the expansion base, 160 through 177 will not be available for control relay assignments. Also, even though you are using these points as I/O, you still enter them as C160–C177 in ***Direct***SOFT.

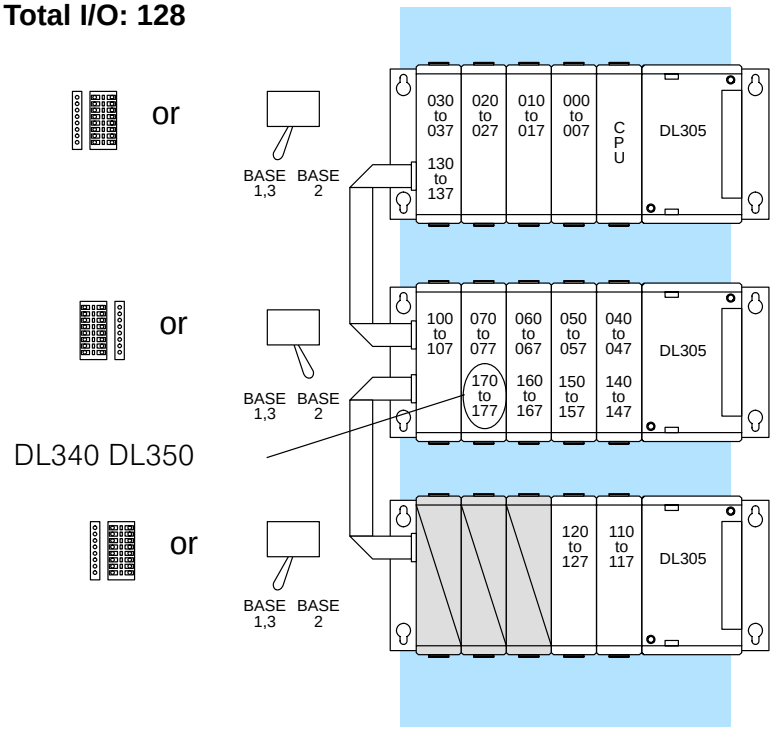
5 Slot Base and Two 5 Slot Expansion Bases with 8 Point I/O

Total I/O: 112



5 Slot Base and Two 5 Slot Expansion Bases with 16 and 8 Point I/O

Total I/O: 128



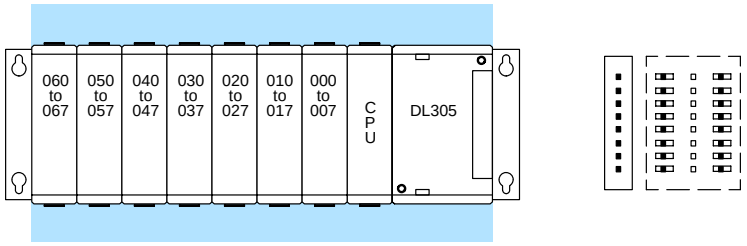
NOTE: If a 16pt modules are used in the second expansion base as shown, 160 through 177 will not be available for control relay assignments. Also, even though you are using these points as I/O, you still enter them as C160–C177 in ***Direct***SOFT.

I/O Configurations with an 8 Slot Local CPU Base

The configurations below show an 8 slot base with 8 point I/O modules, 16 point modules, one 5 slot expansion base and two 5 slot expansion bases. Postion of the jumper for xxxx–1 bases is shown to the right of the base.

8 Slot Base with 8 Point I/O

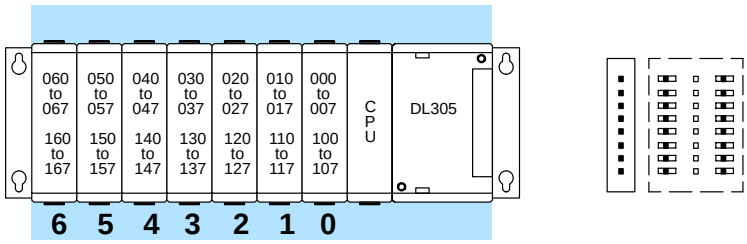
Total I/O: 56



8 Slot Base with 16 Point I/O

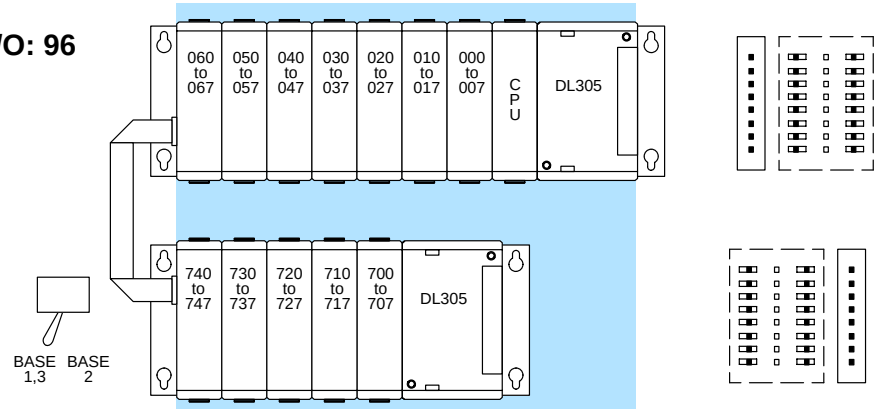
Total I/O: 112

***See note below regarding points 160–167**



8 Slot Base and 5 Slot Expansion Base with 8 Point I/O

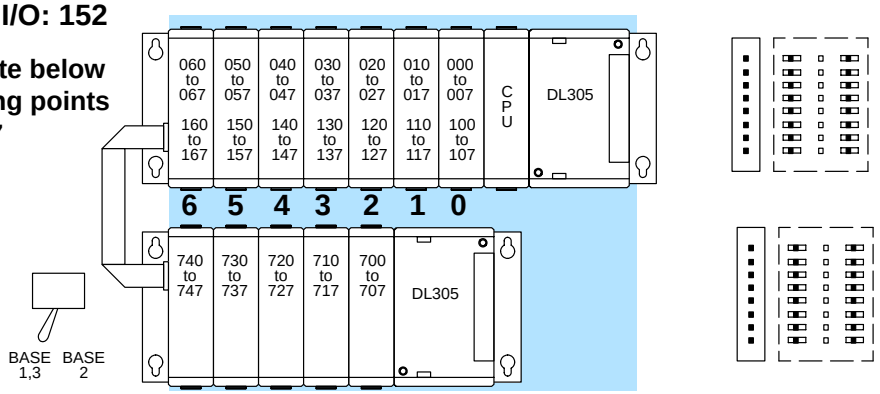
Total I/O: 96



8 Slot Base and 5 Slot Expansion Base with 16 Point I/O

Total I/O: 152

***See note below regarding points 160–167**



NOTE: If a 16pt module is used in Slot 6, 160 through 167 will not be available for control relay assignments. Also, even though you are using these points as I/O, you still enter them as C160–C167 in ***Direct***SOFT.

I/O Configurations with a 10 Slot Local CPU Base

The configurations below and on the next few pages show a 10 slot base with 8 point I/O modules, with 16 point modules, with a 5 slot expansion base and with a 10 slot expansion base.

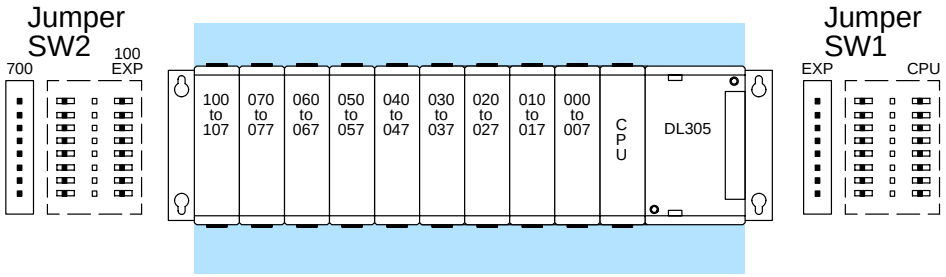
Switch settings

The 10 slot base has two jumper switches to select the base type and the address ranges to use. These switches can be found on the base between slots 3 and 4 (SW1) and slots 9 and 10 (SW2). Jumper switch SW1 is used to select if the base is a local CPU base or an expansion base. Jumper switch SW2 determines the I/O address range (100 – 107 or 700 – 707) for the 10th slot on the local CPU base. By selecting the address range of 700 to 707 for slot 10, it is possible to use a 16 point module next to the CPU (which uses the ranges of 000 to 007 and 100 to 107), however; the position of this switch will affect the I/O numbering for the expansion I/O if used.

NOTE: Jumper switch SW2 must be set to “100 EXP” on the expansion base.

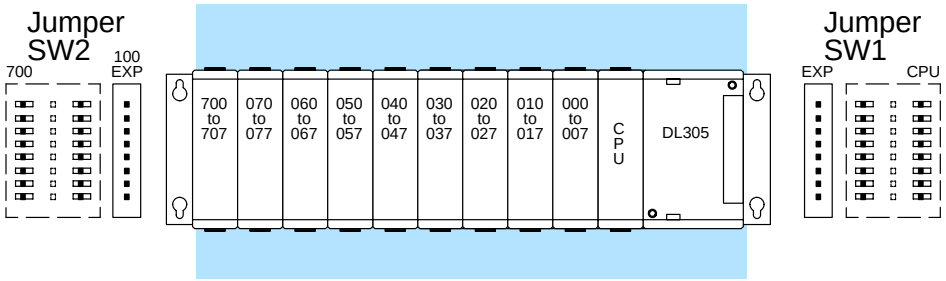
Last Slot Address Range 100 to 107

Total I/O: 72



Last Slot Address Range 700 to 707

Total I/O: 72



10 Slot Expansion Base with 16 Point I/O

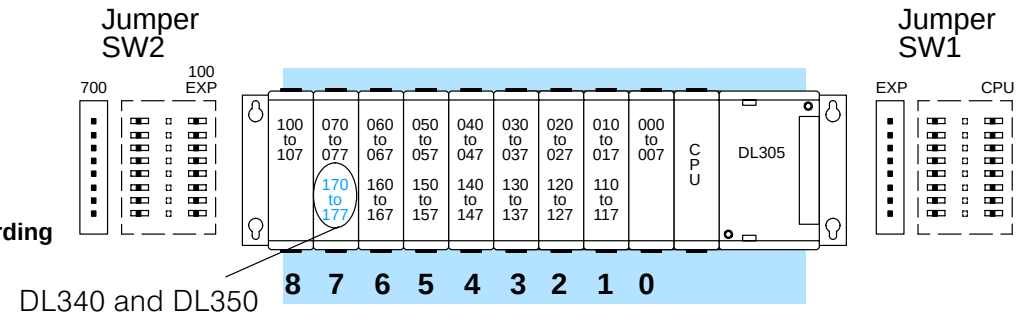
The next two configurations show a local CPU base using 16 point I/O modules and the two possibilities for how to configure the base to use the maximum number of I/O points.

Configuration 1

Configuration 1 shows an 8 point I/O module the slot next to the CPU and the address range of 100–107 for the last slot. When jumper switch SW2 is set to the "100 EXP" position, the address range for the last slot is set to 100–107, thereby limiting the address range for the first module to 000–007. This means if you use this configuration, the first module must be an 8 point I/O module. You will have more available addresses for an expansion base as you will see in the example using a 10 slot expansion base.

Total I/O:128

Configuration 1



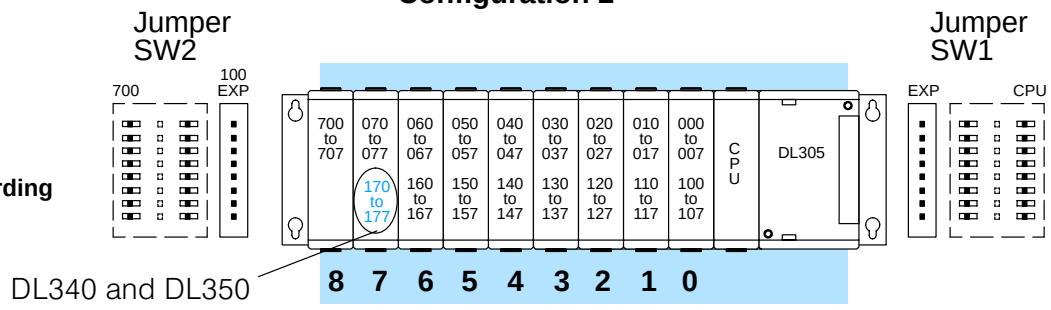
***See note below regarding points 160–167 and 170–177.**

Configuration 2

Configuration 2 shows a 16 point I/O module in the slot next to the CPU and the address range of 700–707 for the last slot. This is the maximum I/O configuration for a 10 slot local CPU base. When jumper switch SW2 is set to the "700" position the address range for the last slot is set to 700–707 making addresses 000–007 and 100–107 available for use in the first slot. The position of jumper switch SW2 can limit the amount of I/O addresses available to the larger expansion bases since expansion I/O numbering would normally start with address 700.

Total I/O: 136

Configuration 2

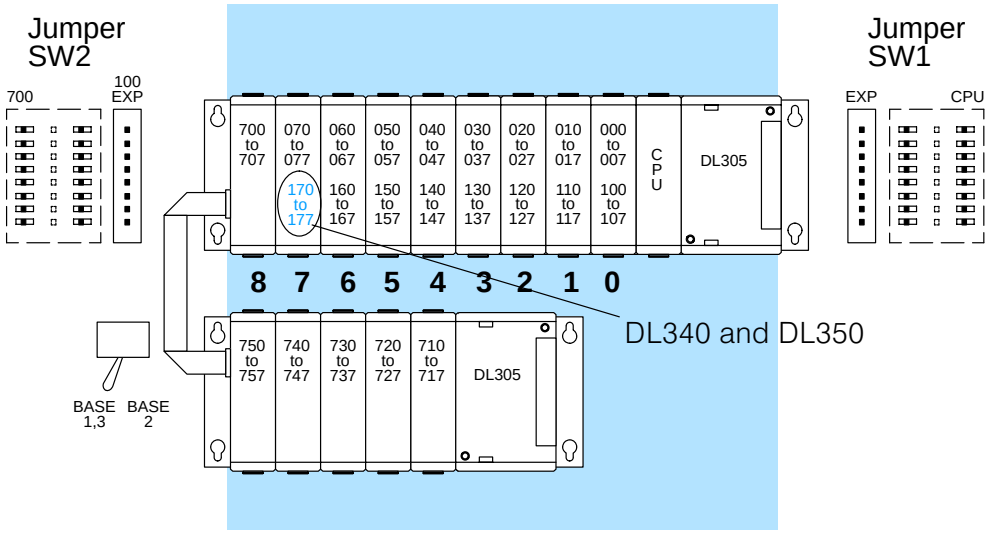


***See note below regarding points 160–167 and 170–177.**

NOTE: If a 16pt module is used in Slot 6 for the DL330 or DL330P CPU, 160 through 167 will not be available for control relay assignments. If a 16pt module is used in Slot 6 and/or Slot 7 for a DL340 or DL350 CPU, 160–167 and/or 170–177 are not available for control relay assignments. Also, even though you are using these points as I/O, you still enter them as C160–C167/C170–C177 in ***Direct***SOFT.

10 Slot Base and 5 Slot Expansion Base with 16 Point I/O

Total I/O: 176



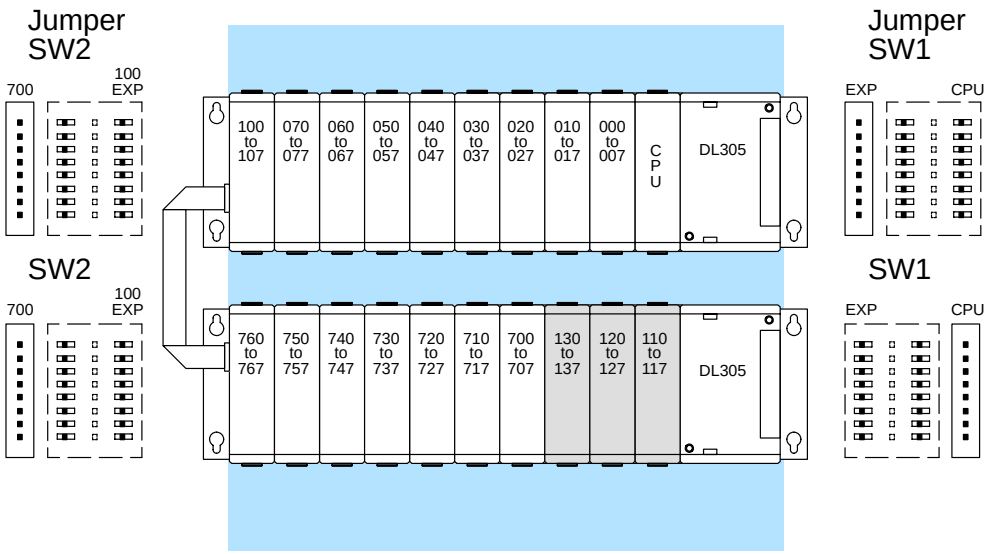
NOTE: If a 16pt module is used in Slot 6 for the DL330 or DL330P CPU, 160 through 167 will not be available for control relay assignments. If a 16pt module is used in Slot 6 and/or Slot 7 for a DL340 or DL350 CPU, 160–167 and/or 170–177 are not available for control relay assignments. Also, even though you are using these points as I/O, you still enter them as C160–C167/C170–C177 in ***Direct***SOFT.

Expansion Addresses Depend on Local CPU Base Configuration.

I/O addresses change depending on the point configuration in the local CPU base. Notice, when the local CPU base contains only 8 point I/O modules, addresses 110–117, 120–127 and 130–137 are available for use in the expansion base. When the local CPU base has 16 point I/O modules, which use the I/O addresses 110–117, 120–127 and 130–137, these addresses are taken up and are not available for use in the expansion base.

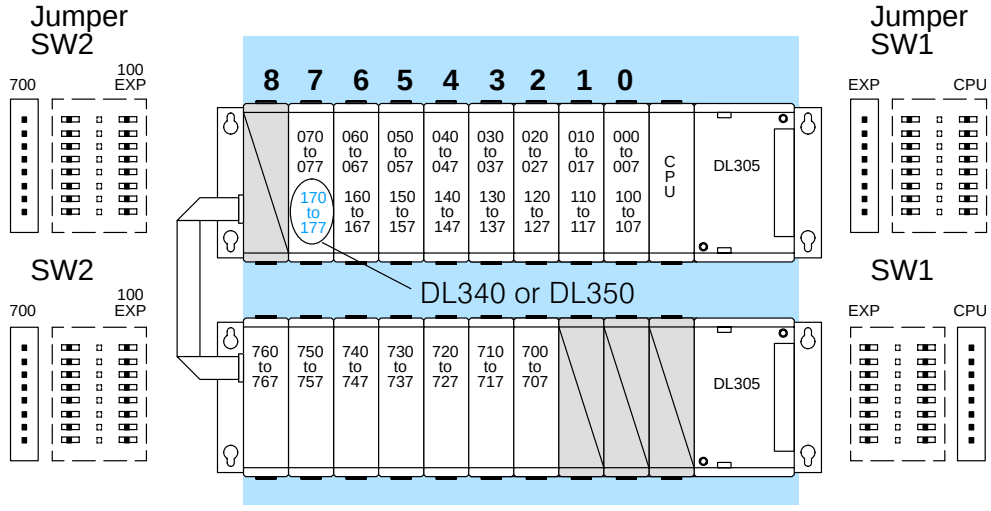
10 Slot Base and 10 Slot Expansion Base with 8 Point I/O

Total I/O: 152



10 Slot Base and 10 Slot Expansion Base with 16 Point I/O

Total I/O: 184



NOTE: If a 16pt module is used in Slot 6 for the DL330 or DL330P CPU, 160 through 167 will not be available for control relay assignments. If a 16pt module is used in Slot 6 and/or Slot 7 for a DL340 or DL350 CPU, 160–167 and/or 170–177 are not available for control relay assignments. Also, even though you are using these points as I/O, you still enter them as C160–C167/C170–C177 in ***Direct***SOFT.

Index

E

F

G

I

J

L

M

N

O

P

R

S

T

V

W