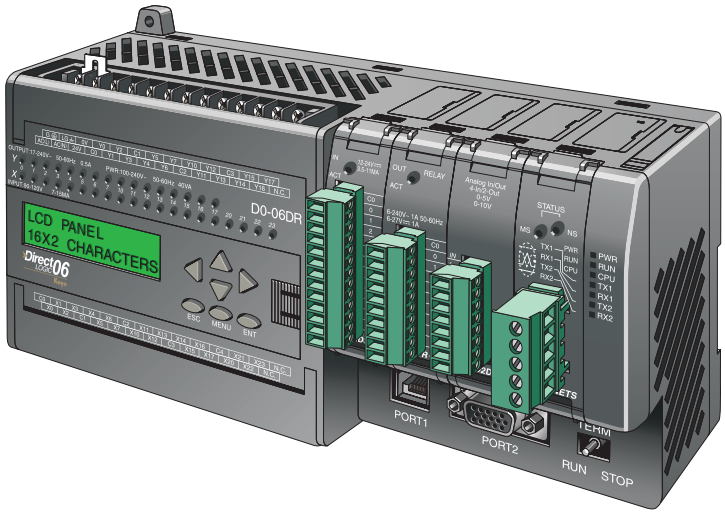
DL06 User Manual

Manual Number: D0-06USER-M

Volume 2 of 2



WARNING

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

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

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

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

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

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

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

Trademarks

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

AVERTISSEMENT

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

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

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

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

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

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

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

Marques de commerce

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

DL06 Micro PLC USER MANUAL

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

Manual Number: D0-06USER-M
Issue: Second Edition
Issue Date: 3/04

Publication History		
Issue	**Date**	**Description of Changes**
First Edition	7/02	Original
Rev. A	10/02	Updated drawing images and made minor corrections.
Rev. B	6/03	Added new PLC and made numerous corrections.
Second Edition	3/04	Added two appendices, removed discrete module data and made numerous corrections

Volume Two:
Table of Contents

Chapter 6: Drum Instruction Programming

Introduction **6–2**
Purpose 6–2
Drum Terminology 6–2
Drum Chart Representation 6–3
Output Sequences 6–3

Step Transitions **6–4**
Drum Instruction Types 6–4
Timer-Only Transitions 6–4
Timer and Event Transitions 6–5
Event-Only Transitions 6–6
Counter Assignments 6–6
Last Step Completion 6–7

Overview of Drum Operation **6–8**
Drum Instruction Block Diagram 6–8
Powerup State of Drum Registers 6–9

Drum Control Techniques **6–10**
Drum Control Inputs 6–10
Self-Resetting Drum 6–11
Initializing Drum Outputs 6–11
Using Complex Event Step Transitions 6–11

Drum Instruction **6–12**
Timed Drum with Discrete Outputs (DRUM) 6–12
Event Drum (EDRUM) 6–14
Handheld Programmer Drum Mnemonics 6–16
Masked Event Drum with Discrete Outputs (MDRMD) 6–19
Masked Event Drum with Word Output (MDRMW) 6–21

Chapter 7: RLLPLUS Stage Programming

Introduction to Stage Programming 7–2
Overcoming "Stage Fright" 7–2

Learning to Draw State Transition Diagrams 7–3
Introduction to Process States 7–3
The Need for State Diagrams 7–3
A 2–State Process 7–3
RLL Equivalent 7–4
Stage Equivalent 7–4
Let's Compare 7–5
Initial Stages 7–5
What Stage Bits Do 7–6
Stage Instruction Characteristics 7–6

Using the Stage Jump Instruction for State Transitions 7–7
Stage Jump, Set, and Reset Instructions 7–7

Stage Program Example: Toggle On/Off Lamp Controller 7–8
A 4–State Process 7–8

Four Steps to Writing a Stage Program 7–9
1. Write a Word Description of the application. 7–9
2. Draw the Block Diagram. 7–9
3. Draw the State Transition Diagram. 7–9
4. Write the Stage Program. 7–9

Stage Program Example: A Garage Door Opener 7–10
Garage Door Opener Example 7–10
Draw the Block Diagram 7–10
Draw the State Diagram 7–11
Add Safety Light Feature 7–12
Modify the Block Diagram and State Diagram 7–12
Using a Timer Inside a Stage 7–13
Add Emergency Stop Feature 7–14
Exclusive Transitions 7–14

Stage Program Design Considerations 7–15
Stage Program Organization 7–15
How Instructions Work Inside Stages 7–16
Using a Stage as a Supervisory Process 7–17
Stage Counter 7–17

Power Flow Transition Technique 7–18
Stage View in DirectSOFT32 7–18

Parallel Processing Concepts 7–19
Parallel Processes 7–19
Converging Processes 7–19
Convergence Stages (CV) 7–19
Convergence Jump (CVJMP) 7–20
Convergence Stage Guidelines 7–20

RLLPLUS (Stage) Instructions 7–21
Stage (SG) 7–21
Initial Stage (ISG) 7–22
Jump (JMP) 7–22
Not Jump (NJMP) 7–22
Converge Stage (CV) and Converge Jump (CVJMP) 7–23
Block Call (BCALL) 7–25
Block (BLK) 7–25
Block End (BEND) 7–25

Questions and Answers about Stage Programming 7–27

Chapter 8: PID Loop Operation

DL06 PID Loop Features 8–2
Main Features 8–2
The Basics of PID Loops 8–4

Loop Setup Parameters 8–6
Loop Table and Number of Loops 8–6
PID Error Flags 8–6
Establishing the Loop Table Size and Location 8–7
Loop Table Word Definitions 8–8
PID Mode Setting 1 Bit Descriptions (Addr + 00) 8–9
PID Mode Setting 2 Bit Descriptions (Addr + 01) 8–10
Mode / Alarm Monitoring Word (Addr + 06) 8–11
Ramp / Soak Table Flags (Addr + 33) 8–11
Ramp/Soak Table Location (Addr + 34) 8–12
Ramp/Soak Table Programming Error Flags (Addr + 35) 8–12

Loop Sample Rate and Scheduling 8–13
Loop Sample Rates Addr + 07 8–13

Choosing the Best Sample Rate 8–13
Determining a suitable sample rate (Addr+07): 8–14
Programming the Sample Rate 8–14
PID Loop Effect on CPU Scan Time 8–15

Ten Steps to Successful Process Control 8–17

Basic Loop Operation 8–19
Data Locations 8–19
Data Sources 8–19
Auto Transfer to Analog I/O 8–20
PV Auto Transfer Functions with Filtering Options 8–21
The built-in filter uses the following algorithm : 8–21
PV Auto Transfer (Addr + 36) from I/O Module Base/Slot/Channel Option 8–22
PV Auto Transfer (Addr + 36) from V–memory Option 8–22
Loop Modes 8–23
CPU Modes and Loop Modes 8–24
How to Change Loop Modes 8–25
Operator Panel Control of PID Modes 8–26
PLC Modes' Effect on Loop Modes 8–26
Loop Mode Override 8–26
Bumpless Transfers 8–27

PID Loop Data Configuration 8–28
Loop Parameter Data Formats 8–28
Choosing Unipolar or Bipolar Format 8–28
Handling Data Offsets 8–29
Setpoint (SP) Limits 8–29
Remote Setpoint (SP) Location 8–30
Process Variable (PV) Configuration 8–30
Control Output Configuration 8–31
Error Term Configuration 8–32

PID Algorithms 8–33
Position Algorithm 8–33
Velocity Algorithm 8–34
Direct-Acting and Reverse-Acting Loops 8–35
P-I-D Loop Terms 8–36
Using a Subset of PID Control 8–37
Derivative Gain Limiting 8–38
Bias Term 8–38

Bias Freeze 8–39

Loop Tuning Procedure 8–40
Open-Loop Test 8–40
Manual Closed Loop Tuning Procedure 8–41
Auto Tuning Procedure 8–42
Tuning Cascaded Loops 8–46

PV Analog Filter 8–47
The DL06 Built-in Analog Filter 8–47
Creating an Analog Filter in Ladder Logic 8–48

Feedforward Control 8–49
Feedforward Example 8–50
Time-Proportioning Control 8–51
On/Off Control Program Example 8–52

Cascade Control 8–53
Introduction 8–53
Cascaded Loops in the DL06 CPU 8–54
Process Alarms 8–55
PV Absolute Value Alarms 8–56
PV Deviation Alarms 8–56
PV Rate-of-Change Alarm 8–57
PV Alarm Hysteresis 8–58
Alarm Programming Error 8–58

Ramp/Soak Generator 8–59
Introduction 8–59
Ramp/Soak Table 8–60
Ramp/Soak Table Flags 8–62
Ramp/Soak Generator Enable 8–62
Ramp/Soak Controls 8–62
Ramp/Soak Profile Monitoring 8–63
Ramp/Soak Programming Errors 8–63
Testing Your Ramp/Soak Profile 8–63

Troubleshooting Tips 8–64

Bibliography 8–65

Glossary of PID Loop Terminology 8–66

Chapter 9: Maintenance and Troubleshooting

Hardware System Maintenance 9–2
Standard Maintenance 9–2

Diagnostics 9–2
Diagnostics 9–2
Fatal Errors 9–2
Non-fatal Errors 9–2
V-memory Error Code Locations 9–3
Special Relays (SP) Corresponding to Error Codes 9–3
DL06 Micro PLC Error Codes 9–4
Program Error Codes 9–5

CPU Indicators 9–6
RUN Indicator 9–7
CPU Indicator 9–7

Communications Problems 9–7

I/O Point Troubleshooting 9–8
Possible Causes 9–8
Some Quick Steps 9–8
Handheld Programmer Keystrokes Used to Test an Output Point 9–9

Noise Troubleshooting 9–10
Electrical Noise Problems 9–10
Reducing Electrical Noise 9–10

Machine Startup and Program Troubleshooting 9–11
Syntax Check 9–11
Special Instructions 9–12
Duplicate Reference Check 9–13
Run Time Edits 9–14
Run Time Edit Example 9–15
Forcing I/O Points 9–16
Regular Forcing with Direct Access 9–18
Bit Override Forcing 9–19
Bit Override Indicators 9–19

Chapter 10: LCD Display Panel

Introduction to the DL06 LCD Display Panel 10–2

Keypad **10–2**
Snap-in installation **10–3**
Display Priority **10–4**
Menu Navigation **10–5**
Confirm PLC Type, Firmware Revision Level, Memory Usage, Etc. **10–6**
Examining Option Slot Contents **10–8**
Menu 2, M2:SYSTEM CFG. 10–8
Menu 3, M3:MONITOR 10–10
Monitoring and Changing Data Values **10–10**
Data Monitor 10–10
V-memory values 10–10
Pointer values 10–12
Bit Monitor **10–13**
Bit status 10–13
Changing Date and Time **10–14**
Menu 4, M4 : CALENDAR R/W 10–14
Setting Password and Locking **10–17**
Menu 5, M5 : PASSWORD R/W 10–17
Reviewing Error History **10–20**
Menu 6, M6 : ERR HISTORY 10–20
Toggle Light and Beeper, Test Keys **10–21**
Menu 7, M7 : LCD TEST&SET 10–21
PLC Memory Information for the LCD Display **10–22**
Data Format Suffixes for Embedded V-memory Data 10–22
Reserved memory registers for the LCD Display Panel 10–23
V7742 bit definitions 10–24
Changing the Default Screen **10–25**
Example program for setting the default screen message 10–25
DL06 LCD Display Panel Instruction (LCD) **10–26**
Source of message 10–26
ASCII Character Codes 10–27
Example program: alarm with embedded date/time stamp 10–28
Example program: alarm with embedded V-memory data 10–29

Example program: alarm text from V-memory with embedded V-memory data .10–30

Appendix A: Auxiliary Functions

Introduction A–2
Purpose of Auxiliary Functions A–2
Accessing AUX Functions via DirectSOFT32 . . . . . A–3
Accessing AUX Functions via the Handheld Programmer A–3
AUX 2* — RLL Operations A–4
AUX 21 Check Program A–4
AUX 22 Change Reference A–4
AUX 23 Clear Ladder Range A–4
AUX 24 Clear Ladders A–4
AUX 3* — V-memory Operations A–4
AUX 31 Clear V Memory A–4
AUX 4* — I/O Configuration A–4
AUX 41 Show I/O Configuration A–4
AUX 5* — CPU Configuration A–5
AUX 51 Modify Program Name A–5
AUX 53 Display Scan Time A–5
AUX 54 Initialize Scratchpad A–5
AUX 55 Set Watchdog Timer A–5
AUX 56 CPU Network Address A–5
AUX 57 Set Retentive Ranges A–6
AUX 58 Test Operations A–6
AUX 59 Bit Override A–6
AUX 5B Counter Interface Configuration A–7
AUX 5D Select PLC Scan Mode A–7
AUX 6* — Handheld Programmer Configuration A–8
AUX 61 Show Revision Numbers A–8
AUX 62 Beeper On/Off A–8
AUX 65 Run Self Diagnostics A–8
AUX 7* — EEPROM Operations A–8
Transferrable Memory Areas A–8
AUX 71 CPU to HPP EEPROM A–8
AUX 72 HPP EEPROM to CPU A–9

AUX 73 Compare HPP EEPROM to CPU A–9
AUX 74 HPP EEPROM Blank Check A–9
AUX 75 Erase HPP EEPROM A–9
AUX 76 Show EEPROM Type A–9

AUX 8* — Password Operations A–9
AUX 81 Modify Password A–9
AUX 82 Unlock CPU A–10
AUX 83 Lock CPU A–10

Appendix B: DL06 Error codes

DL06 Error Codes B–2

Appendix C: Instruction Execution Times

Introduction C–2
V-Memory Data Registers C–2
V-Memory Bit Registers C–2
How to Read the Tables C–2

Instruction Execution Times C–3
Boolean Instructions C–3
Comparative Boolean Instructions C–4
Bit of Word Boolean Instructions C–13
Immediate Instructions C–14
Timer, Counter and Shift Register C–14
Accumulator Data Instructions C–16
Logical Instructions C–17
Math Instructions C–19
Differential Instructions C–22
Bit Instructions C–22
Number Conversion Instructions C–23
Table Instructions C–23
CPU Control Instructions C–25
Program Control Instructions C–25
Interrupt Instructions C–25
Network Instructions C–25
Intelligent I/O Instructions C–26
Message Instructions C–26

RLL plus Instructions C–26
Drum Instructions C–26
Clock / Calender Instructions C–27
MODBUS Instructions C–27
ASCII Instructions C–27

Appendix D: Special Relays

DL06 PLC Special Relays D–2
Startup and Real-Time Relays D–2
CPU Status Relays D–2
System Monitoring D–3
Accumulator Status D–3
HSIO Input Status D–4
HSIO Pulse Output Relay D–4
Communication Monitoring Relay D–4
Option Slot Communication Monitoring Relay D–4
Option Slot Special Relay D–4
Counter 1 Mode 10 Equal Relays D–5
Counter 2 Mode 10 Equal Relays D–6

Appendix E: Product Weights

Product Weight Table E–2

Appendix F: PLC Memory

DL06 PLC Memory F-2

Appendix G: ASCII TABLE

ASCII Conversion Table G–2

Appendix H: European Union Directives (CE)

European Union (EU) Directives H-2
Member Countries H-2
Applicable Directives H-2
Compliance H-2
General Safety H-3

Special Installation Manual H-4
Other Sources of Information H-4

Basic EMC Installation Guidelines H-4
Enclosures H-4
AC Mains Filters H-5
Suppression and Fusing H-5
Internal Enclosure Grounding H-5
Equi–potential Grounding H-6
Communications and Shielded Cables H-6
Analog and RS232 Cables H-7
Multidrop Cables H-7
Shielded Cables within Enclosures H-7
Network Isolation H-7
DC Powered Versions H-8
Items Specific to the DL06 . . . . . H-9

Index

Drum Instruction Programming

In This Chapter...

Introduction 6–2
Step Transitions 6–4
Overview of Drum Operation 6–8
Drum Control Techniques 6–10
Drum Instruction 6–12

Introduction

Purpose

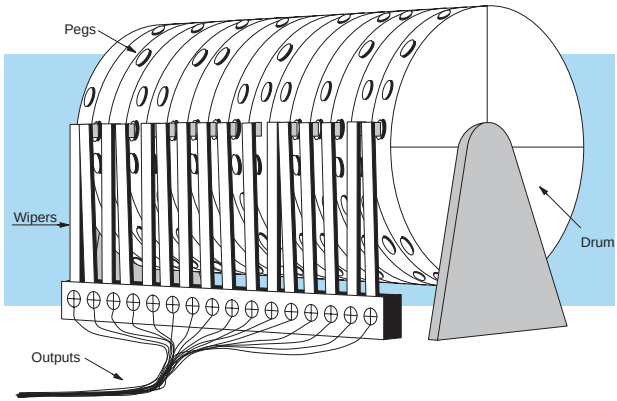
The Event Drum (EDRUM) instruction in the DL06 CPU electronically simulates an electro-mechanical drum sequencer. The instruction offers enhancements to the basic principle, which we describe first.

Drum Terminology

Drum instructions are best suited for repetitive processes that consist of a finite number of steps. They can do the work of many rungs of ladder logic with elegant simplicity. Therefore, drums can save a lot of programming and debugging time.

We introduce some terminology associated with the **drum** instruction by describing the original mechanical drum shown 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!

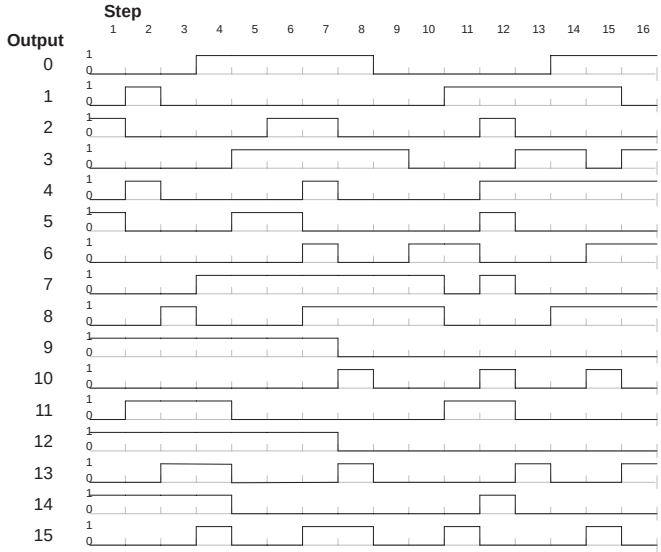
Drum Chart Representation

For editing purposes, the electronic drum is presented in chart form in *Direct*SOFT32 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).

	OUTPUTS															
STEP	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 that they are equivalent! If you can see their equivalence, you are well on your way to understanding drum instruction operation.



Step Transitions

Drum Instruction Types

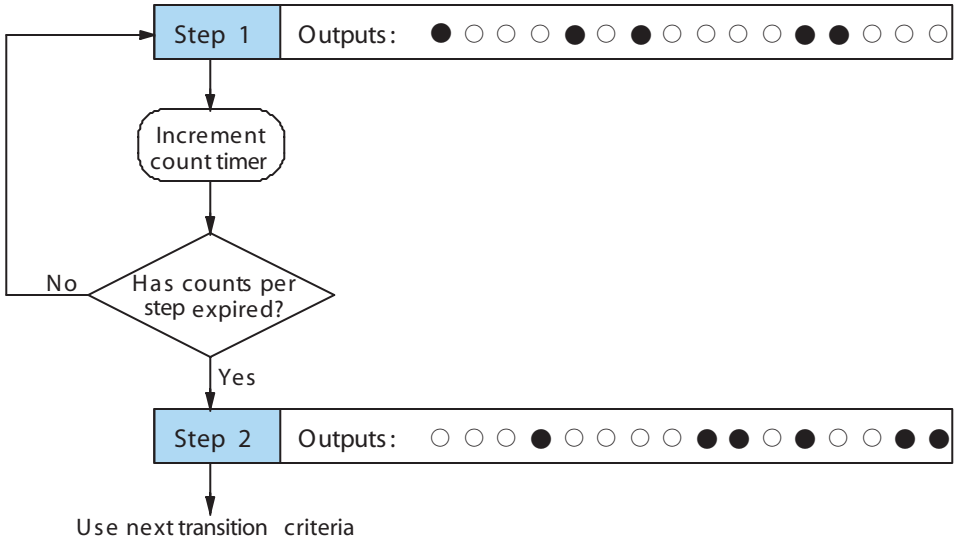
There are two types of Drum instructions in the DL06 CPU:

- Timed Drum with Discrete Outputs (DRUM)
- Time and Event Drum with Discrete Outputs (EDRUM)

The two drum instructions include time-based step transitions, and the EDRUM includes event-based transitions as well. 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.

Timer-Only Transitions

Drums move from one step to another based on time and/or an external event (input). Each step has its own transition condition which you assign during the drum instruction entry. The figure below shows how timer-only transitions work.



The drum stays in Step 1 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. When the counts for Step 1 have expired, then the drum moves to Step 2. The outputs change immediately to match the new pattern for Step 2.

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

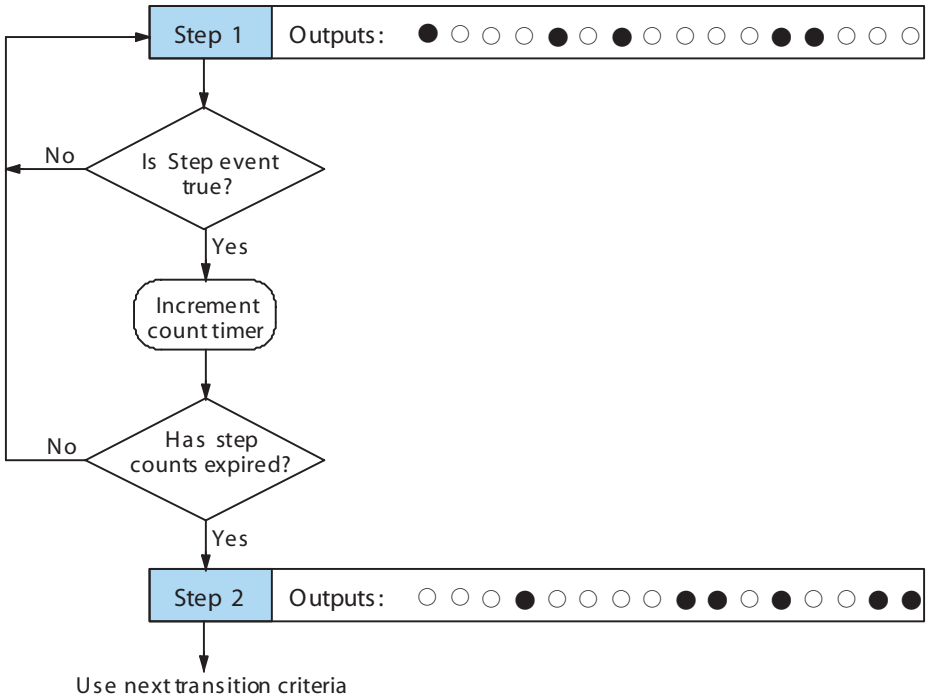
For example, if you program a 5 second time base and 12 counts for Step 1, then the drum will spend 60 seconds in Step 1. The maximum time for any step is given by the formula:

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 of thumb is to make it about 1/10 the duration of the shortest step in your drum. Then 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 that the drum instruction executes once per CPU scan. Therefore, it is pointless to specify a drum timebase that is much faster than the CPU scan time.*

Timer and Event Transitions

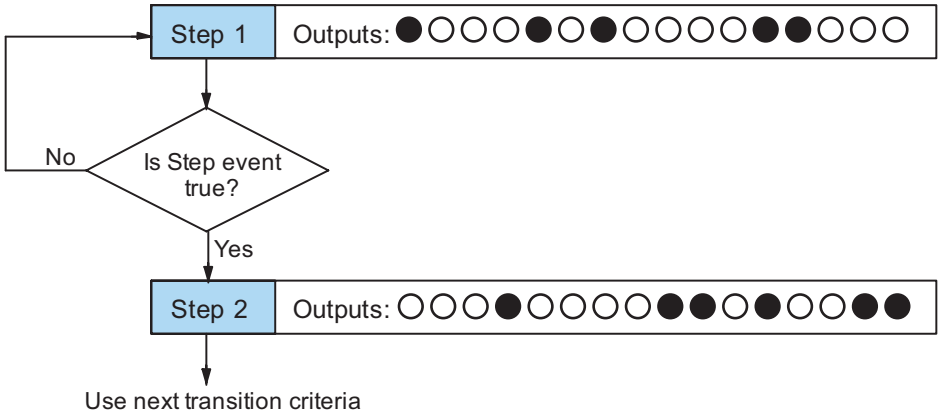
Step transitions may also occur based on time and/or external events. The figure below shows how step transitions work in these cases.



When the drum enters Step 1, it sets the output pattern as shown. Then 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 (X0) remains true. When the counts for Step 1 have expired, then the drum moves to Step 2.The outputs change immediately to match the new pattern for Step 2.

Event-Only Transitions

Step transitions do not require both the event and the timer criteria programmed for each step. You have the option of programming just 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 the counter bit precisely indicate the progress of the drum instruction, and can be monitored by your ladder program.

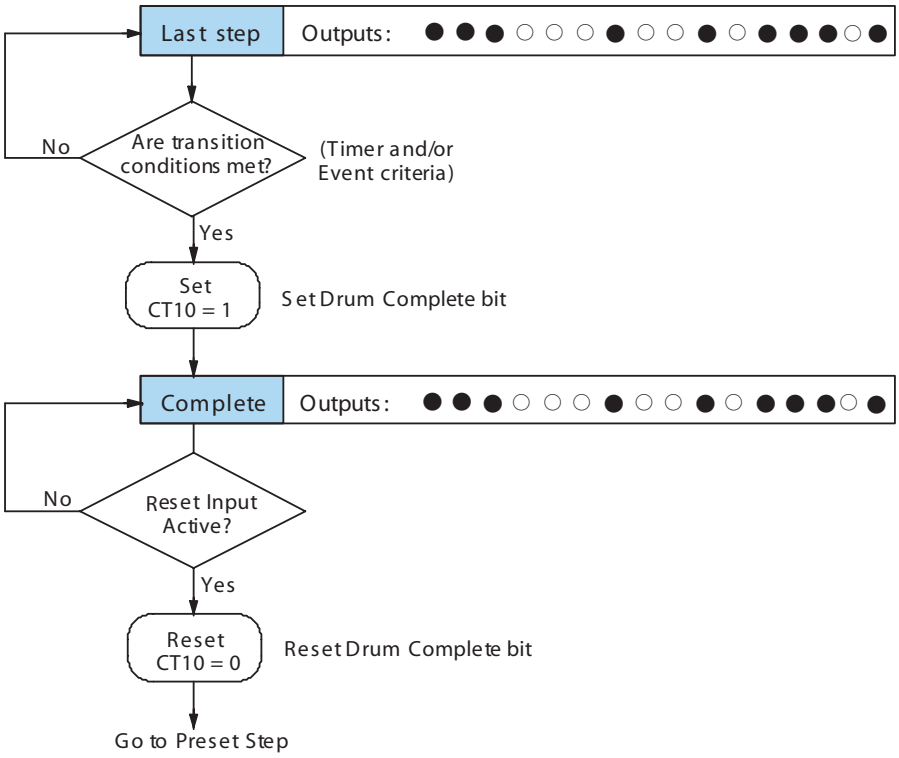
Suppose we program a timer drum to have 8 steps, and we select CT10 for the counter number (remember, counter numbering is in octal). Counter usage is shown to the right. The right column holds typical values, interpreted below.

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 that we 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, then the step is just 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 that we have been in the current count (1528) for 2 seconds (0.01 x 200). 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 changes only if the ladder program writes to it, or the drum instruction is edited and the program is restarted. Counter bit CT10 turns on when the drum cycle is complete, and turns off when the drum is reset.

Last Step Completion

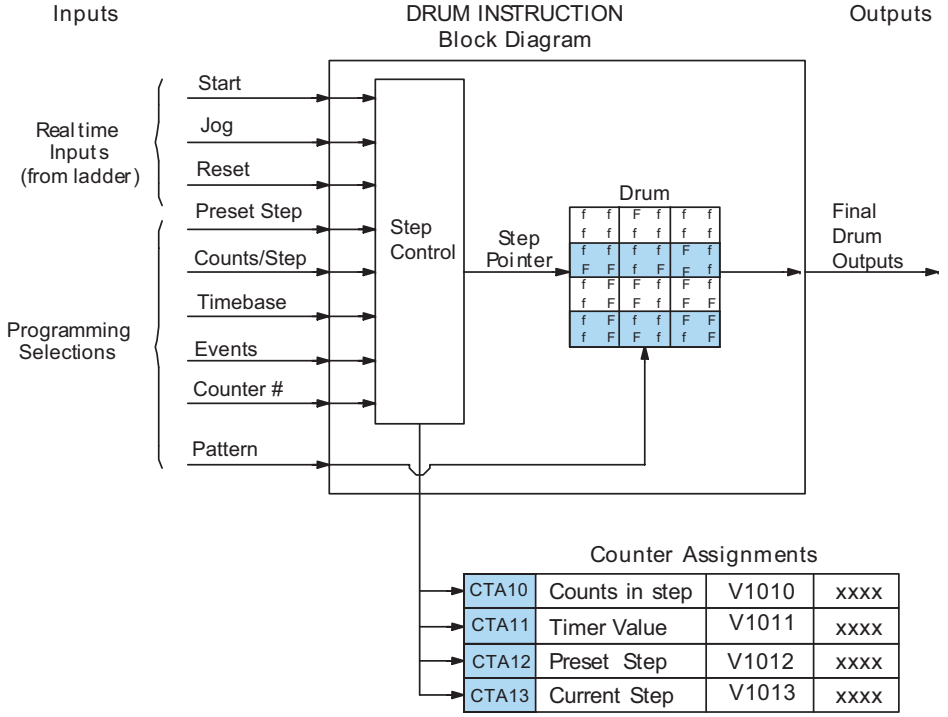
The last step in a drum sequence may be any step number, since partial drums are valid. Refer to the following figure. When the transition conditions of the last step are met, the drum sets the counter bit corresponding to the counter named in the drum instruction box (such as CT10). Then it moves to a final "drum complete" state. The drum outputs remain in the pattern defined for the last step. 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 CT10), 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 (only EDRUM supports the 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.
- **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. The DL06 has 128 counters (CT0 – CT177 in octal).
- **Events** – Either an X, Y, C, S, T, or CT type discrete point serves as step transition inputs. Each step has its own event. However, programming the event is optional.

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 current step of the drum. This initial step number depends on the counter memory configuration: non-retentive versus retentive.

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.

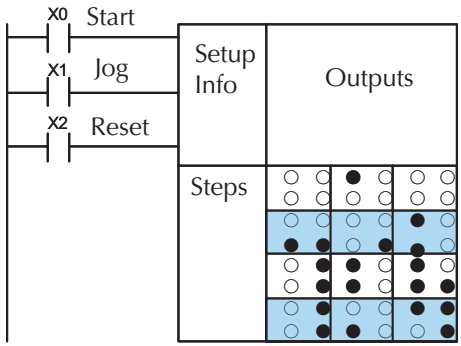
Counter Number	Function	Initialization on Powerup	
		Non-Retentive Case	Retentive Case
CTA(n)	Current Step Count	Initialize = 0	Use Previous (no change)
CTA(n + 1)	Counter Timer Value	Initialize = 0	Use Previous (no change)
CTA(n + 2)	Preset Step	Initialize = Preset Step #	Use Previous (no change)
CTA(n + 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.

Drum Control Techniques

Drum Control Inputs

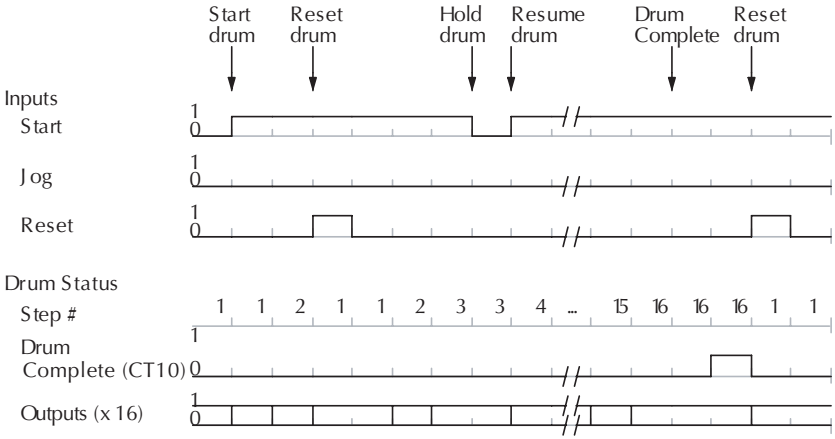
Now we are ready to put together the concepts on the previous pages and demonstrate general control of the drum instruction box. The drawing to the right shows a simplified generic drum instruction. Inputs from ladder logic control the Start, Jog, and Reset Inputs (only the EDRUM instruction supports the Jog Input). The first counter bit of the drum (CT10, 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 it is Step 1). When the Start input turns on the drum begins running, waiting for an event and/or running the timer (depends on the 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 that 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.

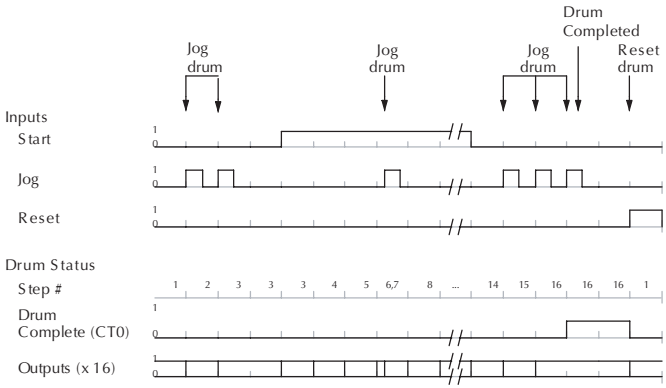


When the drum completes the last step (Step 16 in this example), the Drum Complete bit (CT10) turns on, and the step number remains at 16. When the Reset input turns on, it turns off the Drum Complete bit (CT10), and forces the drum to enter the preset step.

NOTE: The timing diagram shows all steps using equal time durations. Step times can vary greatly, depending on the counts/step programmed.

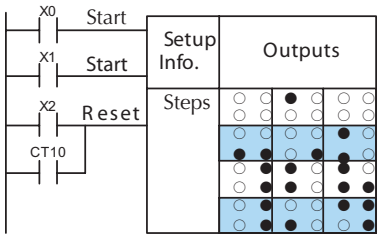
In the figure below, we focus on how the Jog input works on event drums. To the left of the diagram, note that the off-to-on transitions of the Jog input increments the step. Start may be either on or off (however, Reset must be off). Two jogs takes the drum to step three. Next, the Start input turns 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 CT10, so we logically OR the drum complete bit (CT10) with the Reset input. When the last step is done, the drum turns on CT10 which resets itself to the preset step, also resetting CT10. Contact X2 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, make the preset step in the drum a "reset step", with all outputs off.

Using Complex Event Step Transitions

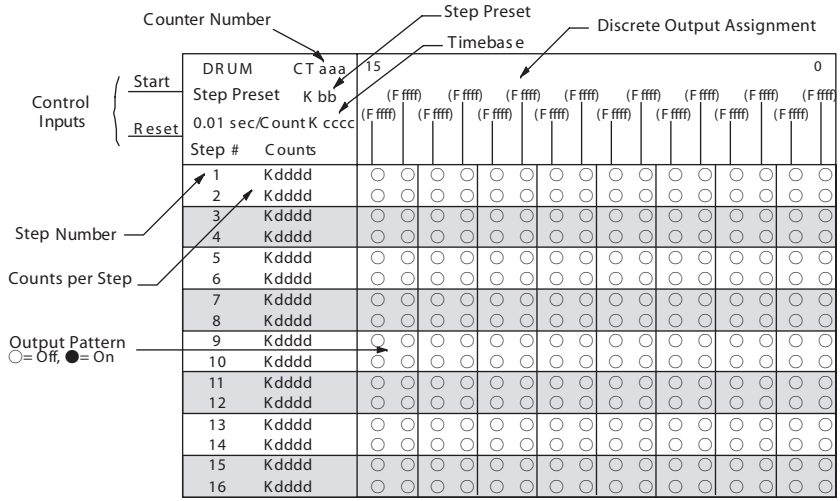
Each event-based transition accepts only one contact reference for the event. However, this does not limit events to just one contact. Just use a control relay contact such as C0 for the step transition event. Elsewhere in ladder logic, you may use C0 as an output coil, making it dependent on many other "events" (contacts).

Drum Instruction

The DL06 drum instructions may be programmed using *Direct*SOFT32 or for the EDRUM instruction only you can use a handheld programmer (firmware version v2.21 or later). This section covers entry using *Direct*SOFT32 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 DL06'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*SOFT32.



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*SOFT32.

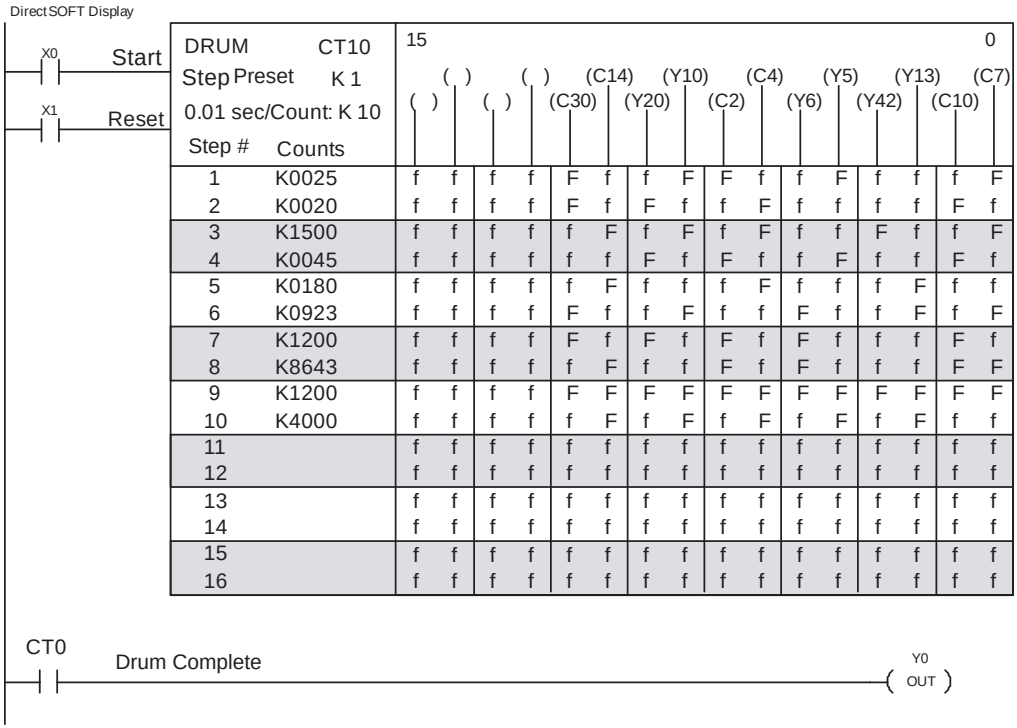
Whenever the Start input is energized, the drum's timer is enabled. It stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa --	0 --174	
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 memory map

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 CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

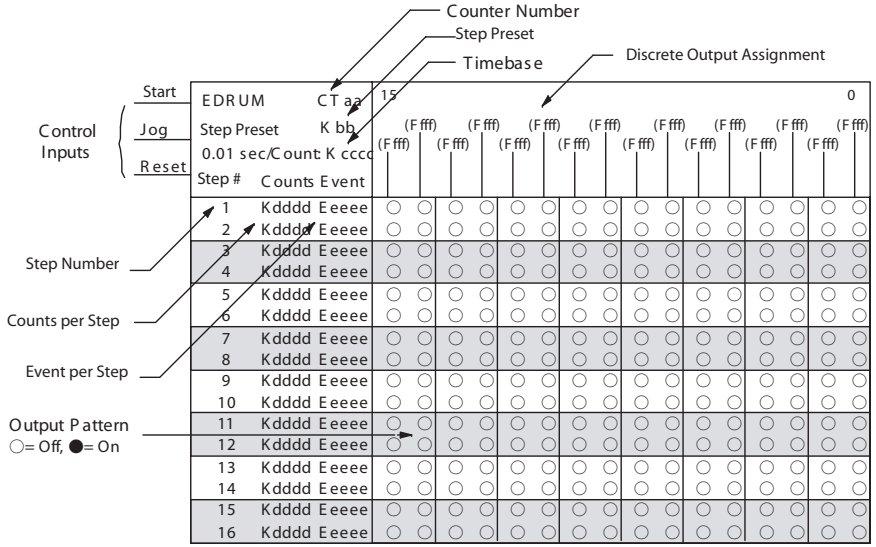
Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 -- 174	Counts in step	CT(n) = Drum Complete
CTA(n+1)	1 -- 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 -- 176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 -- 177	Current Step	CT(n+3) = (not used)

The following ladder program shows the DRUM instruction in a typical ladder program, as shown by ***Direct***SOFT32. 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 (K10 x 0.01) = 0.1 second per count. Therefore, the duration of step 1 is (25 x 0.1) = 2.5 seconds. 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.



Event Drum (EDRUM)

The Event Drum (EDRUM) features time-based and event-based step transitions. It operates according to the general principles of drum operation covered in the beginning of this chapter. Below is the instruction as displayed by *Direct*SOFT32.



The Event Drum features 16 steps and 16 discrete 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 and events must be left blank. The discrete output points may be individually assigned.

Drum Parameters	Field	Data Types	Ranges
Counter Number	aa	--	0 -- 174
Preset Step	bb	K	1 -- 16
Timer base	cccc	K	0 -- 99.99 seconds
Counts per step	dddd	K	0 -- 9999
Event	Eeeee	X, Y, C, S, T, CT, SP	see memory map
Discrete Outputs	ffff	X, Y, C	see memory map

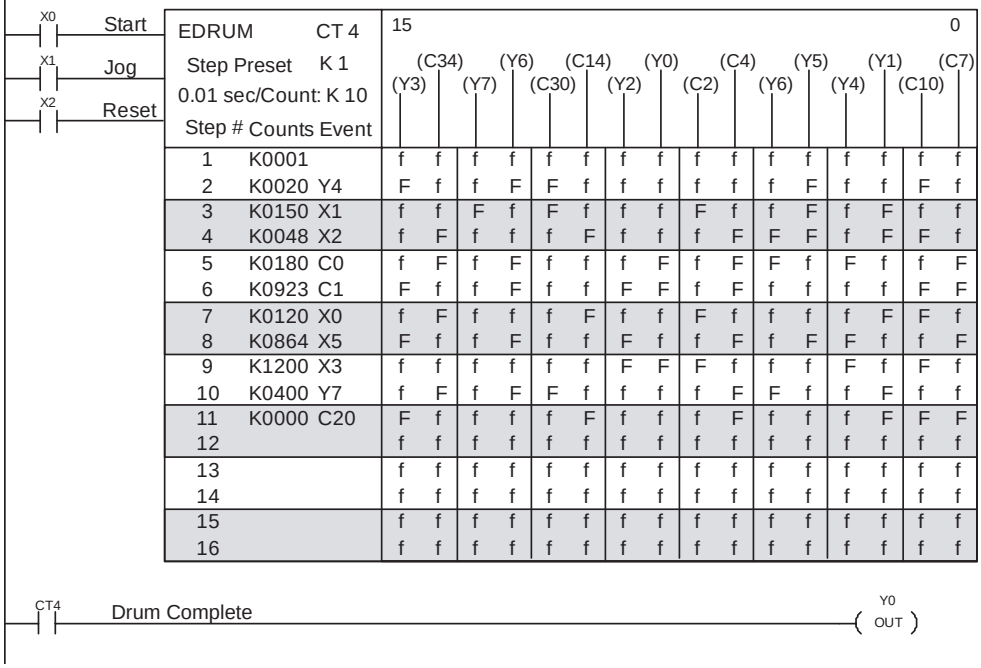
Whenever the Start input is energized, the drum's timer is enabled. As long as the event is true for the current step, the timer runs during that step. When the step count equals the counts per step, the drum transitions to the next step. This process stops when the last step is complete, or when the 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 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 CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 -- 174	Counts in step	CT(n)= Drum Complete
CTA(n+1)	1 -- 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 -- 176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 -- 177	Current Step	CT(n+3) = (not used)

The following ladder program shows the EDRUM instruction in a typical ladder program, as shown by ***Direct***SOFT32. Steps 1 through 11 are used, and all sixteen output points are used. The preset step is step 1. The timebase runs at (K10 x 0.01) = 0.1 second per count. Therefore, the duration of step 1 is (1 x 0.1) = 0.1 second. Note that step 1 is time-based only (event is left blank). 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 11). A drum reset also resets CT4.

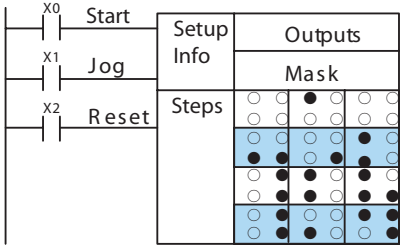
DirectSOFT Display



Handheld Programmer Drum Mnemonics

The EDRUM instruction may be programmed using either *Direct*SOFT32 or a handheld programmer. This section covers entry via the handheld programmer (Refer to the *Direct*SOFT32 manual for drum instruction entry using that tool).

First, enter Store instructions for the ladder rungs controlling the drum's ladder inputs. In the example to the right, the timer drum's Start, Jog, and Reset inputs are controlled by X0, X1 and X2 respectively. The required keystrokes are listed beside the mnemonic.



These keystrokes precede the EDRUM instruction mnemonic. Note that the ladder rungs for Start, Jog, and Reset inputs are not limited to being single–contact rungs.

Handheld Programmer Keystrokes

Store X0 | $ STR | → | A 0 | ENT

(Repeat for Store X1 and Store X2)

Handheld Programmer Keystrokes

EDRUM CNT4 | SHFT | E 4 | D 3 | R ORN | U ISG | M ORST | → | E 4 | ENT

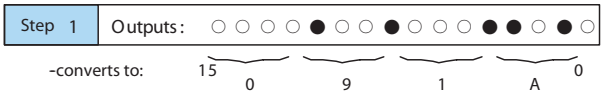
After the Store instructions, enter the EDRUM (using Counter CT0) as shown:

After entering the EDRUM mnemonic as above, the handheld programmer creates an input form for all the drum parameters. The input form consists of approximately fifty or more default mnemonic entries containing DEF (define) statements. The default mnemonics are already "input" for you, so they appear automatically. Use the NXT and PREV keys to move forward and backward through the form. Only the editing of default values is required, thus eliminating many keystrokes. The entries required for the basic timer drum are in the chart below.

NOTE: Default entries for output points and events are "DEF 0000", which means they are unassigned. If you need to go back and change an assigned output as unused again, enter "K0000". The entry will again show as "DEF 0000".

Drum Parameters	Multiple Entries	Mnemonic / Entry	Default Mnemonic	Valid Data Types	Ranges
Start Input	--	STR (plus input rung)	--	--	--
Jog Input	--	STR (plus input rung)	--	--	--
Reset Input	--	STR (plus input rung)	--	--	--
Drum Mnemonic	--	DRUM CNT aa	--	CT	0 -- 174
Preset Step	1	bb	DEF K0000	K	1 -- 16
Timer base	1	cccc	DEF K0000	K	1 -- 9999
Output points	16	ffff	DEF 0000	X, Y, C	see memory map
Counts per step	16	dddd	DEF K0000	K	0 -- 9999
Events	16	dddd	DEF K0000	X, Y, C, S, T, CT, SP	see memory map
Output pattern	16	gggg	DEF K0000	K	0 -- FFFF

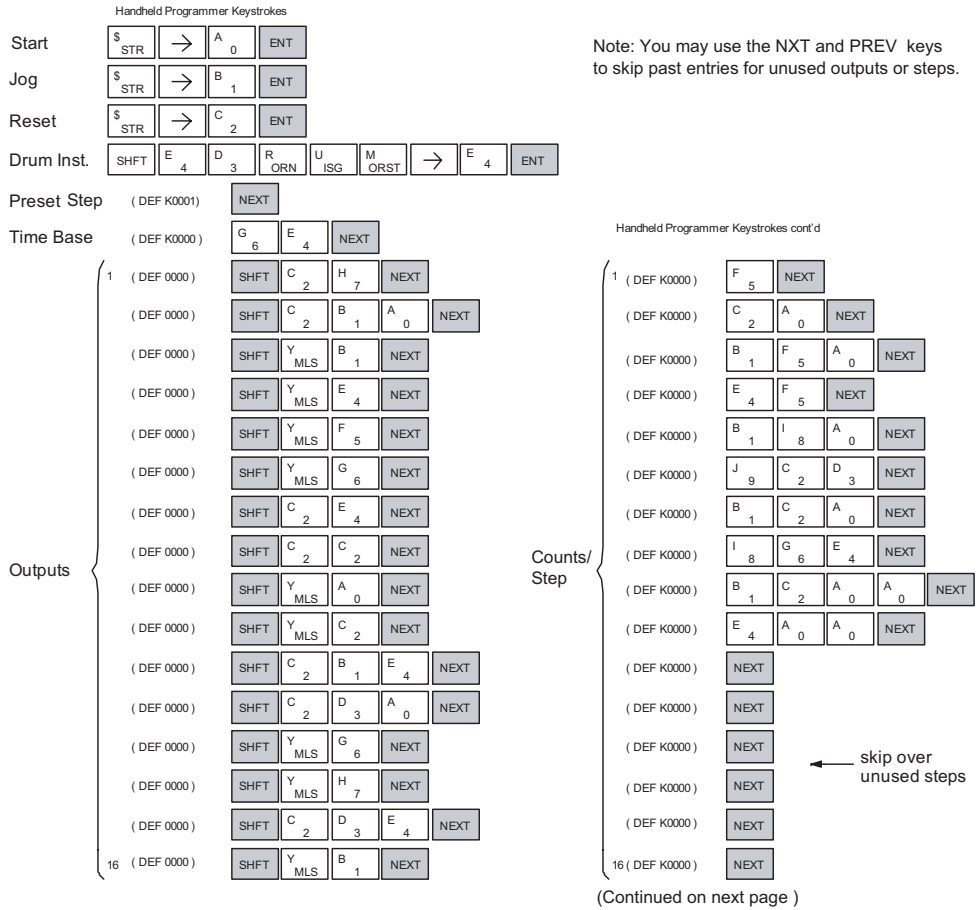
Using the DRUM entry chart (two pages before), we show the method of entry for the basic time/event drum instruction. First, we convert the output pattern for each step to the equivalent hex number, as shown in the following example.

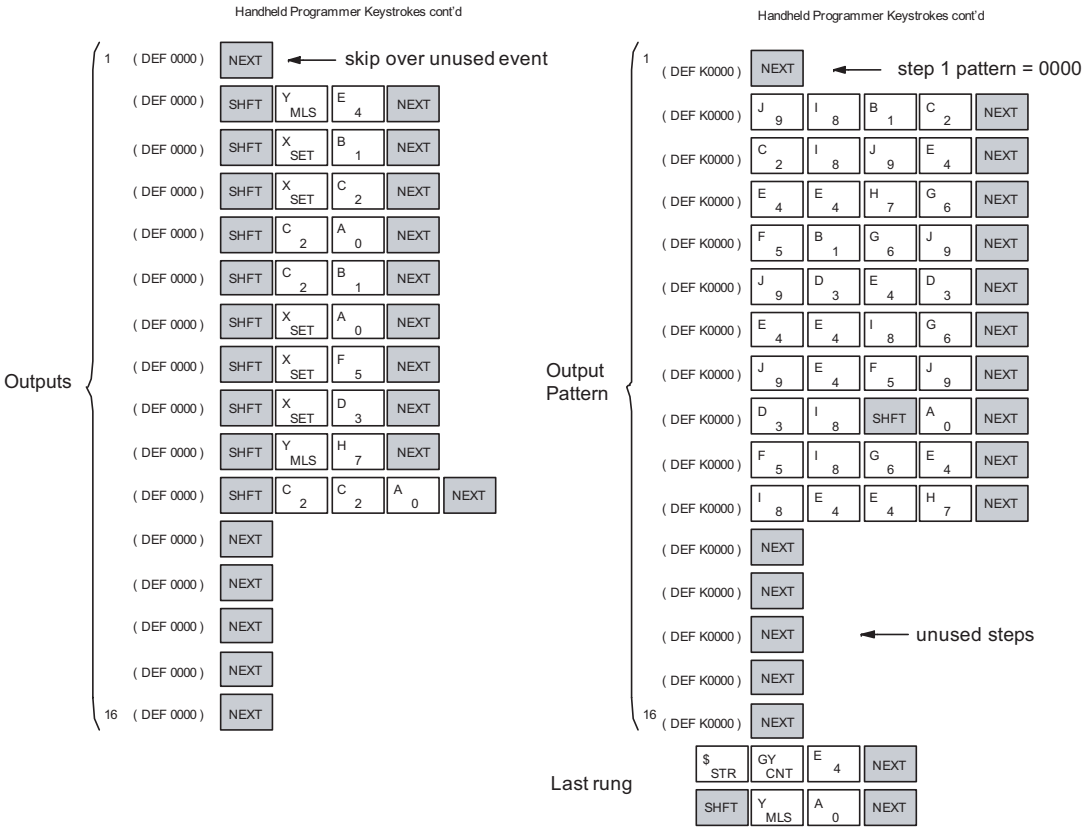


The following diagram shows the method for entering the previous EDRUM example on the HHP. The default entries of the form are in parenthesis. After the drum instruction entry (on the fourth row), the remaining keystrokes over-write the numeric portion of each default DEF statement. **NOTE**: Drum editing requires Handheld Programmer firmware version 2.21 or later.

***Note**: You may use the NXT and PREV keys to skip past entries for unused outputs or steps.*

6





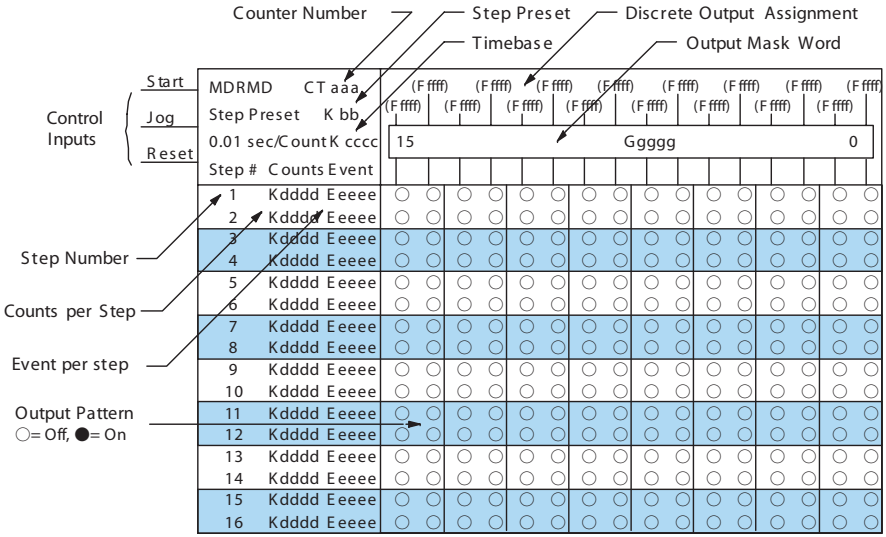
NOTE: You may use the NXT and PREV keys to skip past entries for unused outputs or stops.

***Note**: you may use the NXT and PREV keys to skip past entries for unused outputs or steps.*

***Note:** For ease of operation when using the EDRUM instruction, we recommend using DirectSOFT32 over the handheld programmer.*

Masked Event Drum with Discrete Outputs (MDRMD)

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 DirectSOFT32.



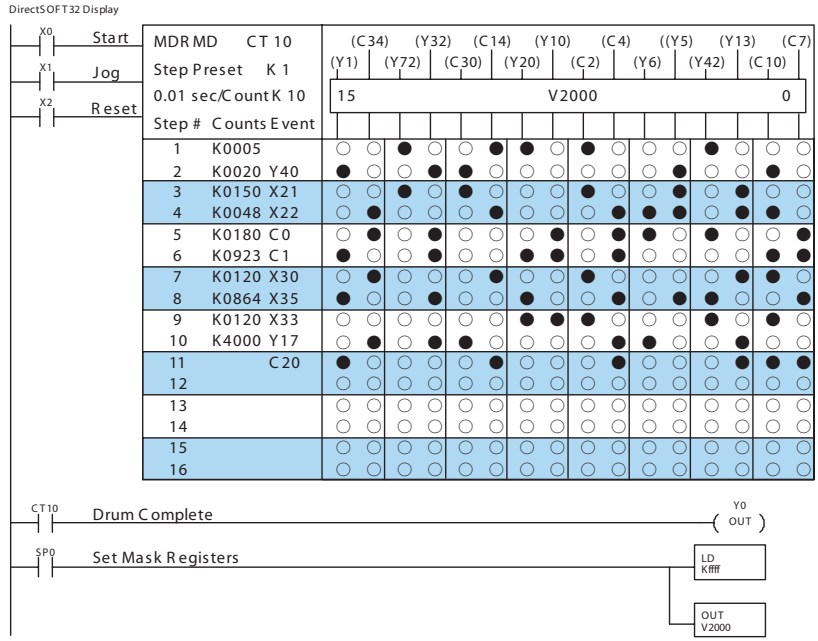
The Masked Event Drum with Discrete Outputs features sixteen steps and sixteen outputs. Drum outputs are logically ANDed bit-by-bit with an output mask word for each step. The Ggggg field specifies the beginning location of the 16 mask words. Step transitions occur on timed and/or event basis. The jog 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 and events can be left blank (this is the default entry). 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 – 174
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, GX, GY. CT, SP	see memory map
Discrete Outputs	Fffff	X, Y, C, GX, GY	
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 CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 – 174	Counts in step	CT(n) = Drum Complete
CTA(n+1)	1 – 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 –176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 –177	Current Step	CT(n+1) = (not used)

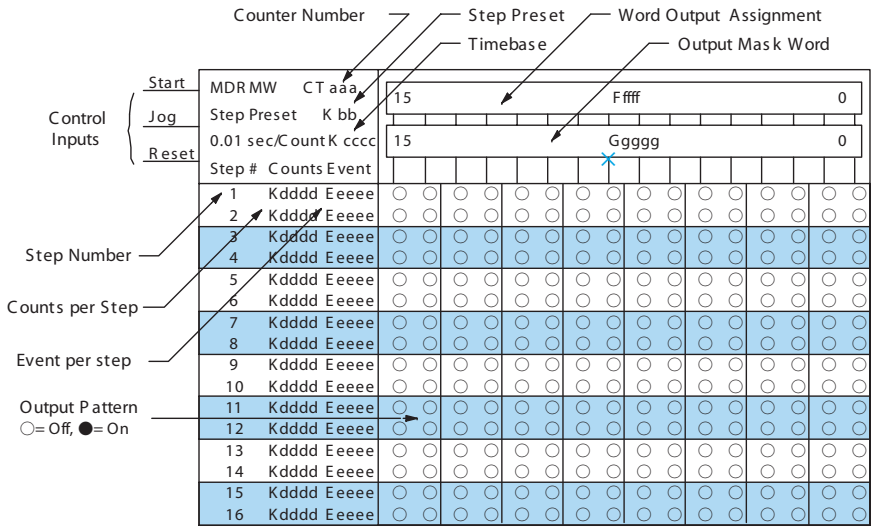
The following ladder program shows the MDRMD instruction in a typical ladder program, as shown by DirectSOFT32. Steps 1 through 11 are used, and all 16 output points are used. The output mask word is at V2000. The final drum outputs are shown above the mask word as individual bits. The data bits in V2000 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 (K10 x 0.01)=0.1 second 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 is left blank). 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.



NOTE: The ladder program must load constants in V2000 through V2012 to cover all mask registers for the eleven steps used in this drum

Masked Event Drum with Word Output (MDRMW)

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 DirectSOFT32.



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 and events can be left blank (this is the default entry).

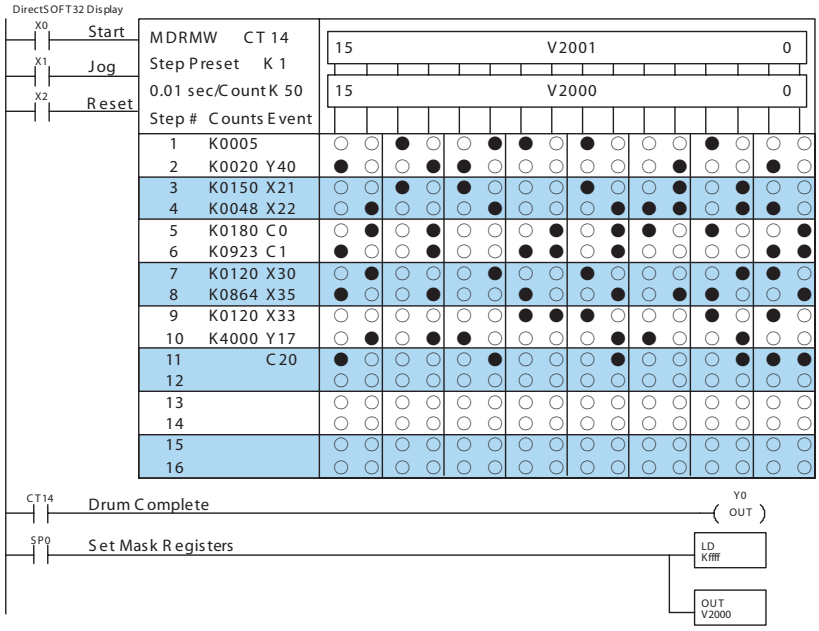
Whenever the Start input is energized, the drum's timer is enabled. As long as the event is true for the current step, the timer runs during that step. When the step count equals the counts per step, the drum transitions to the next step. This process stops when the last step is complete, or when the 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 – 174
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, GX, GY, SP	see memory map
Word Output	Fffff	V	see memory map
Output Mask	Ggggg	V	see memory map

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 CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 – 174	Counts in step	CT(n) = Drum Complete
CTA(n+1)	1 – 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 –176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 –177	Current Step	CT(n+1) = (not used)

The following ladder program shows the MDRMD instruction in a typical ladder program, as shown by DirectSOFT32. 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 (K50 x 0.01)=0.5 seconds per count. Therefore, the duration of step 1 is (5 x 0.5) = 2.5 seconds. Note that step 1 is time-based only (event is left blank). 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.



NOTE: The ladder program must load constants in V2000 through V2012 to cover all mask registers for the eleven steps used in this drum

RLL PLUS STAGE PROGRAMMING

In This Chapter...

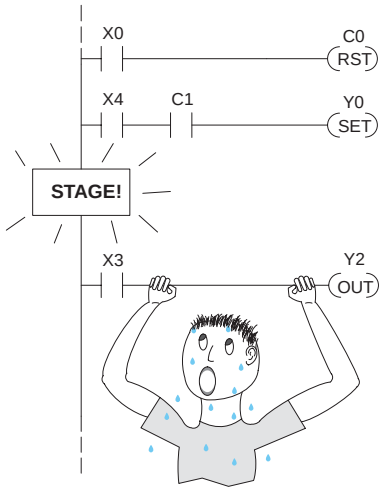
Introduction to Stage Programming 7–2
Learning to Draw State Transition Diagrams 7–3
Using the Stage Jump Instruction for State Transitions 7–7
Stage Program Example: Toggle On/Off Lamp Controller 7–8
Four Steps to Writing a Stage Program 7–9
Stage Program Example: A Garage Door Opener 7–10
Stage Program Design Considerations 7–15
Parallel Processing Concepts 7–19
RLLPLUS (Stage) Instructions 7–21
Questions and Answers about Stage Programming 7–27

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 won't 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.
- 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:

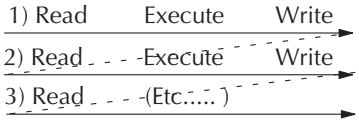
- Start at the beginning and do not skip over any sections.
- Study each stage programming concept by working through each example. The examples build progressively on each other.
- Read the Stage Questions and Answers at the end of the chapter for a quick review.

Learning to Draw State Transition Diagrams

Introduction to Process States

Those familiar with ladder program execution know that 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 just a few milliseconds.

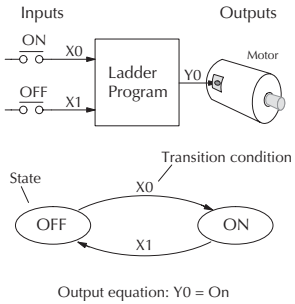
Most manufacturing processes consist of a series of activities or conditions , each lasting for several seconds. minutes, or even hours. We might call these "process states", which are either active or inactive at any particular time. A challenge for RLL programs is that a particular input event may last for just 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 just a tool to help us draw a picture of our process!* You'll discover that if we can get the picture right, **our program will also be right!**

A 2–State Process



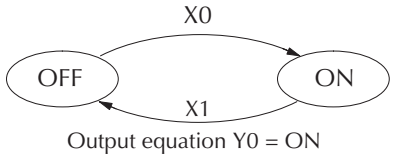
Consider the simple process shown to the right, which controls an industrial motor. We will use a green momentary SPST pushbutton to turn the motor on, and a red one to turn it off. The machine operator will press the appropriate pushbutton for just a second or so. The two states of our process are ON 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.

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.

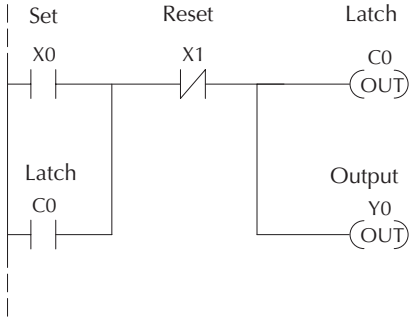
The state transition diagram to the right is a picture of the solution we need to create. The beauty of it is this: it expresses the problem independently of the programming language we may use to realize it. In other words, *by drawing the diagram we have already solved the control problem!*



First, we'll translate the state diagram to traditional RLL. Then we'll show how easy it is to translate the diagram into a stage programming solution.

RLL Equivalent

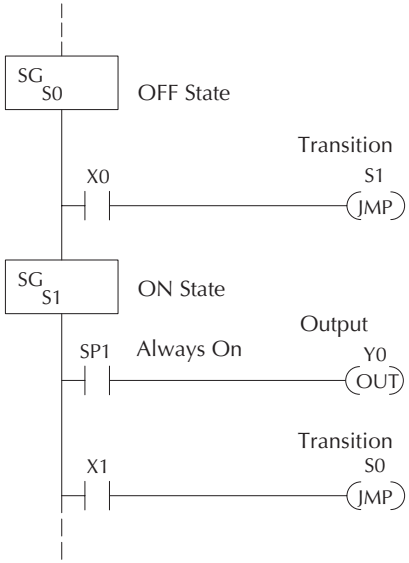
The RLL solution is shown to the right. It consists of a self-latching control relay, C0. When the On pushbutton (X0) is pressed, output coil C0 turns on and the C0 contact on the second row latches itself on. So, X0 **sets the latch** C0 on, and it remains on after the X0 contact opens. The motor output 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 that 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!

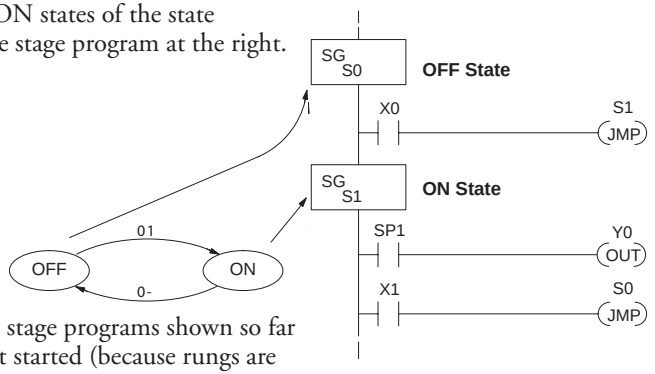
In the On State (Stage S1), we want the motor to always be on. The special relay contact SP1 is defined as always on, so Y0 turns the motor on.

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

Right now, you may be thinking "I don't see the big advantage to Stage Programming... in fact, the stage program is longer than the plain RLL program". Well, now is the time to exercise a bit of faith. 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!

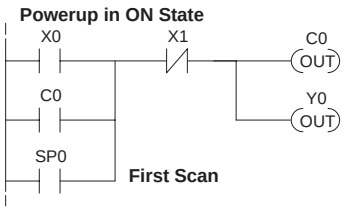


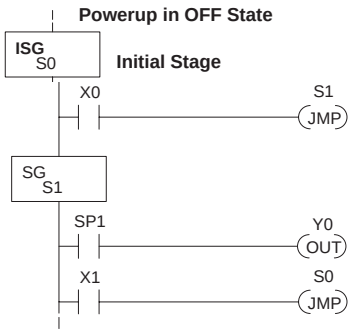
Initial Stages

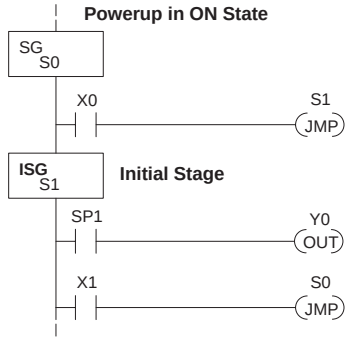
At powerup and Program-to-Run Mode transitions, the PLC always begins with all normal stages (SG) off. So, the stage programs shown so far have actually had no way to get started (because rungs are not scanned unless their stage is active).

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 just like any other stage!**

We can change both programs so that 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.

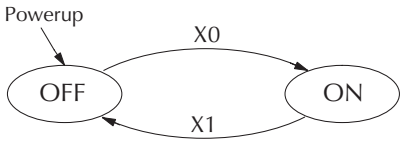






7

We can mark our 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.

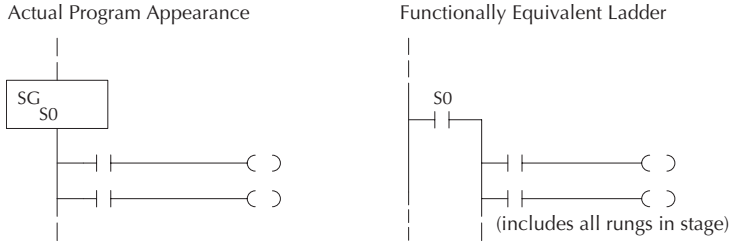


What Stage Bits Do

You may recall that a stage is just a section of ladder program which is either active or inactive at a given moment. All stage bits (S0 to 1777) reside in the PLC's 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.

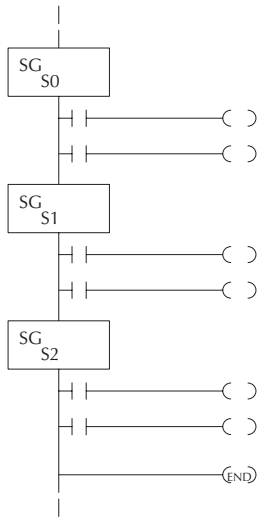
- If Stage bit S0 = 0, its ladder rungs *are not scanned* (executed).
- If Stage bit S0 = 1, its ladder rungs *are scanned* (executed).



Stage Instruction Characteristics

The inline stage boxes on the left power rail divide the ladder program rungs into stages. Some stage rules are:

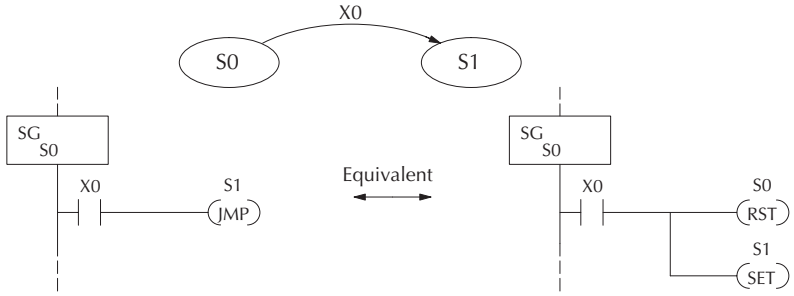
- **Execution** – Only logic in active stages are executed on any scan.
- **Transitions** – Stage transition instructions take effect on the next occurrence of the stages involved.
- **Octal numbering** – Stages are numbered in octal, like I/O points, etc. So "S8" is not valid.
- **Total Stages** – The DL06 offers up to 1024 stages (S0 to 1777 in octal).
- **No duplicates** –Each stage number is unique and can be used just 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

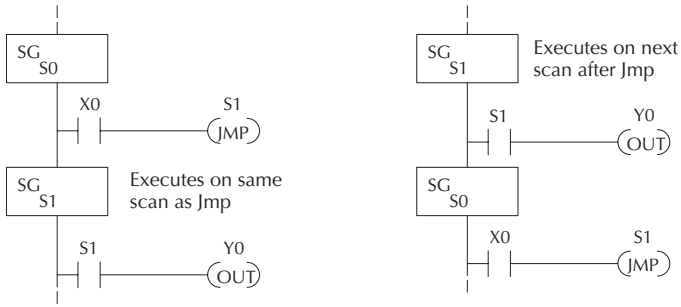
Stage Jump, Set, and Reset Instructions

The Stage JMP instruction we have used 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. So, the Stage Jump instruction is equal to a Stage Reset of the current stage, plus a Stage Set instruction for the stage to which we 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:

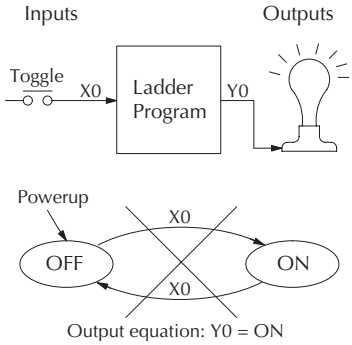
- The jump instruction resets the stage bit of the stage in which it occurs. All rungs in the stage still finish executing during the current scan, *even if there are other rungs in the stage below the jump instruction!*
- The reset will be in effect on the following scan, so the stage that executed 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

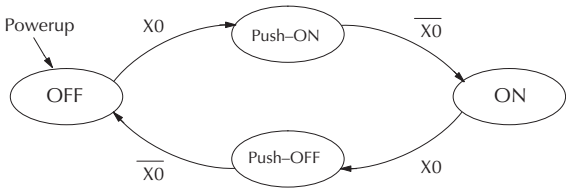


In the process shown to the right, we use an ordinary momentary pushbutton to control a light bulb. The ladder program will latch the switch input, so that we will push and release to turn on the light, push and release again to turn it off (sometimes called toggle function). Sure, we could just buy a mechanical switch with the alternate on/off action built in... However, this example is educational and also fun! Next we 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).

Note that this example differs from the motor example, because now we have just one pushbutton. When we press the pushbutton, both transition conditions are met. We would just transition around the state diagram at top speed. If implemented in Stage, this solution would flash the light on or off each scan (obviously undesirable)!

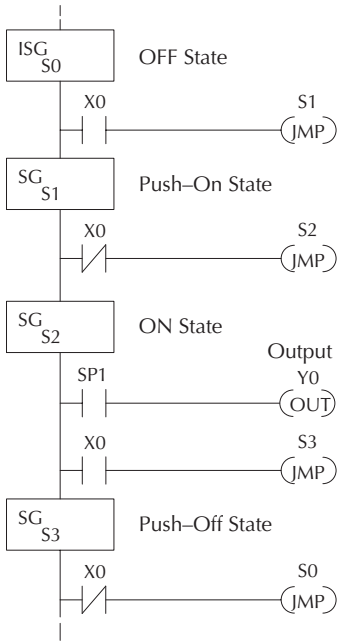
The solution is to make the push and the release of the pushbutton separate events. Refer to the new state transition diagram below. At powerup we enter the OFF state. When switch X0 is pressed, we enter the Press-ON state. When it is released, we enter the ON state. Note that X0 with the bar above it denotes X0 NOT.



When in the ON state, another push and release cycle similarly takes us back to the OFF state. Now we have 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, so we make S0 an initial stage (ISG). In the ON state, we add special relay contact SP1, which is always on.

Note that even as our 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 we follow the same steps 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 us 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 DL06, numbered 0 to 1777 in octal.
- 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'll notice that Steps 1 through 3 just prepare us to write the stage program in Step 4. However, the program virtually writes itself because of the preparation beforehand. Soon you'll 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

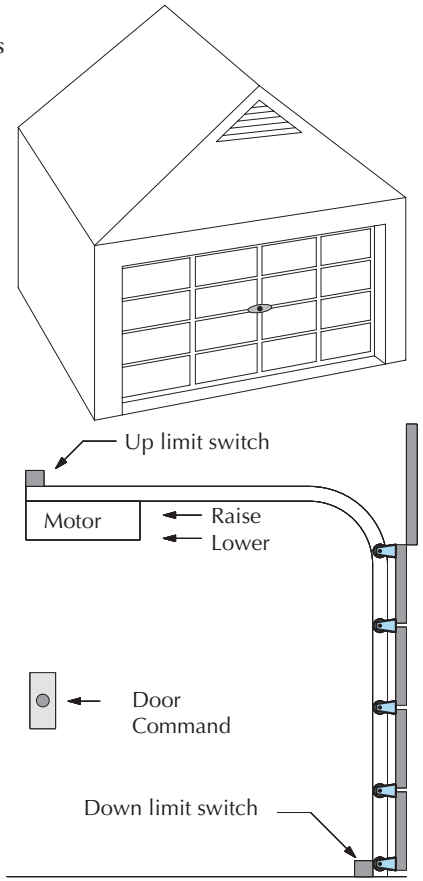
In this next stage programming example we'll create a garage door opener controller. Hopefully most readers are familiar with this application, and we can have fun besides!

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 reach the end of travel in the corresponding direction. In the middle of travel, neither limit switch is closed.

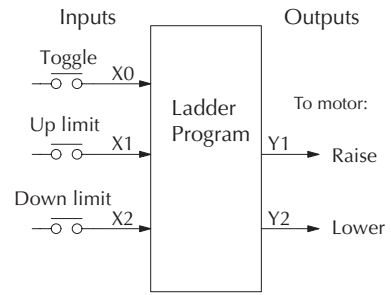
The motor has two command inputs: raise and lower. When neither input is active, the motor is stopped. The door command is just a simple pushbutton. Whether wall-mounted as shown, or a radio-remote control, all door control commands logical OR together as one pair of switch contacts.



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.

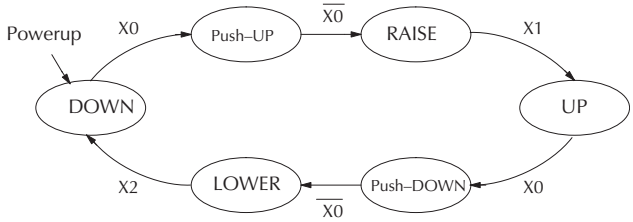
The controller has two outputs to drive the motor. Y1 is the up (raise the door) command, and Y2 is the down (lower the door) command.



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 just 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.

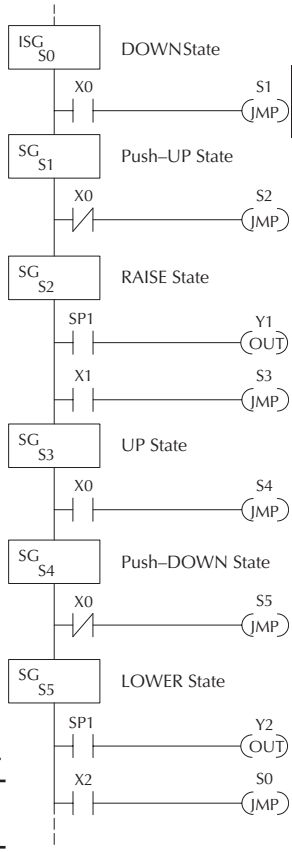


Output equations: Y1 = Raise Y2 = Lower

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.

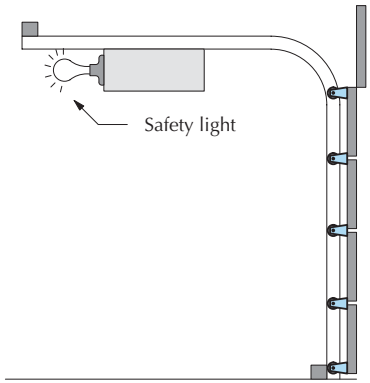
In the UP Stage S3, a push-release cycle of the pushbutton will take us to the LOWER Stage S5, where we activate Y2 to command the motor to lower the door. This continues until the door reaches the down limit switch, X2. When X2 closes, we transition from Stage S5 to the DOWN stage S0, where we began.



NOTE: *The only special thing about an initial stage (ISG) is that it is automatically active at powerup. Afterwards, it is just 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.

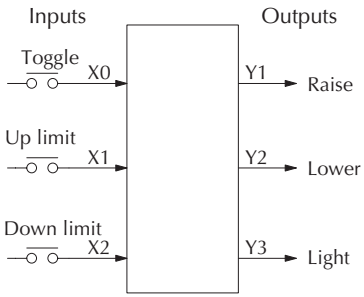


The safety light is standard on many commercially-available garage door openers. It is shown to the right, mounted on the motor housing. The light turns on upon any door activity, remaining on for approximately 3 minutes afterwards.

This part of the exercise will demonstrate the use of parallel states in our state diagram. Instead of using the JMP instruction, we'll use the set and reset commands.

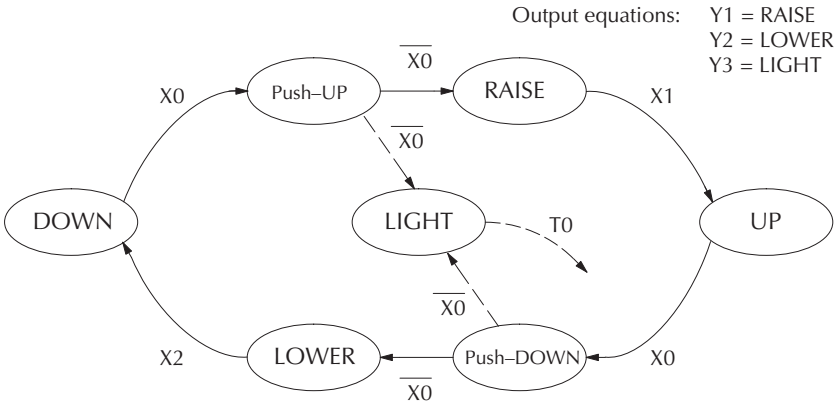
Modify the Block Diagram and State Diagram

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.

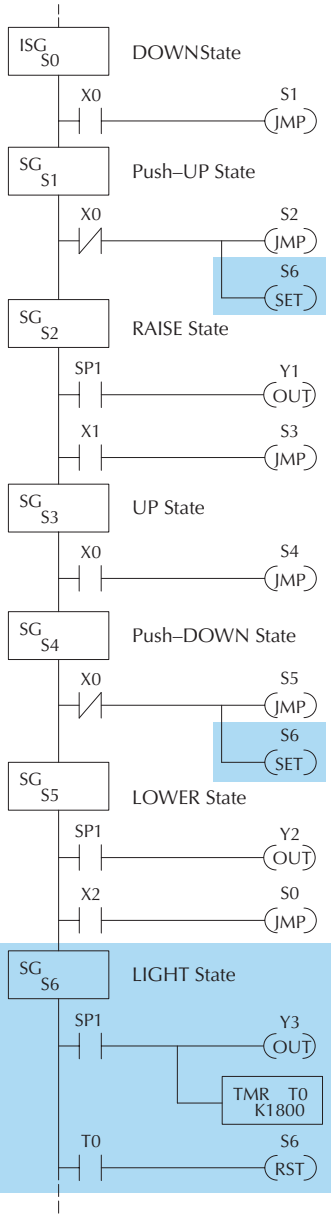
We can think of the Light state as a parallel process to the raise and lower state. The paths to the Light state are not a transition (Stage JMP), but a State Set command. In the logic of the Light stage, we will place a three-minute timer. When it expires, timer bit T0 turns on and resets the Light stage. The path out of the Light stage goes nowhere, indicating the Light stage just 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 closes, 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!

- Just 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:

$$K = \frac{3 \text{ min.} \times 60 \text{ sec/min}}{0.1 \text{ sec/count}}$$

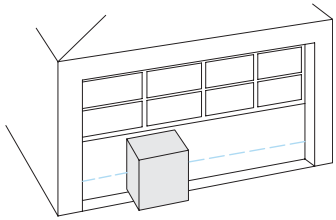
$$K = 1800 \text{ counts}$$

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.

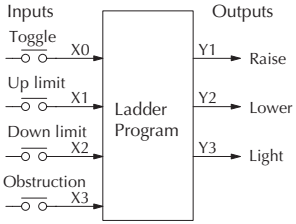
While Stage S6 is active and the light is on, stage transitions in the primary path continue normally and independently of Stage 6. That is, the door can go up, down, or whatever, but the light will be on for precisely 3 minutes.

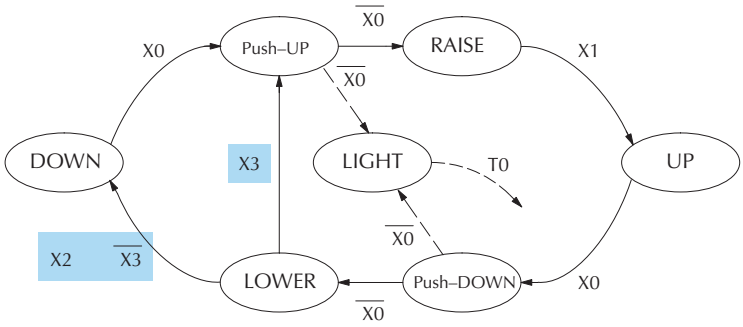
Add Emergency Stop Feature

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.

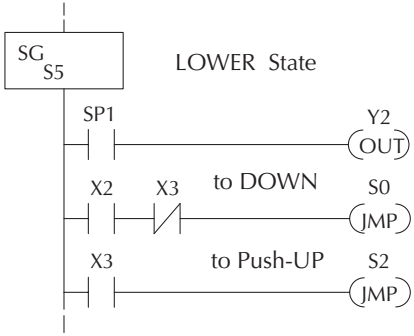




Exclusive Transitions

It is theoretically possible that 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.

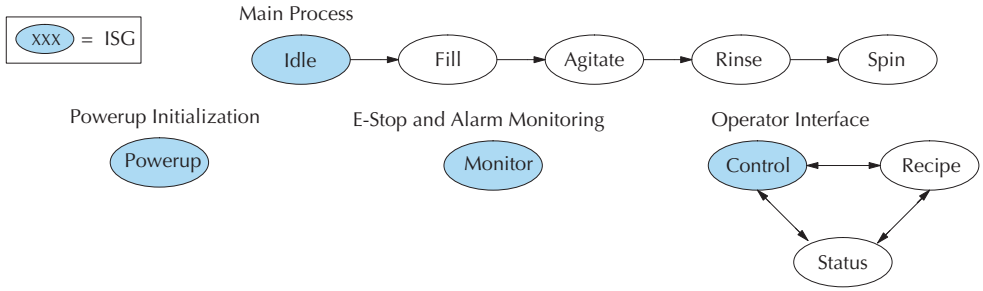


Stage Program Design Considerations

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, just put them in a stage that is always on.

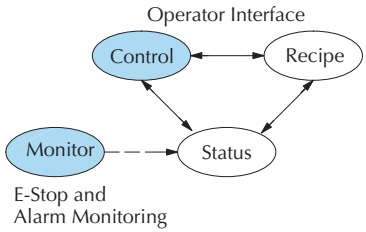
The following figure shows a typical application. During operation, the primary manufacturing activity Main Process, Powerup Initialization, E-Stop and Alarm Monitoring, and Operator Interface are all running. At powerup, three initial stages shown begin operation.



In a typical application, the separate stage sequences above operate as follows:

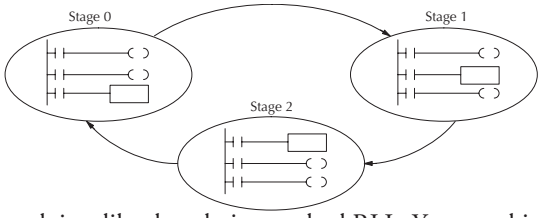
- **Powerup Initialization** – This stage contains ladder rung tasks done just once at powerup. Its last rung resets the stage, so this stage is only active for one scan (or only as many scans that are required).
- **Main Process** –This stage sequence controls the heart of the process or machine. One 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 all instructions work just like they do in standard RLL. You can think of a stage just 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.
- 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. Therefore, 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 that 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 – In using a counter inside a stage, the stage must be active for one scan before the input to the counter makes a 0–1 transition. Otherwise, there is no real transition and the counter will not count.

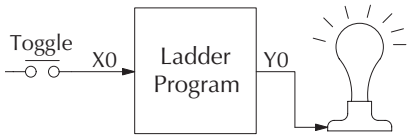
The ordinary Counter instruction does have a restriction inside stages: it may not be reset from other stages using the RST instruction for the counter bit. However, the special Stage counter provides a solution (see next paragraph).

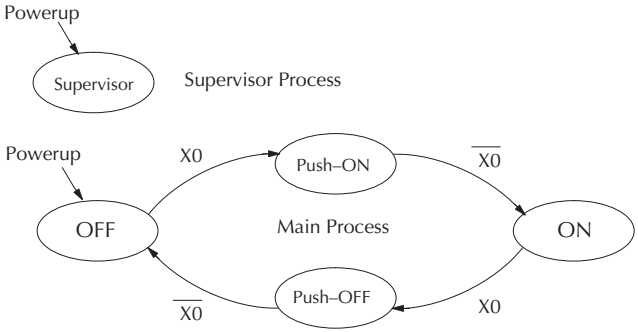
Stage Counter – The Stage Counter has the benefit that its count may be globally reset from other stages by using the RST instruction. It has a count input, but no reset input. This is the only difference from a standard counter.

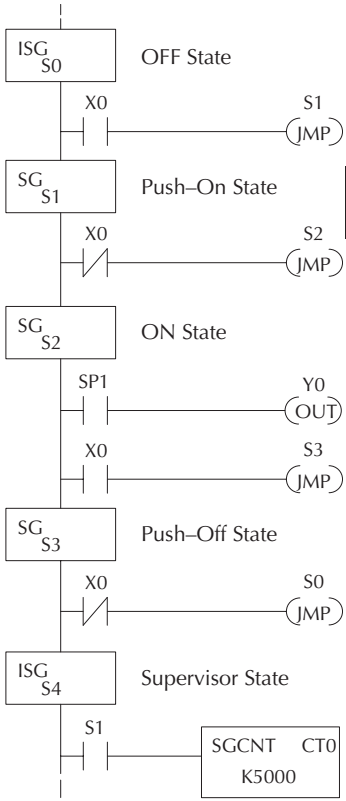
Drum – Realize that the drum sequencer is its own process, and is a different programming method than stage programming. If you need to use a drum with stages, be sure to place the drum instruction in an ISG stage that is always active.

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.

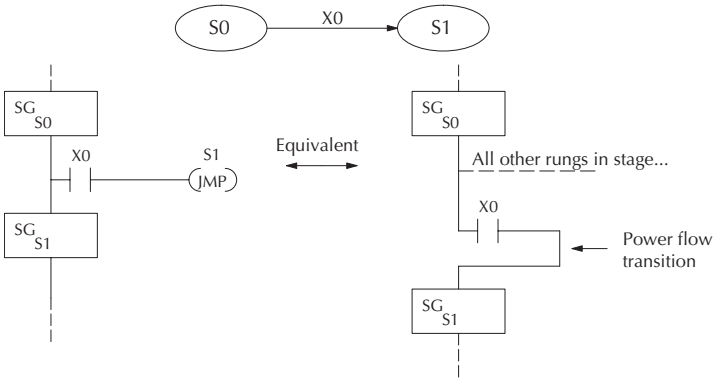
Stage Counter

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.

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*SOFT32, you may use the power flow method for stage transitions.

The main requirement is that 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.

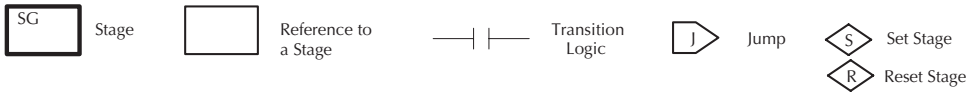


Recall that the Stage JMP instruction may occur anywhere in the current stage, and the result is the same. However, power flow transitions (shown above) must occur as the last rung in a stage. All other rungs in the stage will precede it. The power flow transition method is also achievable on the handheld programmer, by simply following the transition condition with the Stage instruction for the next stage.

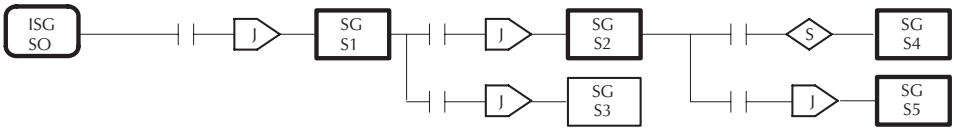
The power flow transition method does eliminate one Stage JMP instruction, its only advantage. However, it is not as easy to make program changes as using the Stage JMP. Therefore, we advise using Stage JMP transitions for most programmers.

Stage View in DirectSOFT32

The Stage View option in *Direct*SOFT32 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.



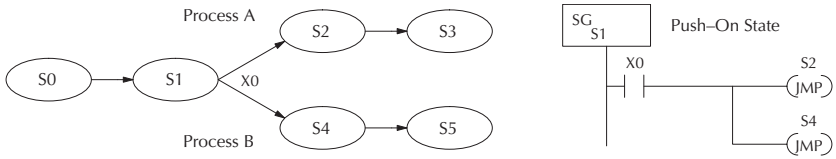
The following diagram is a typical stage view of a ladder program containing stages. Note the left-to-right direction of the flow chart.



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).

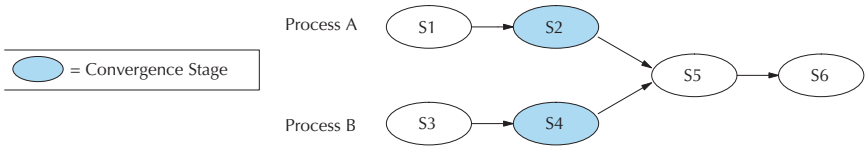


Note that if we want Stages S2 and S4 to energize exactly on the same scan, both stages must be located below or above Stage S1 in the ladder program (see the explanation at the bottom of page 7–7). Overall, parallel branching is easy!

7

Converging Processes

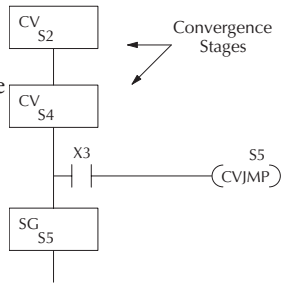
Now we consider the opposite case of parallel branching, which is converging processes. This simply means we stop doing multiple things and continue doing one thing at a time. In the figure below, processes A and B converge when stages S2 and S4 transition to S5 at some point in time. So, S2 and S4 are *Convergence Stages.*



Convergence Stages (CV)

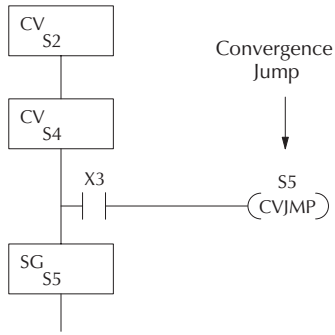
While the converging principle is simple enough, it brings a new complication. As parallel processing completes, the multiple processes almost never finish at the same time. In other words, how can we know whether Stage S2 or S4 will finish last? This is an important point, because we have to decide 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 16. In other words, a maximum of 16 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.

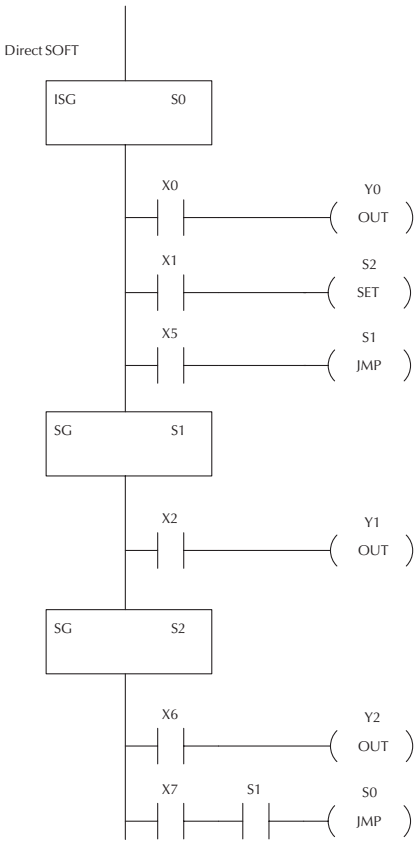
RLL^{PLUS} (Stage) Instructions

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	DL06 Range
	aaa
Stage S	0–1777

The following example is a simple RLL^{PLUS} program. This program utilizes an initial stage, stage, and jump instructions to create a structured program.

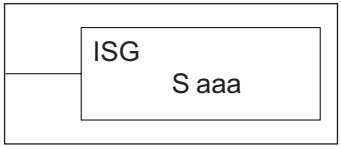


Handheld Programmer Keystrokes

U ISG	→	A 0	ENT		
$ STR	→	A 0	ENT		
GX OUT	→	A 0	ENT		
$ STR	→	B 1	ENT		
X SET	→	SHFT	S RST	C 2	ENT
$ STR	→	F 5	ENT		
K JMP	→	B 1	ENT		
2 SG	→	B 1	ENT		
$ STR	→	C 2	ENT		
GX OUT	→	B 1	ENT		
2 SG	→	C 2	ENT		
$ STR	→	G 6	ENT		
GX OUT	→	C 2	ENT		
$ STR	→	H 7	ENT		
V AND	→	SHFT	S RST	B 1	ENT
K JMP	→	A 0	ENT		

Initial Stage (ISG)

The Initial Stage instruction is normally used as the first segment of an RLL^{PLUS} program. Multiple Initial Stages are allowed in a program. They will be active when the CPU enters the Run mode allowing for a starting point in the program.

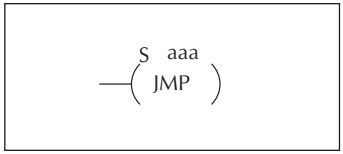


Operand Data Type	DL06 Range
	aaa
Stage . . . . . S	0–1777

Initial Stages are also activated by transitional logic, a jump or a set stage executed from an active stage.

Jump (JMP)

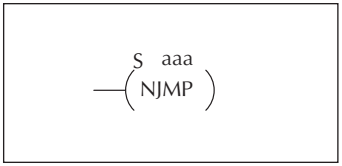
The Jump instruction allows the program to transition from an active stage containing the jump instruction to another stage (specified in the instruction). The jump occurs when the input logic is true. The active stage containing the Jump will deactivate 1 scan later.



Operand Data Type	DL06 Range
	aaa
Stage . . . . . S	0–1777

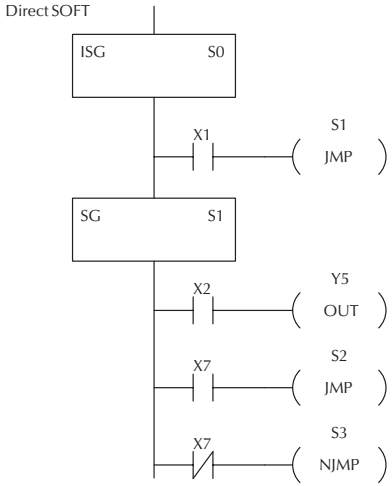
Not Jump (NJMP)

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	DL06 Range
	aaa
Stage . . . . . S	0–1777

In the following example, only stage ISG0 will be active when program execution. begins. When X1 is on, program execution will jump from Initial Stage 0 to Stage 1.



Handheld Programmer Keystrokes

U ISG	→	A 0	ENT				
$ STR	→	B 1	ENT				
K JMP	→	B 1	ENT				
2 SG	→	B 1	ENT				
$ STR	→	C 2	ENT				
GX OUT	→	F 5	ENT				
$ STR	→	H 7	ENT				
K JMP	→	C 2	ENT				
SHFT	N TMR	SHFT	K JMP	→	D 3	ENT	

7

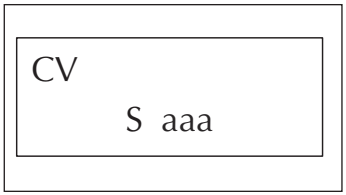
Converge Stage (CV) and Converge Jump (CVJMP)

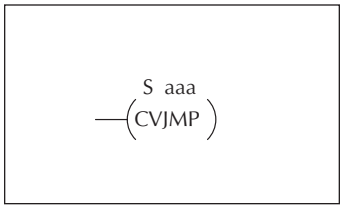
The Converge Stage instruction is used to group certain stages together by defining them as Converge Stages.

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 CVJMP 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.

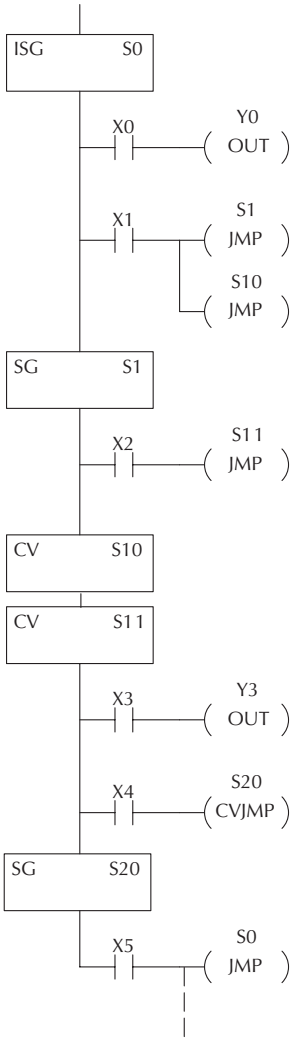




Operand Data Type	DL06 Range
	aaa
Stage . . . . . S	0–1777

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.

Direct SOFT Display



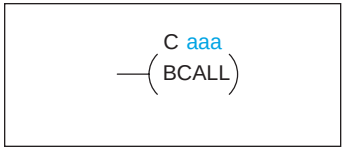
Handheld Programmer Keystrokes

U ISG	→	A 0	ENT							
$ STR	→	A 0	ENT							
GX OUT	→	A 0	ENT							
$ STR	→	B 1	ENT							
K JMP	→	B 1	ENT							
K JMP	→	B 1	A 0	ENT						
2 SG	→	B 1	ENT							
$ STR	→	C 2	ENT							
K JMP	→	B 1	B 1	ENT						
SHFT	C 2	V AND	→	B 1	A 0	ENT				
SHFT	C 2	V AND	→	B 1	B 1	ENT				
$ STR	→	D 3	ENT							
GX OUT	→	D 3	ENT							
$ STR	→	E 4	ENT							
SHFT	C 2	V AND	SHFT	K JMP	→	C 2	A 0	ENT		
2 SG	→	C 2	A 0	ENT						
$ STR	→	F 5	ENT							
K JMP	→	A 0	ENT							

Block Call (BCALL)

The stage block instructions are used to activate a block of stages. The Block Call, Block, and Block End instructions must be used together. 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.

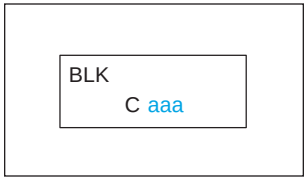


- Must Remain Active — The BCALL instruction actually controls all the stages between the BLK and the BEND instructions even after the stages inside the block have started executing. The BCALL must remain active or all the stages in the block will automatically be turned off. If either the BCALL instruction, 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	DL06 Range
	aaa
Control Relay . . . . . S	0–1777

Block (BLK)

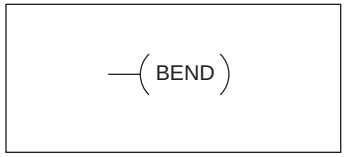
The Block instruction is a label which marks the beginning of a block of stages that can be activated as a group. A Stage instruction must immediately follow the Start Block instruction. Initial Stage instructions are not allowed in a block. The control relay (Caaa) specified in Block instruction must not be used as an output any where else in the program.



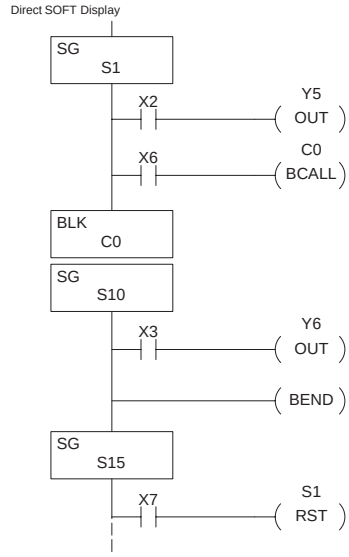
Operand Data Type	DL06 Range
	aaa
Control Relay . . . . . S	0–1777

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.

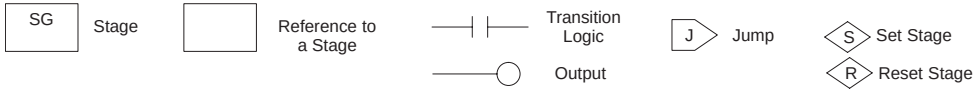


Handheld Programmer Keystrokes

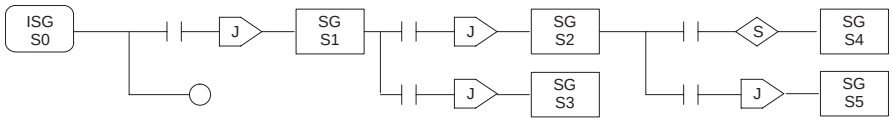
SG	→	S(SG)	1	ENT					
STR	→	X(IN)	2	ENT					
OUT	→	Y(OUT)	5	ENT					
STR	→	X(IN)	6	ENT					
SHFT	B	C	A	L	L	→	C(CR)	0	ENT
SHFT	B	L	K	→	C(CR)	0	ENT		
SG	→	S(SG)	1	0	ENT				
STR	→	X(IN)	3	ENT					
OUT	→	Y(OUT)	6	ENT					
SHFT	B	E	N	D	ENT				
SG	→	S(SG)	1	5	ENT				
STR	→	X(IN)	7	ENT					
RST	→	S(SG)	1	ENT					

Stage View in DirectSOFT32

The Stage View option in DirectSOFT32 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 can't 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. What are Stage Bits?

A. A stage bit is just 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 Reset 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*SOFT32, 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, just 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 just 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 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 just 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 have 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.

PID Loop Operation

In This Chapter...

DL06 PID Loop Features 8–2
Loop Setup Parameters 8–6
Loop Sample Rate and Scheduling 8–13
Ten Steps to Successful Process Control 8–17
Basic Loop Operation 8–19
PID Loop Data Configuration 8–28
PID Algorithms 8–33
Loop Tuning Procedure 8–40
PV Analog Filter 8–47
Feedforward Control 8–49
Cascade Control 8–53
Ramp/Soak Generator 8–59
Troubleshooting Tips 8–64
Bibliography 8–65
Glossary of PID Loop Terminology 8–66

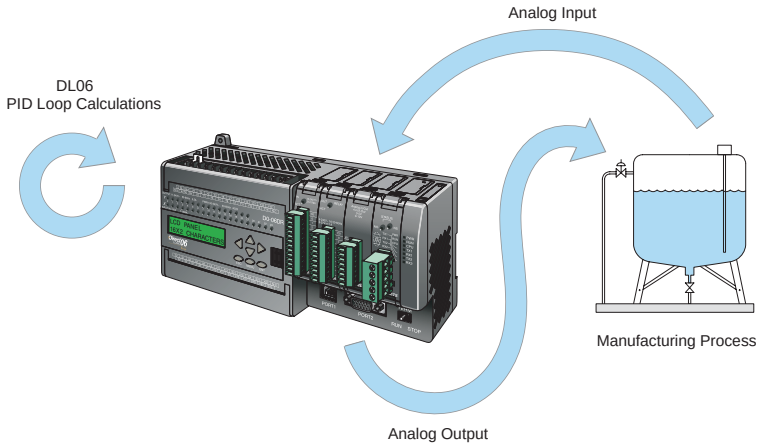
DL06 PID Loop Features

Main Features

The DL06 process loop control offers a sophisticated set of features to address many application needs. The main features are:

- Up to 8 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 DL06 CPU has process control loop capability in addition to ladder program execution. You can select and configure up to eight loops. All sensor and actuator wiring connects directly to DL06 analog modules. All process variables, gain values, alarm levels, etc., associated with each loop reside in a Loop Variable Table in the CPU. The DL06 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 a Proportional-Integral-Derivative (PID) algorithm to generate the control output. This chapter describes how the loops operate, and what you must do to configure and tune the



loops.

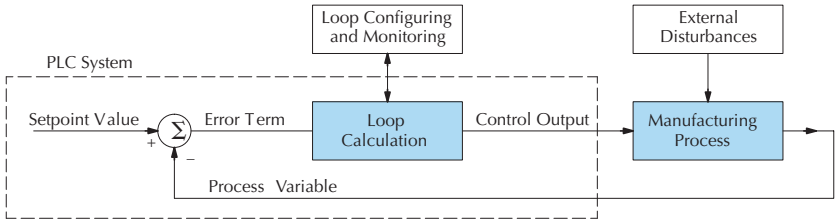
The best tool for configuring loops in the DL06 is the *Direct*SOFT32 programming software, release 4.0, or later. *Direct*SOFT32 uses dialog boxes to help you set up the individual loops. After completing the setup, you can use *Direct*SOFT32's PID Trend View to tune each loop. The configuration and tuning selections you make are stored in the DL06's FLASH memory, which is retentive. The loop parameters also may be saved to disk for recall later.

PID Loop Feature	Specifications
Number of loops	Selectable, 8 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, 0.1 seconds for 5 to 8 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

The Basics of PID Loops

The key parts of a PID control loop are shown in the block diagram below. The path from the PLC to the Manufacturing Process and back to the PLC is the "loop" 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, you will most likely want to control temperature. 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 DL06 uses the Proportional-Integral-Derivative (PID) algorithm (more on this later).

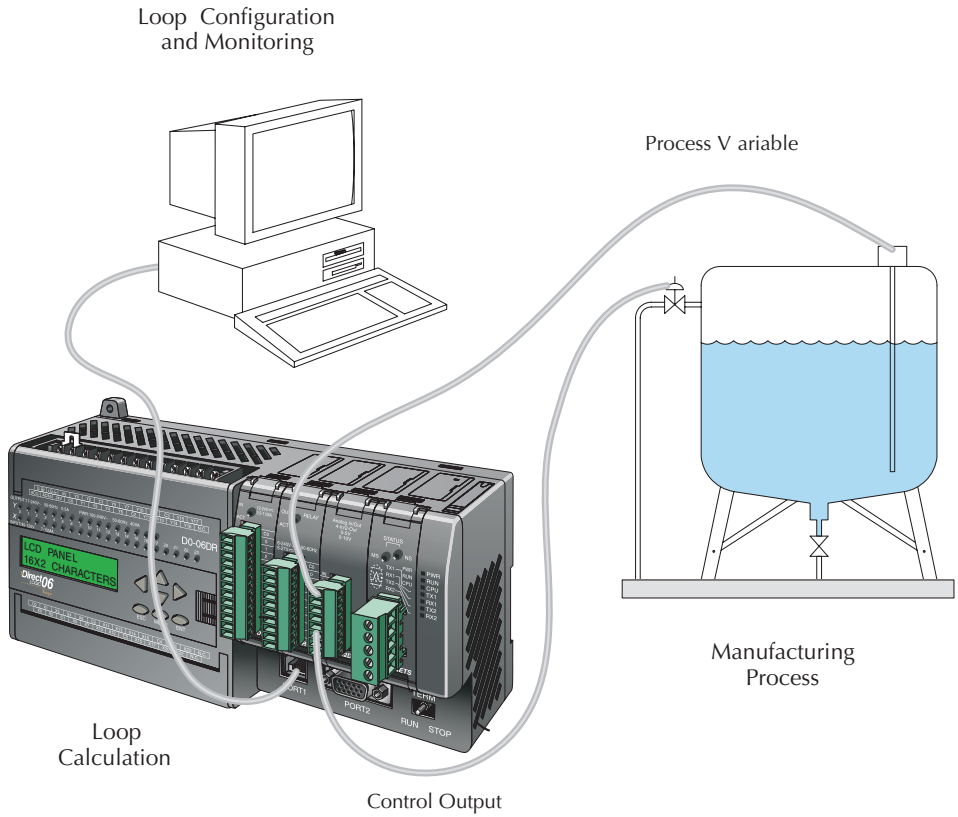
Control Output – the result of the loop calculation, which becomes a command for the process (such as the heater level in an oven).

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).

The diagram below shows each loop element in the form of its real-world physical component. The example manufacturing process involves a 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 its analog input. The CPU executes the loop calculation, and writes to the analog output. This signal goes to a device in the manufacturing process, such as 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 used to run *Direct*SOFT32, the PLC programming software for *Direct*LOGIC programmable controllers. *Direct*SOFT32, release 4.0 or later, can program the DL06 PLC (including the PID feature). The software features a forms-based editor to configure loop parameters. It also features a PID loop trending screen which will be helpful during the loop tuning process. Details on how to use that software are in the *Direct*SOFT32 Manual.

Loop Setup Parameters

Loop Table and Number of Loops

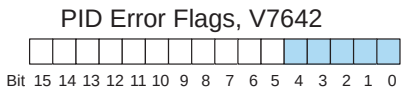
The DL06 PLC gets its PID loop processing instructions only from tables in V-memory. A "PID instruction" type in RLL does not exist for the *Direct*Logic PLCs. Instead, the CPU reads setup parameters from reserved V-memory locations. Shown in the table below, you must program a value in V7640 to point 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	V1200 V7340 V10000-V17740	write
V7641	Number of Loops	BCD	0 – 8	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.

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.



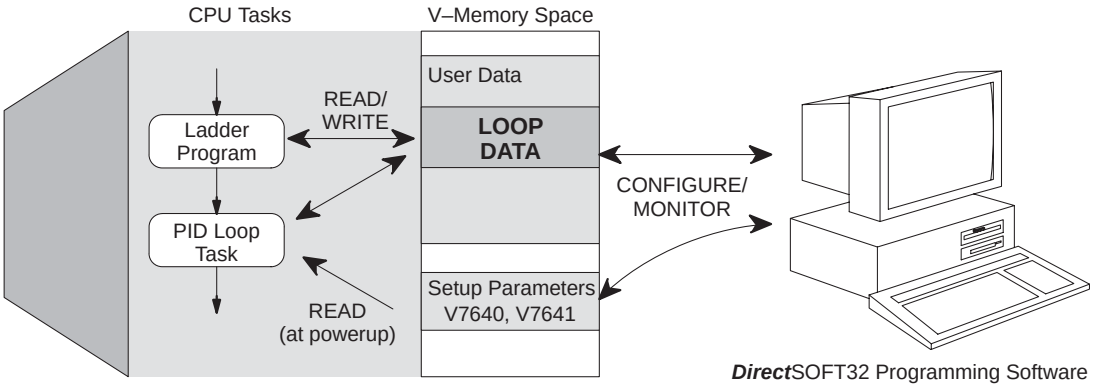
If you use the *Direct*SOFT32 loop setup dialog box, its automatic range checking prohibits possible setup errors. However, the setup parameters may be written using other methods such as RLL, so the error flag register may be helpful in those cases. The following table lists the errors reported in V7642.

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 8
3	The loop table extends past (straddles) the boundary at V7377. Use an address closer to V1200.

As a quick check, if the CPU is in Run mode and V7642=0000, there are no programming errors.

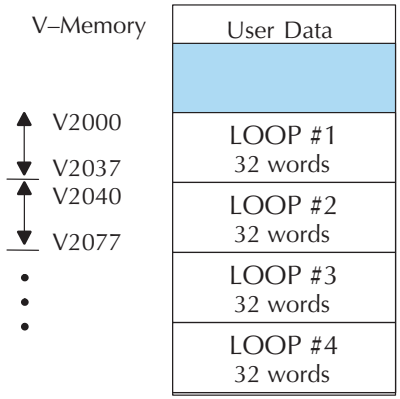
Establishing the Loop Table Size and Location

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 as you selected in V7641. Each loop configuration occupies 32 words (0 to 37 octal) in the loop table.

For example, suppose you have an application with 4 loops, and you 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# (in the first column) to calculate addresses.

Word #	Address+Offset	Description	Format	Read on the Fly
1	Addr + 0	PID Loop Mode Setting 1	bits	Yes
2	Addr + 1	PID Loop Mode Setting 2	bits	Yes
3	Addr + 2	Setpoint Value (SP)	word/binary	Yes
4	Addr + 3	Process Variable (PV)	word/binary	Yes
5	Addr + 4	Bias (Integrator) Value	word/binary	Yes
6	Addr + 5	Control Output Value	word/binary	Yes
7	Addr + 6	Loop Mode and Alarm Status	bits	-
8	Addr + 7	Sample Rate Setting	word/BCD	Yes
9	Addr + 10	Gain (Proportional) Setting	word/BCD	Yes
10	Addr + 11	Reset (Integral) Time Setting	word/BCD	Yes
11	Addr + 12	Rate (Derivative) Time Setting	word/BCD	Yes
12	Addr + 13	PV Value, Low-low Alarm	word/binary	No*
13	Addr + 14	PV Value, Low Alarm	word/binary	No*
14	Addr + 15	PV Value, High Alarm	word/binary	No*
15	Addr + 16	PV Value, High-high Alarm	word/binary	No*
16	Addr + 17	PV Value, deviation alarm (YELLOW)	word/binary	No*
17	Addr + 20	PV Value, deviation alarm (RED)	word/binary	No*
18	Addr + 21	PV Value, rate-of-change alarm	word/binary	No*
19	Addr + 22	PV Value, alarm hysteresis setting	word/binary	No*
20	Addr + 23	PV Value, error deadband setting	word/binary	Yes
21	Addr + 24	PV low-pass filter constant	word/BCD	-
22	Addr + 25	Loop derivative gain limiting factor setting	word/BCD	No**
23	Addr + 26	SP value lower limit setting	word/binary	Yes
24	Addr + 27	SP value upper limit setting	word/binary	Yes
25	Addr + 30	Control output value lower limit setting	word/binary	No**
26	Addr + 31	Control output value upper limit setting	word/binary	No**
27	Addr + 32	Remote SP Value V-Memory Address Pointer	word/hex	Yes
28	Addr + 33	Ramp/Soak Setting Flag	bit	Yes
29	Addr + 34	Ramp/Soak Programming Table Starting Address	word/hex	No**
30	Addr + 35	Ramp/Soak Programming Table Error Flags	bits	No**
31	Addr + 36	PV auto transfer, base/slot/channel number/pointer	word/hex	
32	Addr + 37	Control output auto transfer, channel number	word/hex	

*Read data only when alarm enable bit transitions 0 to 1
**Read data only on PLC Mode change

PID Mode Setting 1 Bit Descriptions (Addr + 00)

The individual bit definitions of the PID Mode Setting 1 word (Addr+00) are listed in the following table. Additional information about the use of this word is available 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 the PID Mode Setting 2 word (Addr+01) are listed in the following table. Additional information about the use of this word is available later in this chapter.

Bit	PID Mode Setting 2 Description	Read/Write	Bit=0	Bit=1
0	Input (PV) and Control Output Range Unipolar/Bipolar select (See Notes 1 and 2)	write	unipolar	bipolar
1	Input/Output Data Format select (See Notes 1 and 2)	write	12 bit	15 bit
2	Analog Input filter/Auto-Transfer	write	off	on
3	SP Input limit enable	write	disable	enable
4	Integral Gain (Reset) units select	write	seconds	minutes
5	Select Autotune PID algorithm	write	closed loop	open loop
6	Autotune selection	write	PID	PI only (rate = 0)
7	Autotune start	read/write	autotune done	force start
8	PID Scan Clock (internal use)	read	–	–
9	Input/Output Data Format 16-bit select (See Notes 1 and 2)	write	not 16 bit	select 16 bit
10	Select separate data format for input and output (See Notes 2, and 3)	write	same format	separate formats
11	Control Output Range Unipolar/Bipolar select See Notes 2, and 3)	write	unipolar	bipolar
12	Output Data Format select (See Notes 2, and 3)	write	12 bit	15 bit
13	Output data format 16-bit select (See Notes 2, and 3)	write	not 16 bit	select16 bit
14–15	Reserved for future use	–	–	–

Note 1: If the value in bit 9 is 0, then the values in bits 0 and 1 are read. If the value in bit 9 is 1, then the values in bits 0 and 1 are not read, and bit 9 defines the data format (the range is automatically unipolar).

Note 2: If the value in bit 10 is 0, then the values in bits 0, 1, and 9 define the input and output ranges and data formats (the values in bits 11, 12, and 13 are not read). If the value in bit 10 is 1, then the values in bits 0, 1, and 9 define only the input range and data format, and bits 11, 12, and 13 are read and define the output range and data format.

Note 3: If bit 10 has a value of 1 and bit 13 has a value of 0, then bits 11 and 12 are read and define the output range and data format. If bit 10 and bit 13 each have a value of 1, then bits 11 and 12 are not read, and bit 13 defines the data format, (the output range is automatically unipolar).

Mode / Alarm Monitoring Word (Addr + 06)

The individual bit definitions of the Mode / Alarm monitoring (Addr+06) word is listed in the following table. More details are in the PID Mode section and 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 Auto-Tune indication	read	Off	On
13	Auto-Tune error indication	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 is listed in the following table. Further details are given in the Ramp / Soak Operation section.

Bit	Ramp / Soak Flag Bit Description	Read/Write	Bit=0	Bit=1
0	Start Ramp / Soak Profile	write	–	01 Start
1	Hold Ramp / Soak Profile	write	–	01 Hold
2	Resume Ramp / soak Profile	write	–	01 Resume
3	Jog Ramp / Soak Profile	write	–	01 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)	

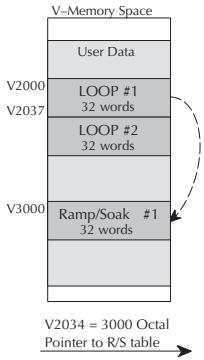
Bits 8–15 must be read as a byte to indicate the current segment number of the Ramp/Soak generator in the profile. This byte will have the values 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and 10, which represent segments 1 to 16 respectively. If the byte=0, then the Ramp/Soak table is not active.

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 that follow a profile. To use the Ramp Soak feature, you must program a separate table of 32 words with appropriate values. A *Direct*SOFT32 dialog box makes this easy to do.

In the 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 adjoin 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 Operation 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** word (Addr+35) is listed in the following table. Further details are given in the PID Loop Mode section and in the PV Alarm 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

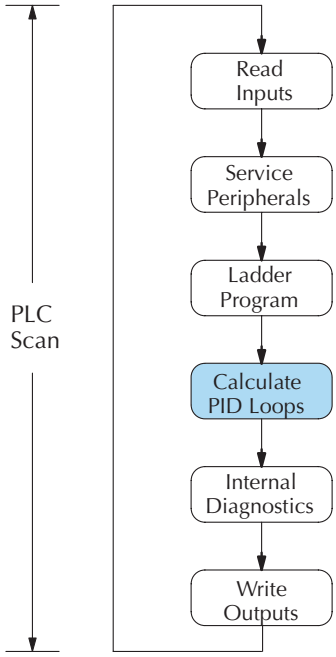
Loop Sample Rates Addr + 07

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.

Note: It is possible to keep the PID loops running even when the ladder is not. This is done by selecting direct access in Addr + 36 and placing a 1 in bit 15 of Addr + 00.

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 DL06 CPU, you can set the sample rate of a loop from 50 mS to 99.99 seconds. Most loops do not require a fresh PID calculation on every PLC scan. Some loops may need to be calculated 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

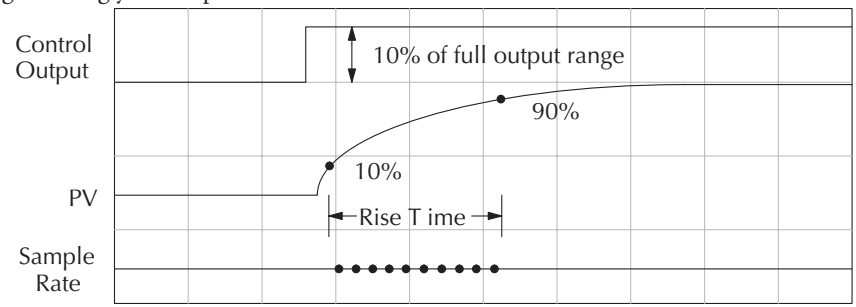
For any particular control loop, there is no single perfect sample rate to use. A good sample rate is a compromise that simultaneously satisfies various guidelines:

- 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, determine a sample rate for your loop which will be fast enough to avoid control instability (which is extremely important). Follow the procedure on the next page to find a starting sample rate:

Determining a suitable sample rate (Addr+07):

1. Operate the process open-loop (the loop does not even need to be configured yet). Place the CPU in run mode (and the loop in Manual mode, if you have already configured it). 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.

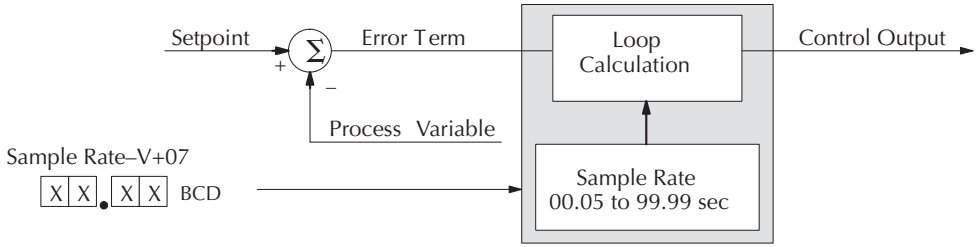


In the figure above, suppose the measured rise time response of the PV was 25 seconds. The suggested sample rate from this measurement will be 2.5 seconds. For illustration, the sample rate time line shows ten samples within the rise time period. These show the frequency of PID calculations as the PV 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 a data location for the sample rate. Referring to the figure below, location addr+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***SOFT32'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). 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:

$$\text{Avg. Scan Time with PID loop} = \left[\frac{\text{Scan time without loop}}{\text{Sample rate of loop}} \times \text{PID calculation time} \right] + \text{Scan time without loop}$$

$$\text{Average Scan time with PID loop} = \left[\frac{50\ \text{mS}}{3\ \text{sec.}} \times 250\ \mu\text{S} \right] + 50\ \text{mS} = 50.004\ \text{mS}$$

As the calculation shows, the addition of only one loop with a slow sample rate has a very small effect on scan time. Next, expand the 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, you 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 you have a DL06 PLC 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

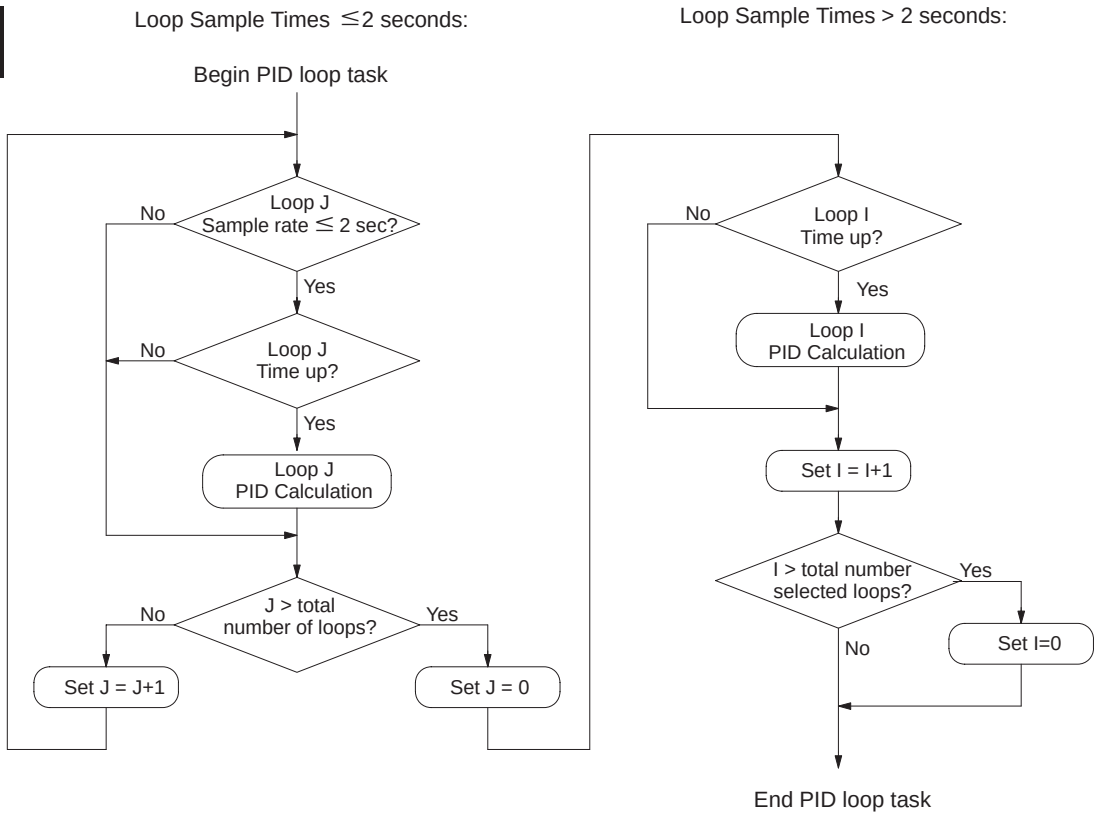
Now adding the summation terms, plus the original scan time value, we have:

$$\text{Avg. Scan Time with PID loops} = \left[50\ \mu\text{S} + 0.42\ \mu\text{S} + 1.25\ \mu\text{S} + 8.3\mu\text{S} \right] + 50\ \text{mS} = 50.06\ \text{mS}$$

The DL06 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 less than or equal to 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 more than 2 seconds are processed at the rate of one or fewer 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 (more than 2 sec) and fast (less than or equal to 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.



Ten Steps to Successful Process Control

Modern electronic controllers such as the DL06 CPU provide sophisticated process control features. Automated control systems can be 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 Variables 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

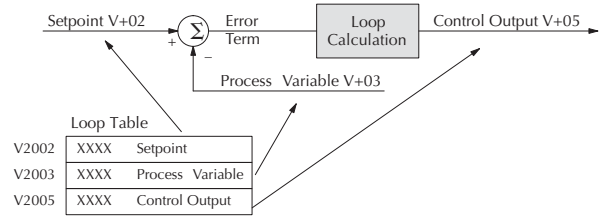
Assuming the control strategy is sound, it is still crucial to *properly size the actuators and properly scale the sensors.*

- 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 DL06 provides 12–bit, 15–bit and 16- bit unipolar and bipolar data format options, and a 16–bit unipolar option. 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, you 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, or use a analog I/O combination module, among several control loops. The example shown sends the PV and Control Output signals for two loops through the same set of modules.

Automationdirect offers DL06 analog input modules with 4 channels per module that accept 0 – 20mA or 4 – 20mA signals. Also, analog input and output combination modules are now available. Refer to the sales catalog for further information on these modules, or find the modules on our website, **www.automationdirect.com.**



Step 5: Wiring and Installation

After selection and procurement of all loop components and I/O module(s), you can perform the wiring and installation. Refer to the wiring guidelines in Chapter 2 of this Manual, and to the **D0–OPTIONS–M** manual. The most common wiring errors when installing PID loop controls are:

- Reversing the polarity of sensor or actuator wiring connections.
- Incorrect signal ground connections between loop components.

Step 6: Loop Parameters

After wiring and installation, choose the loop setup parameters. The easiest method for programming the loop tables is using ***Direct***SOFT32 (4.0 or later). This software provides PID Setup dialog boxes which simplify the task. **Note**: It is important to understand the meaning of all loop parameters mentioned in this chapter before choosing values to enter.

8

Step 7: Check Open Loop Performance

With the sensor and actuator wiring done, and loop parameters entered, we must manually and carefully check out the new control system (use Manual Mode).

- Verify that 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 (see Loop Tuning on page 8–38) shows the PV reading is correct and the control output has the proper effect on the process, you can follow the closed loop tuning procedure (see Automatic Mode on page 8–39). In this step, you tune the loop so the PV automatically follows the SP.

Step 9: Run Process Cycle

If the closed loop test shows the PV will follow small changes in the SP, consider running an actual process cycle. You will need to have completed the programming which will generate the desired SP in real time. In this step, you may want to run a small test batch of product through the machine, watching the SP change according to the recipe.

Step 10: Save Parameters

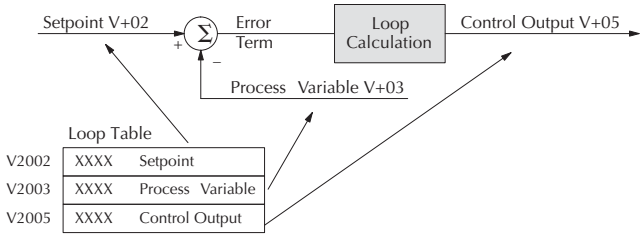
When the loop tests and tuning sessions are complete, be sure to save all loop setup parameters to disk.

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.

Basic Loop Operation

Data Locations

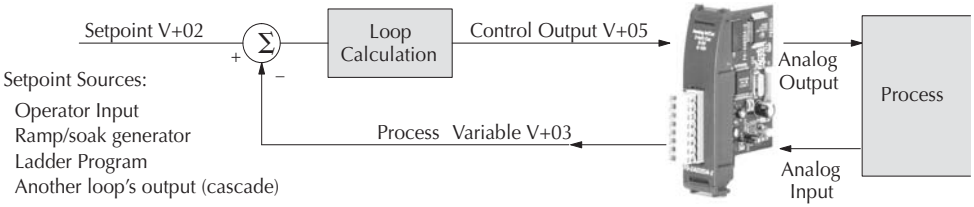
Each PID loop is dependent on the instructions and data values in its respective loop table. The following diagram shows an example of the loop table locations corresponding to the main three loop variables: SP, PV, and Control Output. The example below begins at V2000 (you can use any memory location compatible with Loop Table requirements). 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-world 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 analog input/output combination module to interface with the process itself.

A few rungs of ladder logic are required to copy data from the analog module to the loop table, or vice versa. Refer to the analog module chapter of this manual for an example of the required ladder logic.



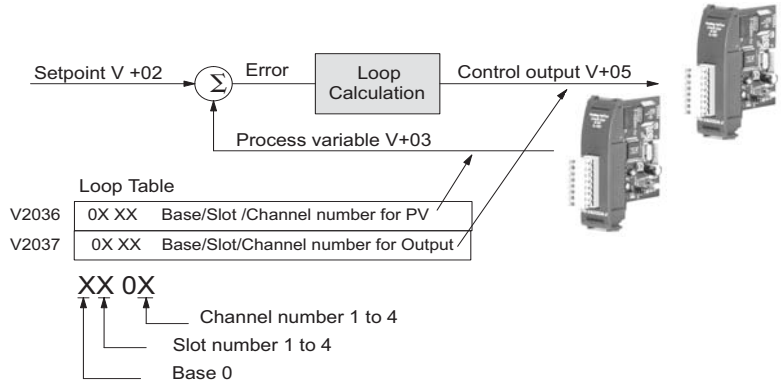
The Setpoint has several possible sources, as listed above. Many applications will use two or more of the sources at different times, depending on the loop mode. In addition, the loop control strategy and programming method also determine how the setpoint is generated.

When using the built-in Ramp/Soak generator or when cascading a loop, the PID controller automatically writes the setpoint data in location addr+02 for you. **If you want to use a setpoint from any other source, the ladder program must write that setpoint to the loop table location addr+02.**

Each of the three main loop parameters can have only one source or destination at any given time. During the application development, it is a good idea to draw loop schematic diagrams showing data sources, etc., to help avoid mistakes.

Auto Transfer to Analog I/O

The loop controller in the DL06 CPUs has the ability to directly access (referred to as auto transfer) analog I/O values or V–memory registers apart from the ladder logic scan. In particular, these parameters are the process variable (PV) and the control output. This feature is helpful if you must perform closed-loop PID control while the CPU is in Program Mode or if you wish to use the pointer method for the analog I/O or calculations in ladder logic to provide the PV values when in RUN 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 to the desired output module. This auto transfer feature, when enabled, accesses the analog values only once per PID calculation for each respective loop. 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 addr+36 and addr+37 and their role in direct access to the analog values.



You may program these loop table parameters directly, or use the PID Setup feature in DirectSOFT32 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. 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 auto transfer to/from I/O function of the loop controller. In this case, ladder logic will need to be used to perform the math and transfer the data to or from the analog modules as required.

NOTE: If the auto transfer to/from I/O function is used, the analog data for all of the channels on the analog modules being used with this feature cannot be accessed by any other method, i.e., pointer or multiplex.

PV Auto Transfer Functions with Filtering Options

The built-in filter uses the following algorithm **:**

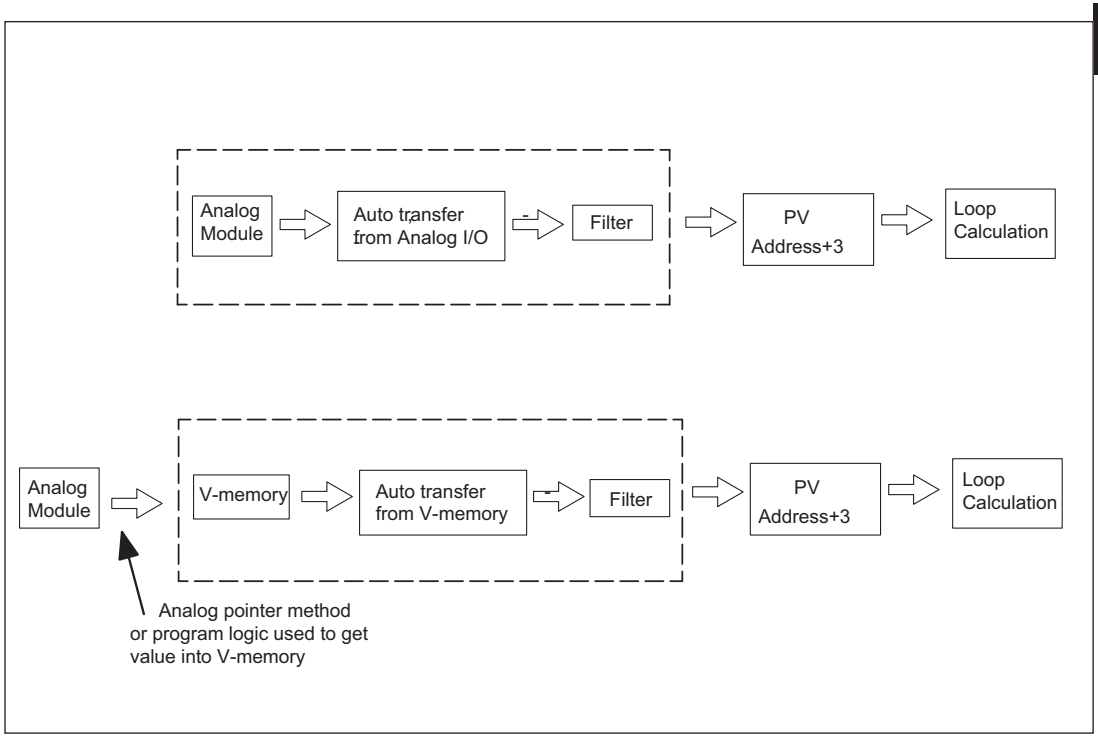
$$y_i = k\,(x_i - y_{i-1}) + y_{i-1}$$

- y_i is the current output of the filter
- x_i is the current input to the filter
- y_{i-1} is the previous output of the filter
- k is the PV Analog Input Filter Factor

The diagrams below show how the auto transfer function (address + 36) and PV filtering (address + 01, bit 2) interact. The options are:

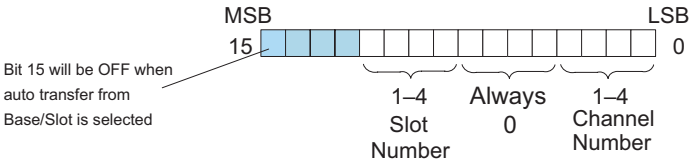
• Auto transfer directly from an analog I/O module channel with the filter enabled or disabled. When this function is used, the analog pointer method cannot be used to read the module's channel values.

• Auto–transfer directly from a V–memory location with the filter enabled or disabled. When this function is used, either the analog pointer method or program logic must be used to write a value to the V–memory location specified.



PV Auto Transfer (Addr + 36) from I/O Module Base/Slot/Channel Option

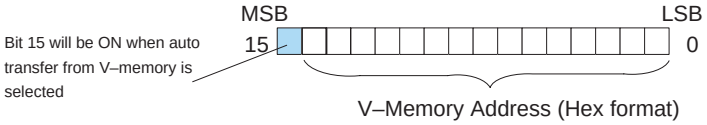
The nibble definitions for PV Auto Transfer word (Addr + 36) are listed in the table below for the Transfer from Base/Slot option. When this option is used for any channel on an analog input module, the ladder logic pointer method cannot be used for this module. (Refer to the DL06 Analog I/O Modules (D0–OPTIONS–M) for pointer method information).



Type	Slot number	Channel number
DL06	1-4	1-4

PV Auto Transfer (Addr + 36) from V–memory Option

The definitions for PV Auto Transfer word (Addr + 36) are listed in the table below for the Transfer from V–memory option. The ladder logic pointer method can be used with this option to get the analog module's channel values into V–memory. (Refer to the DL06 Analog I/O Modules (D0–OPTIONS–M) for pointer method information).

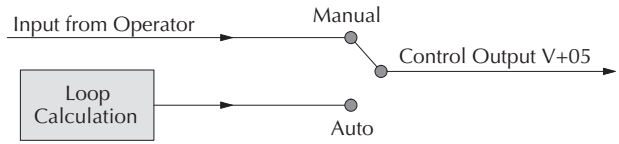


Type	V-Memory Range (Data Area)
DL06	V1200-V7377 / V10000-V17777

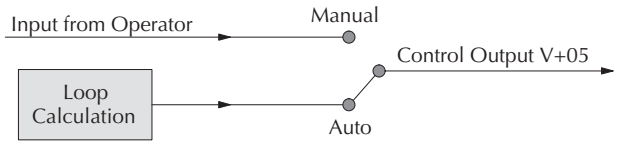
Loop Modes

The DL06 gives you the three standard control modes: *Manual, Automatic,* and *Cascade*. The sources of the three basic variables SP, PV, and control output are different for each mode. An introduction to the three control modes and their signal sources follows.

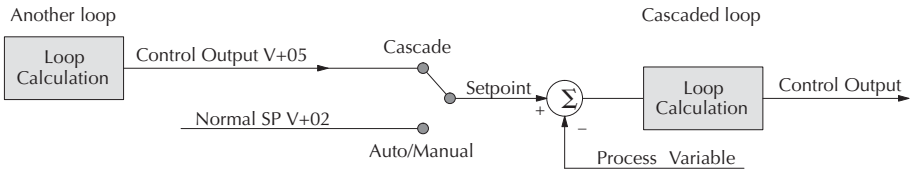
In **Manual Mode**, the loop is not executing PID calculations (however, loop alarms are still active). With regard to the loop table, the CPU stops writing values to location addr+05 for that loop. It is expected that an operator or other intelligent source is manually controlling the output, by observing the PV and writing data to addr+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 addr+05 every sample period of that loop. The equivalent schematic diagram is shown below.



In **Cascade Mode**, the loop operates as it does in Automatic Mode, with one important difference. The data source for the SP changes from its normal location at addr+02, using the control output value from another loop. So in Auto or Manual modes, the loop calculation uses the data at addr+02. In Cascade Mode, the loop calculation reads the control output from another loop's parameter table.



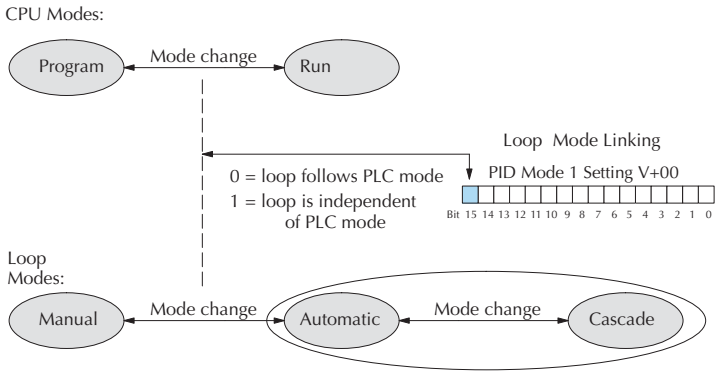
As pictured below, A loop can be changed from one mode to another, but *cannot go from Manual Mode directly to Cascade*, *or vice versa*. This mode change is prohibited because a loop would be changing two data sources at the same time, and could cause a loss of control.

CPU Modes and Loop Modes

The DL06 PLC has the ability to run PID calculations while the CPU is in Program Mode. Usually, a CPU in Program Mode has halted all operations. However, a DL06 PLC in Program Mode may or may not be running PID calculations (and providing PID control output), depending on your configuration settings. Having the ability to run loops independent 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.

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 of 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 (addr+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

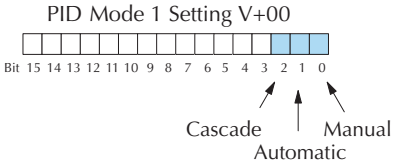


automatically affect the modes of the loops by changing the CPU mode.

If Bit 15 is set to one, then the loops will run independent 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**: To make changes to any **loop table parameter values**, the PID loop must be in Manual Mode and the PLC must be stopped. If you have selected (as shown above) to operate the PID loop independent of the CPU mode, then you must take certain steps to make it possible to make loop parameter changes. You can temporarily make the loops follow the CPU mode by changing bit 15 to 0. Then your programming device (such as **Direct**SOFT32) will be able to place the loop into Manual Mode. After you change the loop's parameter setting, just restore bit 15 to a value of 1 to re-establish PID operation independent of the CPU.*

How to Change Loop Modes

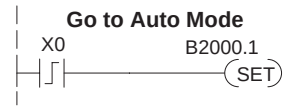


The first three bits of the PID Mode 1 word (addr+00) request the operating mode of the corresponding loop. Note: these bits are mode change *requests*, not commands (certain conditions can prohibit a particular mode change – see next page).

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:

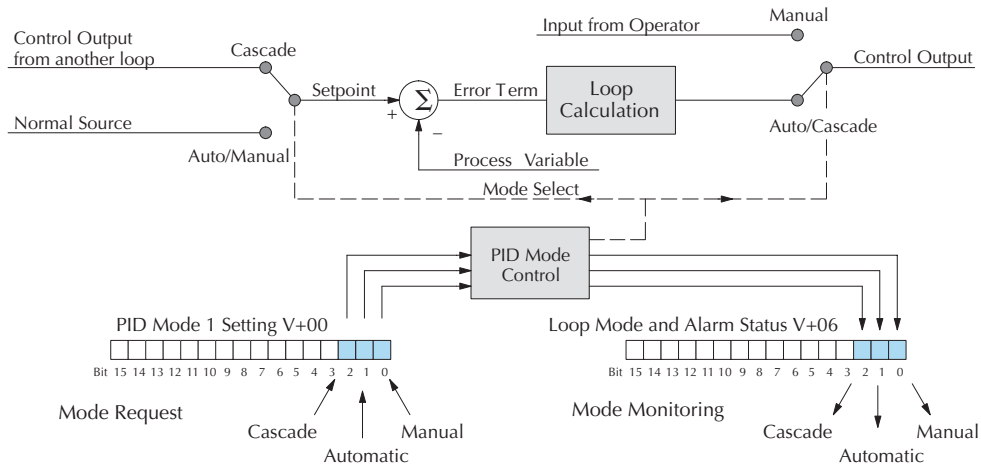
- ***Direct***SOFT32's **PID View** – this is the easiest method. Click on one of the radio buttons, and ***Direct***SOFT32 sets the appropriate bit.
- **HPP** – Use Word Status (WD ST) to monitor the contents of addr+00, which will be a 4-digit BCD/hex value. You must calculate and enter a new value for addr+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.

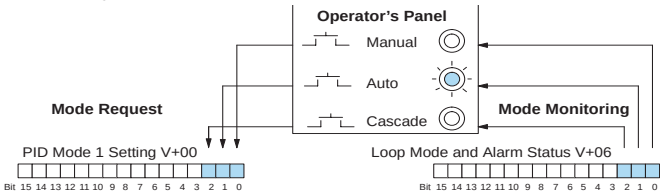
Since we can only *request* mode changes, the PID loop controller decides when to permit mode changes and provides the loop mode status. It reports the current mode on bits 0, 1, and 2 of the Loop Mode and Alarm Status word, location addr+06 in the loop table. The parallel request / monitoring functions are shown in the figure below. The figure also shows the two possible mode-dependent 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 8 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 addr+00, bits 0, 1, and 2. However, some conditions exist which will prevent a requested mode change from occurring:

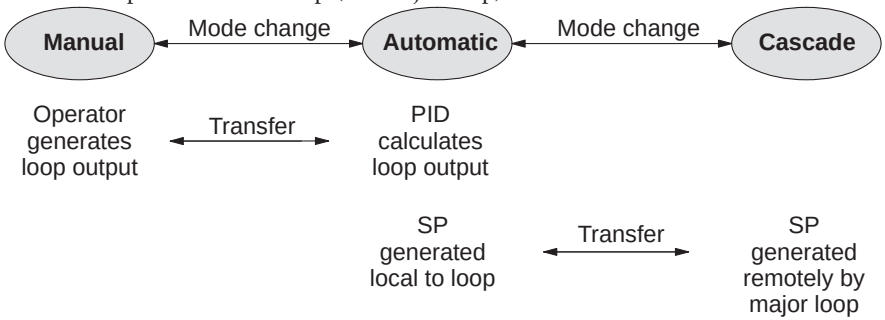
- A loop that is not set independent of PLC mode cannot change modes when the PLC is in Program mode.
- A major loop of a cascaded pair of loops cannot go from Manual to Auto until its minor loop is in Cascade mode.

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.

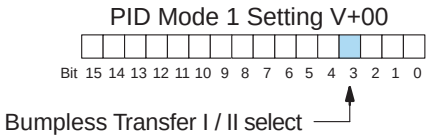
Bumpless Transfers

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 DL06 loop controller is available in two types: Bumpless I, and Bumpless II. Use ***Direct***SOFT32's PID Setup dialog box to select transfer type. Or, you can use bit 3 of PID Mode 1 addr+00 setting as shown.



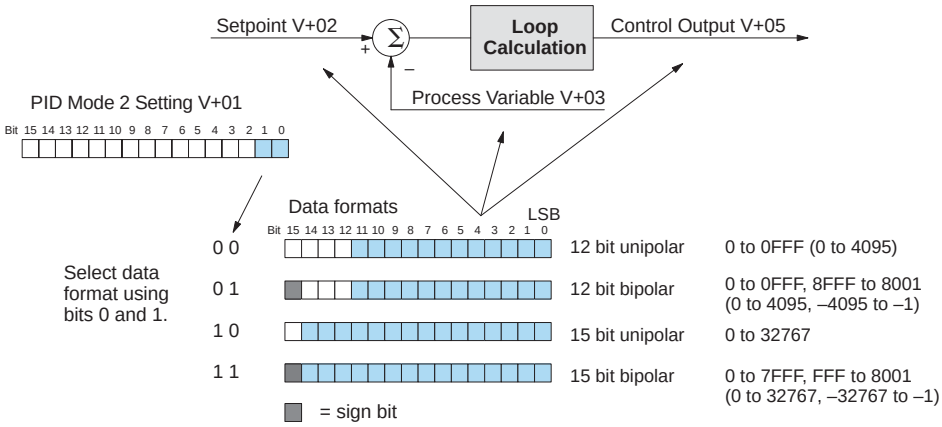
The characteristics of Bumpless I and II transfer types are listed in the chart below. Note that their operation also depends on which PID algorithm you are using, the position or velocity form of the PID equation. Note that you must use Bumpless Transfer type I when using the velocity form of the PID algorithm.

TransferType	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 addr+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 addr+01 choose the format. The *Direct*SOFT32 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 DL06 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.

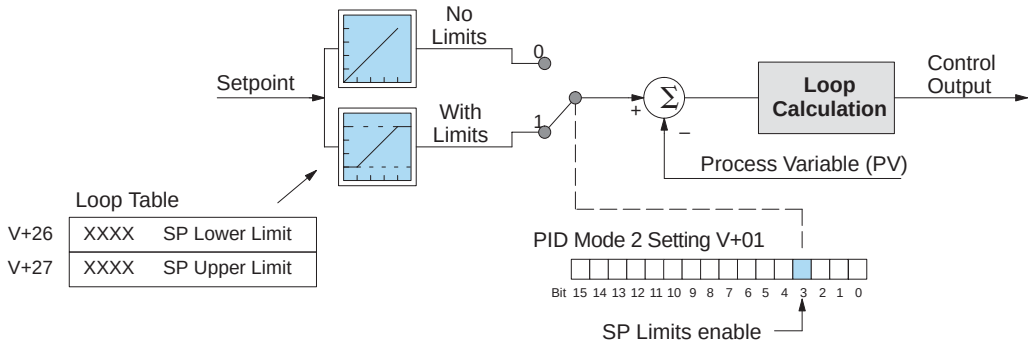
- **PV Offset** – In the event you have a PV value with a 20% offset, convert it to zero–offset by subtracting 20% of the top of its range, and multiply by1.25.
- **Control Output** – In the event the Control Output is going to a device with 20% offset, all you need to do is have the ladder program write a value equivalent to the offset to the integrator register (addr+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 addr+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.

8

In the figure below, the SP has a selectable limit function, enabled by PID Mode 2 Setting addr+01 word, bit 3. If enabled, then locations addr+26 and addr+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 addr+02.

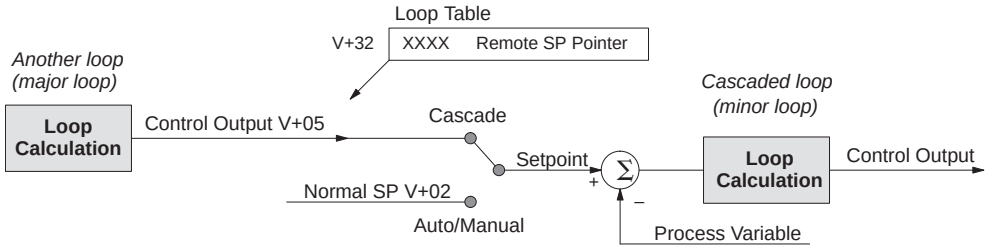


The loop calculation checks these SP upper and lower limits before each calculation. This means ladder logic can change the limit settings while a process is in progress, allowing you to keep a tighter guard band on the SP input value.

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 addr+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 addr+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.

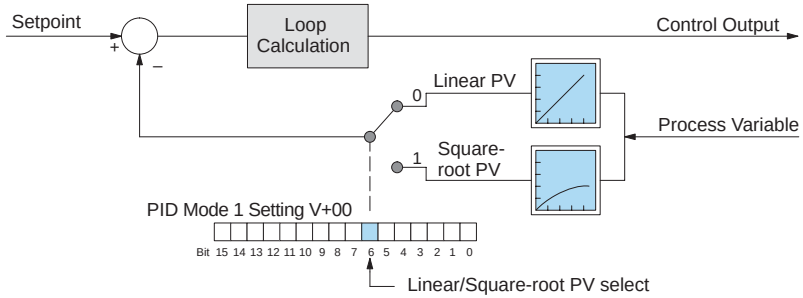


A *Direct*SOFT32 Loop Setup dialog box will allow you to enter the Remote SP pointer if you know the address. Otherwise, you can enter it with a HPP or program it through ladder logic using the LDA instruction.

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 addr+00 word, bit 6.



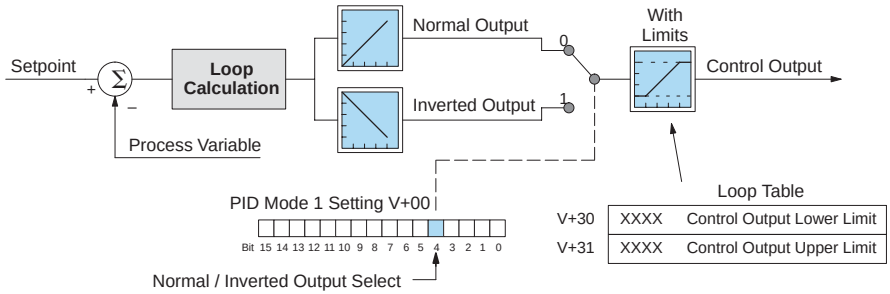
IMPORTANT: The scaling of the SP must be adjusted if you use PV square-root extract, because the loop drives the output so the square root of the PV is equal to the PV input. Divide the desired SP value by the square root of the analog span, and use the result in the addr+02 location for the SP. This 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
16-bit	SP = PV input / 256	0 – 256	0 – 65535

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 addr+30 and addr+31 may be set once using *Direct*SOFT32's PID Setup dialog box.

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*SOFT32). Use bit 4 of the PID Mode 1 Setting addr+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.

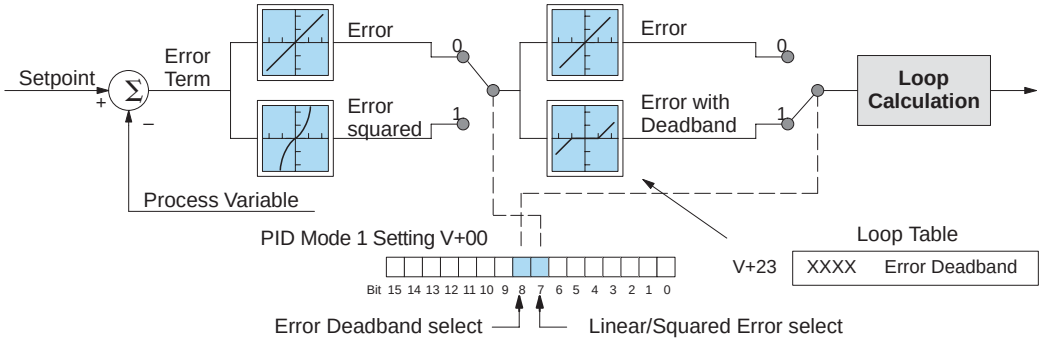
The normal/inverted output selection is used to configure direct-acting/reverse-acting loops. This selection is ultimately determined by the direction of the response of the process variable to a change in the control output in a particular direction. Refer to the PID Algorithms section for more on direct-acting 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 addr+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 addr+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.

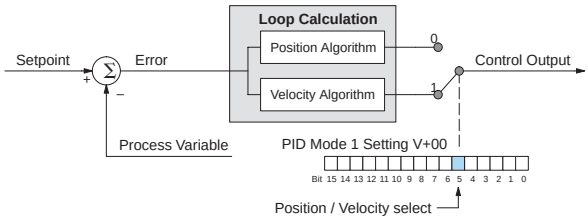
PID Algorithms

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 DL06 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 DL06 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 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***SOFT32's PID View Setup dialog box to select the desired algorithm. Or, use bit 5 of PID Mode 1 Setting addr+00 word as shown below to select the 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 Mn:

$$M_n = K_c * e_n + K_i * \sum_{i=1}^{n} e_i + K_r * (e_n - e_{n-1}) + M_o$$

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:

$$Mx_o = M_o$$

$$Mx_n = K_i * e_n + Mx_{n-1}$$

$$M_n = K_i * \sum_{i=1}^{n} e_i + M_o$$

$$M_n = K_c * e_n + K_r * (e_n - e_{n-1}) + Mx_n \text{.....Output for sampling time "n"}$$

The position algorithm variables and related variables are:

Ts = Sample rate

Kc = Proportional gain

Ki = Kc * (Ts/Ti) coefficient of integral term

Kr = Kc * (Td/Ts) coefficient of derivative term

Ti = Reset time (integral time)

Td = Rate time (derivative time)

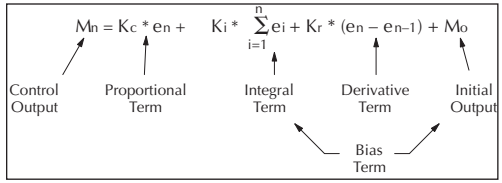
SPn = Set Point for sampling time "n" (SP value)

PVn = Process variable for sampling time "n" (PV)

en = SPn – PVn = Error term for sampling time "n"

M0 = Control Output for sampling time "0"

Mn = 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 above.

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:

Ts = Sample rate

Kc = Proportional gain

Ki = Kc * (Ts/Ti) = coefficient of integral term

Kr = Kc * (Td/Ts) = coefficient of derivative term

Ti = Reset time (integral time)

Td = Rate time (derivative time)

SPn = Set Point for sampling time "n" (SP value)

PVn = Process variable for sampling time "n" (PV)

en = SPn – PVn = Error term for sampling time "n"

Mn = Control Output for sampling time "n"

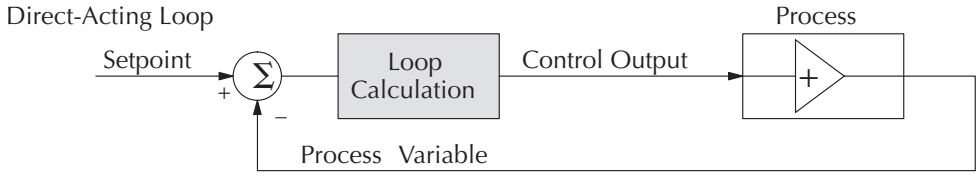
The resulting equations for the Velocity Algorithm form of the PID equation are:

$$\Delta M_n = M_n - M_{n-1}$$
$$\Delta M_n = K_c * (e_n - e_{n-1}) + K_i * e_n + K_r * (e_n - 2*e_{n-1} + e_{n-2})$$

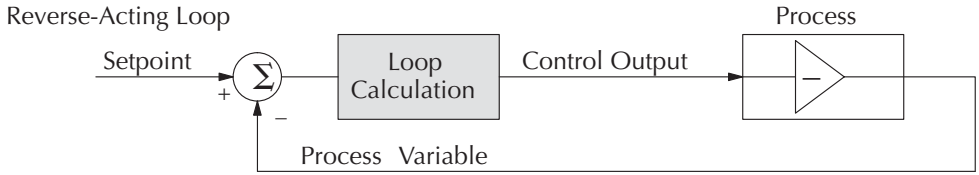
Direct-Acting and Reverse-Acting Loops

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*.

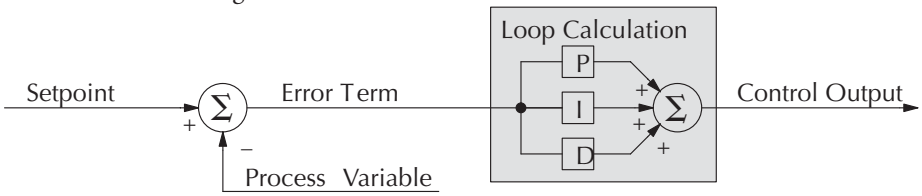


It is crucial to know whether a particular loop is direct or reverse-acting! Unless you are controlling temperature, there is no obvious answer. In a flow control loop, a valve positioning circuit can be configured and wired reverse-acting as easily as direct-acting. One easy way to find out is to run 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.

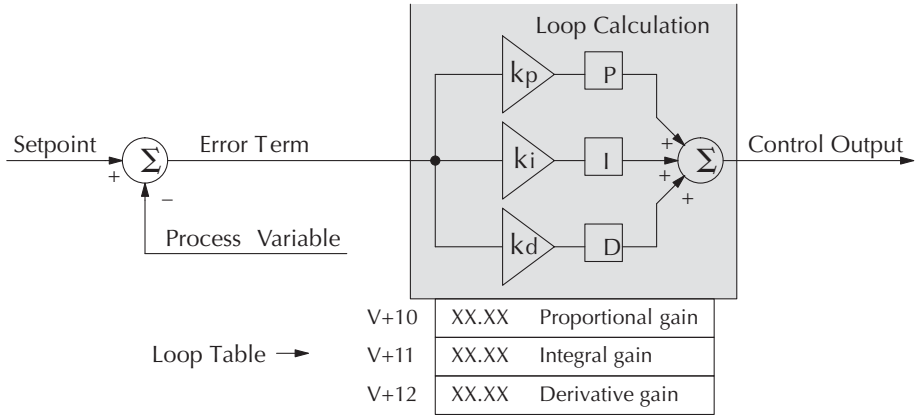


The role of the P, I, and D terms in the control task are as follows:

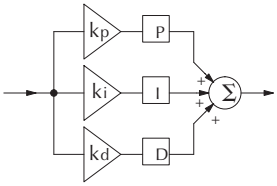
- **Proportional** – the proportional term simply responds proportionally to the current size of the error. This loop controller calculates a proportional term value for each PID calculation. When the error is zero, the proportional term is also zero.
- **Integral** – the integrator (or reset) term integrates (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 kp, ki, and kd 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.

PID Mode 2 Setting V+01

Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Units select

In *Direct*SOFT32's trend view, you can program the gain values and units in realtime 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.

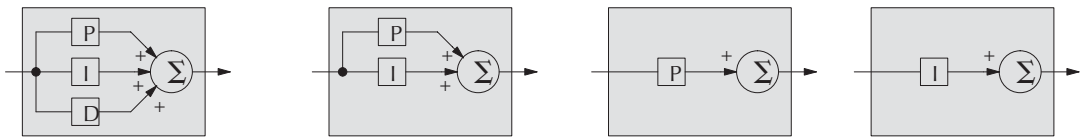
Integral Gain – Values range from 0001 to 9998, but they are used internally as xx.xx. An entry of "0000" or "9999"causes the integral gain to be "∞", effectively removing the integrator term from the PID equation. This accommodates applications which need proportional-only loops. The units of integral 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.*

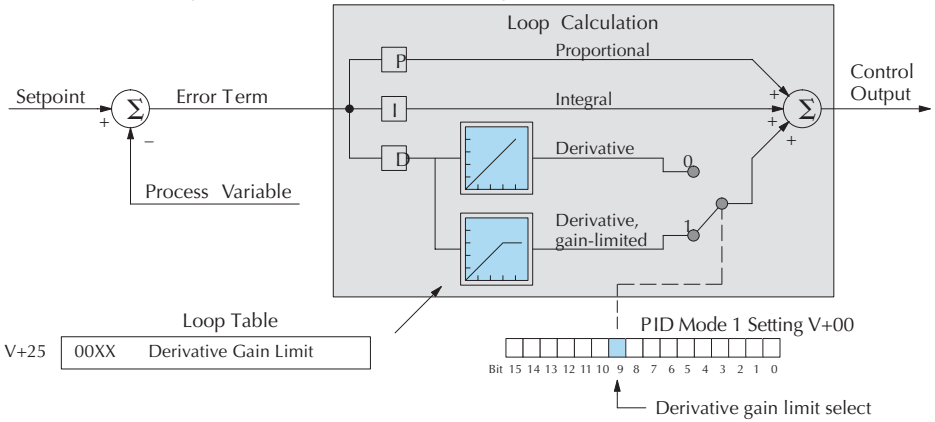
Using a Subset of PID Control

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 DL06. 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 addr+00 word to enable the gain limit.



The derivative gain limit in location addr+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 addr+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

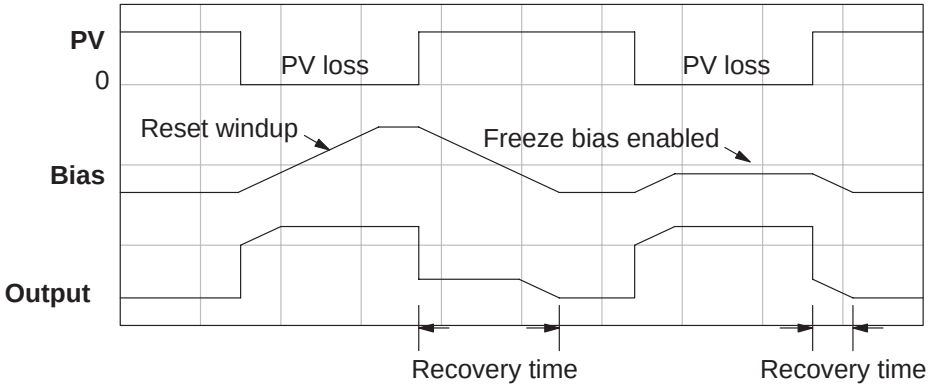
V+04 XXXX Bias term — Bias Term

If we cause the error (en) 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 (Mo). 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.

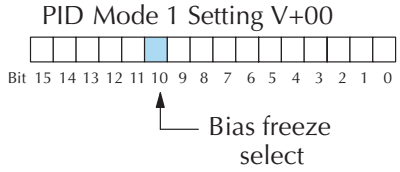
Bias Freeze

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***SOFT32 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 DL06 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 DL06 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 addr+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 addr+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 closed loop tuning procedure. If you want to perform only auto tuning, please skip the next section and proceed directly to the section on auto tuning.

Manual Closed Loop 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***SOFT32.

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 (addr+04). If it is not zero, then write it to zero using ***Direct***SOFT32 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.

8

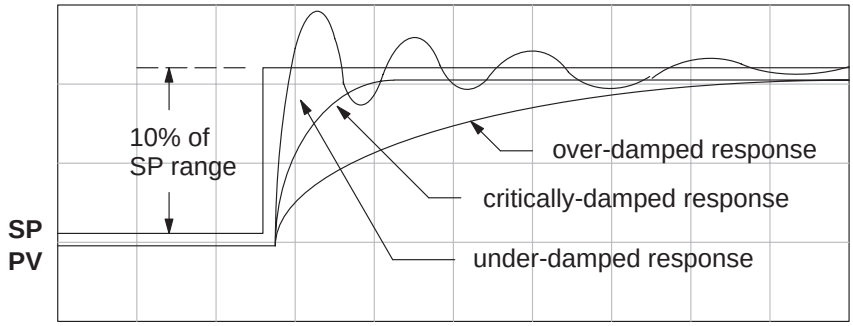
- At this point, the SP should = PV because of the bumpless transfer feature. Increase the SP a little, in order to develop an error value. With only the proportional gain active and the bias term=0, we can easily check the control output value:

  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.

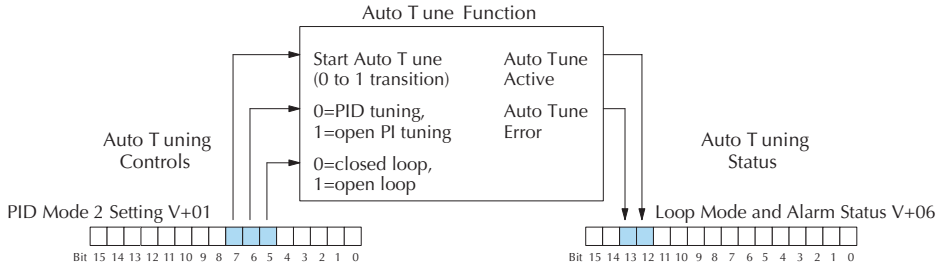
Auto Tuning Procedure

Autotuning is initiated within *Direct*SOFT32. You can use autotuning to establish initial PID parameter values (autotuning is not run continuously during operation). 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.

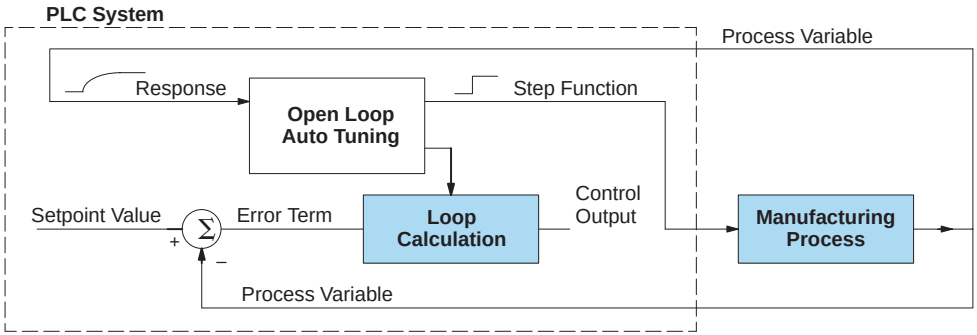
WARNING: Only authorized personnel fully familiar with all aspects of the process should make changes that affect the loop tuning constants. Using the loop auto tuning procedures will affect the process, including inducing large changes in the control output value. Make sure you thoroughly consider the impact of any changes to minimize the risk of injury to personnel or damage to equipment. The auto tune in the DL06 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 addr+01, as shown below. *Direct*SOFT32 will manipulate these bits automatically when you use the auto tune feature within *Direct*SOFT32. 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 allow 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 addr+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.



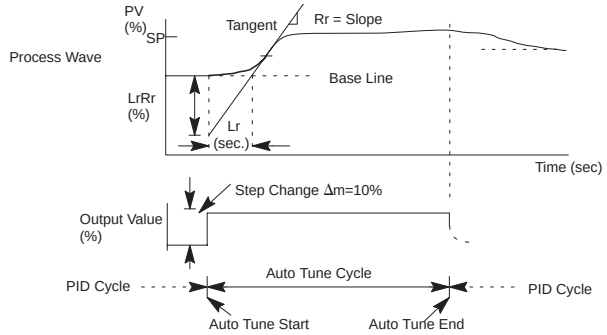
Open-Loop Auto Tuning – During an open-loop auto tuning cycle, the loop controller operates as shown in the diagram below. Before starting this procedure, place the loop in Manual mode and ensure the PV and control output values are in the middle of their ranges (away from the end points).



***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.



* 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%. Open Loop Auto Tune Cycle Wave: Step Response Method

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:

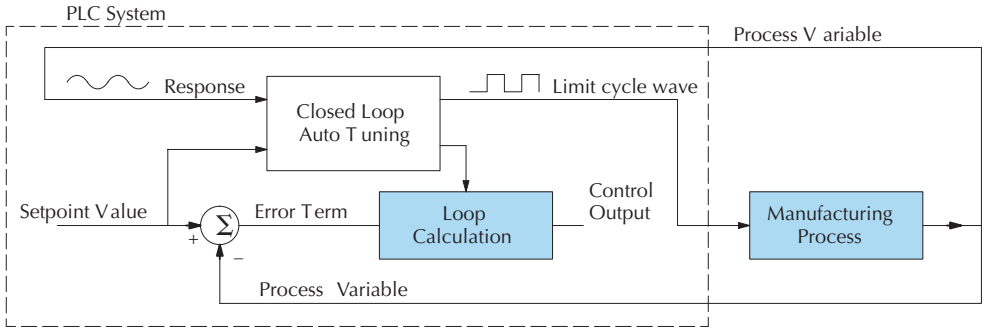
We highly recommend using ***Direct***SOFT32 for the auto tuning interface. The duration of each auto tuning cycle will depend on the mass of our process. A slowly-changing PV will result in a longer auto tune cycle time. When the auto tuning is complete, the proportional,

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)	

integral, and derivative gain values are automatically updated in loop table locations addr+10, addr+11, and addr+12 respectively. The sample time in addr+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).

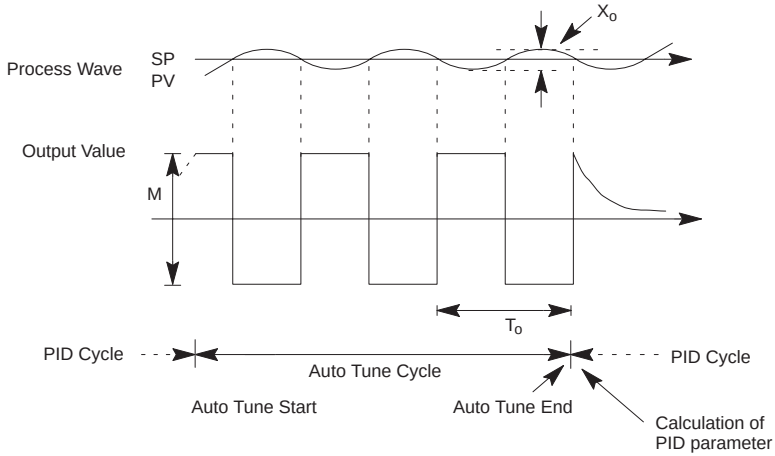
Auto Tuning error: If the auto tune error bit (bit 13 of Loop Mode and Alarm status word addr+06) is on, please verify the PV and SP values are within 5% of full scale difference, as required by the auto tune function. The bit will also turn on if the closed-loop method is in use, and the output goes to the limits of the range.

Closed-Loop Auto Tuning – During a closed-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 examines the direction of the offset of the PV from the SP. The auto tune function then takes control of the control output and induces a full-span step change in the opposite direction. Each time the sign of the error (SP – PV) changes, the output changes full-span in the opposite direction. This proceeds through three full cycles.



*Mmax = Output Value upper limit setting. Mmin = 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 T_o (bump period) and X_o (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:

$Kpc = 4M / (\pi * X_o)$ $\quad$ $Tpc = T_o$	
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

Auto tuning error – if the auto tune error bit (bit 13 of Loop Mode and Alarm status word addr+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.

***NOTE**: If your PV fluctuates rapidly, you probably need to use the built-in analog filter (see page 8–47) or create a filter in ladder logic (see example on page 8–48).*

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.

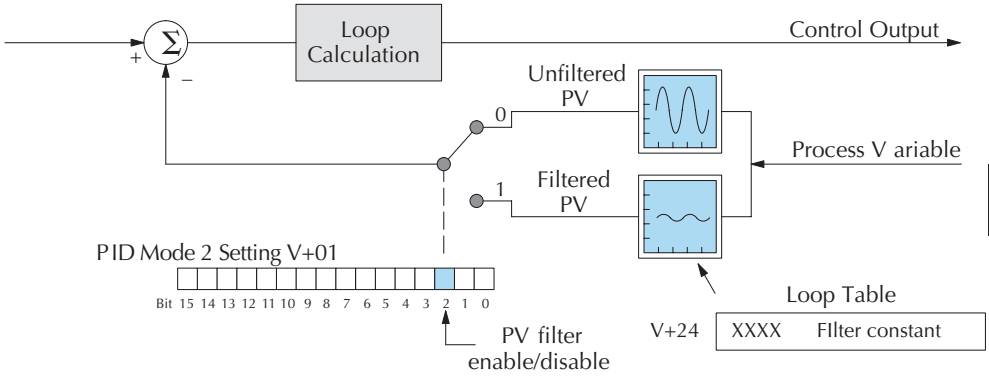
5. Tune the major loop, following the standard loop tuning procedure in this section. The response of the major loop PV is actually the overall response of the cascaded loops together.

PV Analog Filter

A noisy PV signal can make tuning difficult and can cause the control output to be more extreme than necessary, as the output tries to respond to the peaks and valleys of the PV. There are two equivalent methods of filtering the PV input to make the loop more stable. The first method is accomplished using the DL06's built-in filter. The second method achieves a similar result using ladder logic.

The DL06 Built-in Analog Filter

The DL06 provides a selectable first-order low-pass PV input filter which can be particularly helpful during auto tuning, using the closed-loop method. Shown in the figure below, **we strongly recommend the use of a 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 Setting 2 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 addr+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 valid range of 000.1 to 001.0.
- ***Direct***SOFT32 converts values above the valid range to 001.0 and values below this range to 000.1
- 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***SOFT32 for the auto tuning interface. The duration of each auto tuning cycle will depend on the mass of your 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 addr+10, addr+11, and addr+12 respectively. The sample time in addr+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.

The algorithm which the built-in filter follows is:

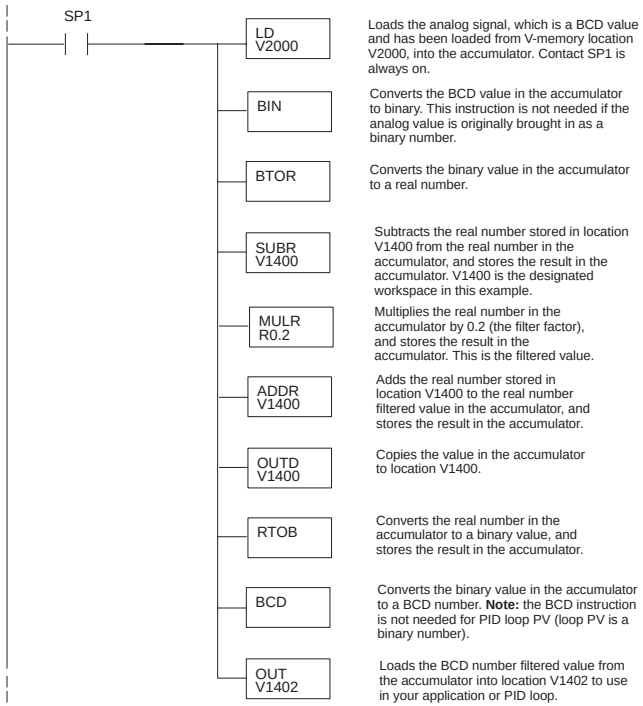
$$yi = k\,(xi - yi{-}1) + yi{-}1$$

yi is the current output of the filter
xi is the current input to the filter
yi–1 is the previous output of the filter
k is the PV Analog Input Filter Factor

Creating an Analog Filter in Ladder Logic

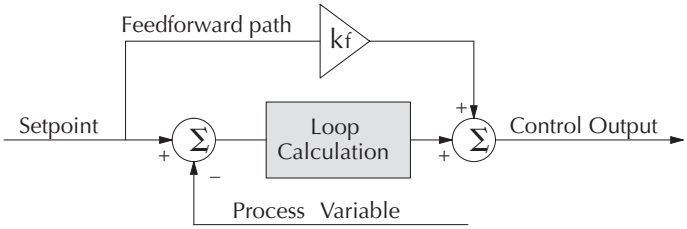
A similar algorithm can be built in your ladder program. Your analog inputs can be filtered effectively using either method. The following programming example describes the ladder logic you will need. Be sure to change the example memory locations to those that fit your application.

Filtering can induce a 1 part in 1000 error in your output because of "rounding." If your process cannot tolerate a 1 part in 1000 error, do not use filtering. Because of the rounding error, you should not use zero or full scale as alarm points. Additionally, the smaller the filter constant the greater the smoothing effect, but the slower the response time. Be sure a slower response is acceptable in controlling your process.



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 DL06. 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.

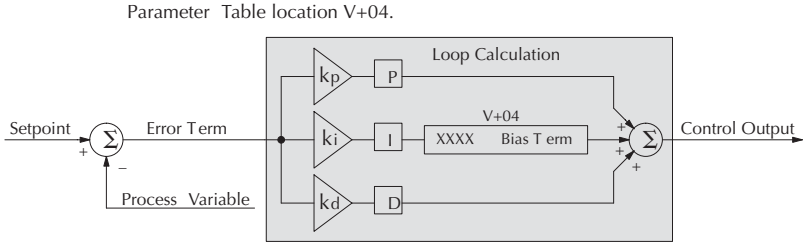


In the previous section on the bias term, we said that "the bias term value establishes a "working region" or operating point for the control output. *When the error fluctuates around its zero point, the output fluctuates around the bias value.*" Now, when there is a change in setpoint, an error is generated 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 DL06 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 addr+04.

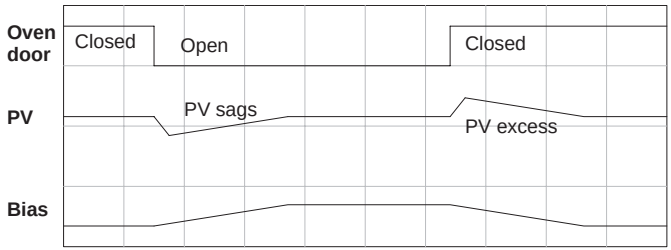


To change the bias (operating point), ladder logic only has to write the desired value to addr+04. The PID loop calculation first reads the bias value from addr+04 and modifies the value based on the current integrator calculation. Then it writes the result back to location addr+04. This arrangement 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).

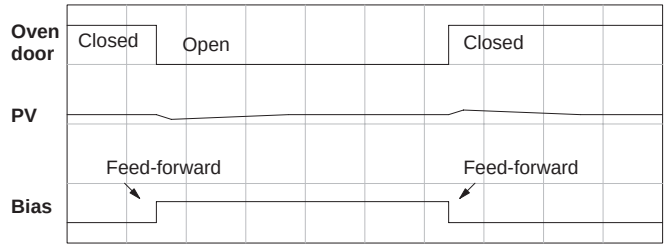
***NOTE**: When writing the bias term, one must be careful to design ladder logic to write the value only once, at the moment when the new bias operating point is to occur. If ladder logic writes the bias value on every scan, the loop's integrator is effectively disabled.*

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 addr+04, adds the desired change amount, and writes it back to addr+04. When the door closes, we duplicate the procedure, but subtracting desired change amount instead. The following figure shows the results.



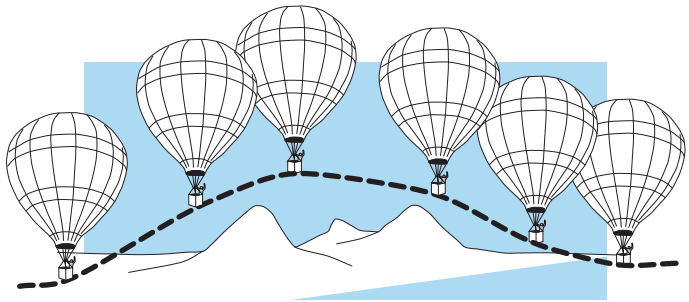
The step changes in the bias are the result of our two feed-forward writes to the bias term. We can see the PV variations are greatly reduced. The same technique may be applied for changes in setpoint.

Time-Proportioning Control

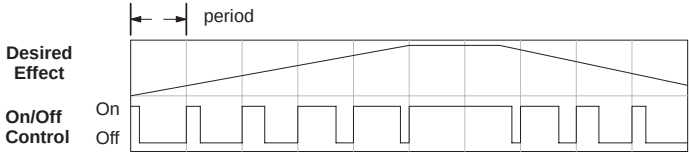
The PID loop controller in the DL06 CPU generates a smooth control output signal across a numerical range. The control output value is suitable to drive an analog output module, which connects to the process. In the process control field, this is called *continuous control*, because the output is on (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*.



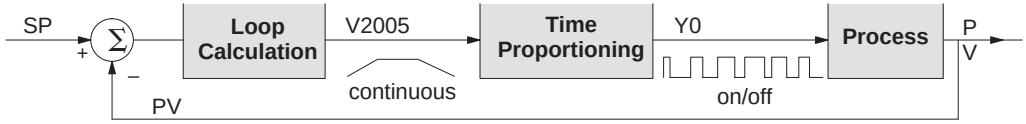
Time-proportioning control approximates continuous control by virtue of its duty-cycle – the ratio of ON time to OFF time. The following figure shows an example of how duty cycle approximates a continuous level when it is averaged by a large process mass.



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 segment provides a time proportioned on/off control output. It converts the continuous output in V2005 to on/off control using the output coil, Y0.



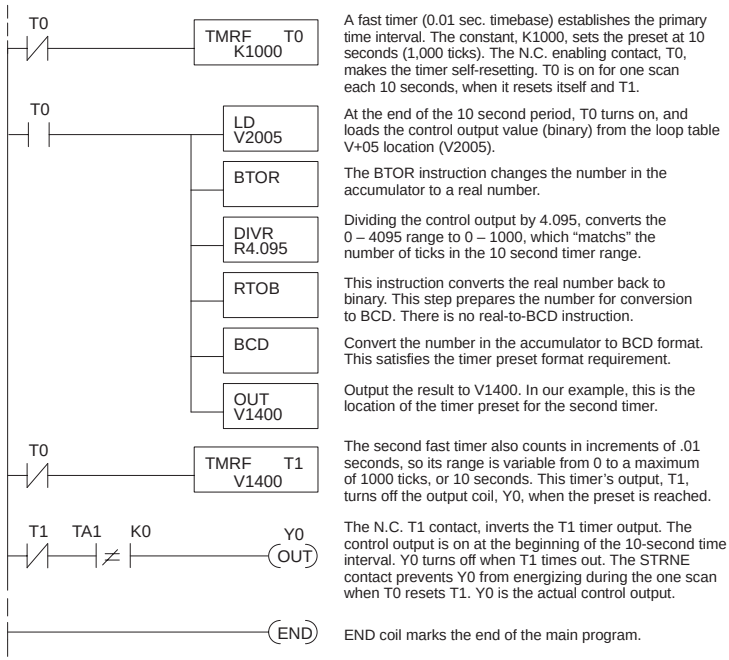
The example program uses two timers to generate On/Off control. It makes the following **assumptions**, which you can alter to fit your application:

- 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) or 0-4,095).
- The On/Off control output is Y0.

The time proportioning program must match the resolution of the output (1 part in 1000) to the resolution of the time base of T0 (also 1 part in 1000).

8

***NOTE**: Some processes change too fast for time proportioning control. Consider the speed of your process when you choose this control method. Use continuous control for processes that change too fast for time proportioning control.*



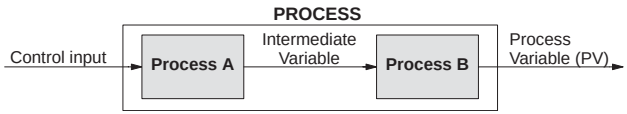
Cascade Control

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 DL06 also provides Cascaded Mode.

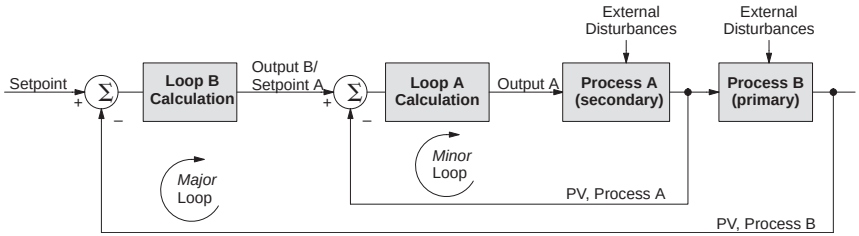
***NOTE**: Cascaded loops are an advanced process control technique. Therefore we recommend their use only for experienced process control engineers.*

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.

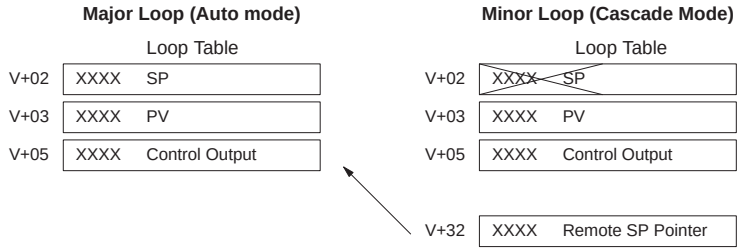
Cascaded Loops in the DL06 CPU

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.*

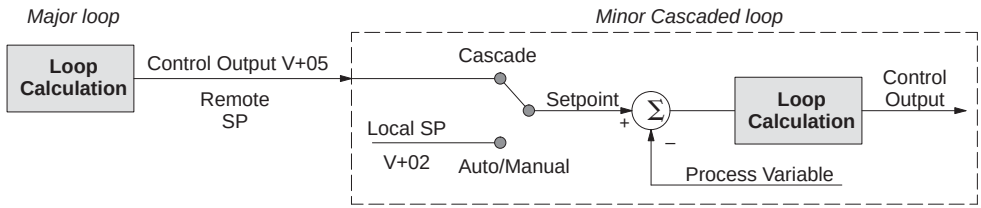
You can cascade together as many loops as necessary on the DL06, and you may have multiple groups of cascaded loops. For proper operation on cascaded loops you must use the same data range (12/15 bit) and unipolar/bipolar settings on the major and minor loop.

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 addr+32, as shown below. The pointer must be the address of the addr+05 location (control output) of the major loop. In Cascade Mode, the minor loop will ignore the its local SP register (addr+02), and read the major loop's control output as its SP instead.



When using *Direct*SOFT32's PID View to watch the SP value of the minor loop, *Direct*SOFT32 automatically reads the major loop's control output and displays it for the minor loop's SP. The minor loop's normal SP location, addr+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.

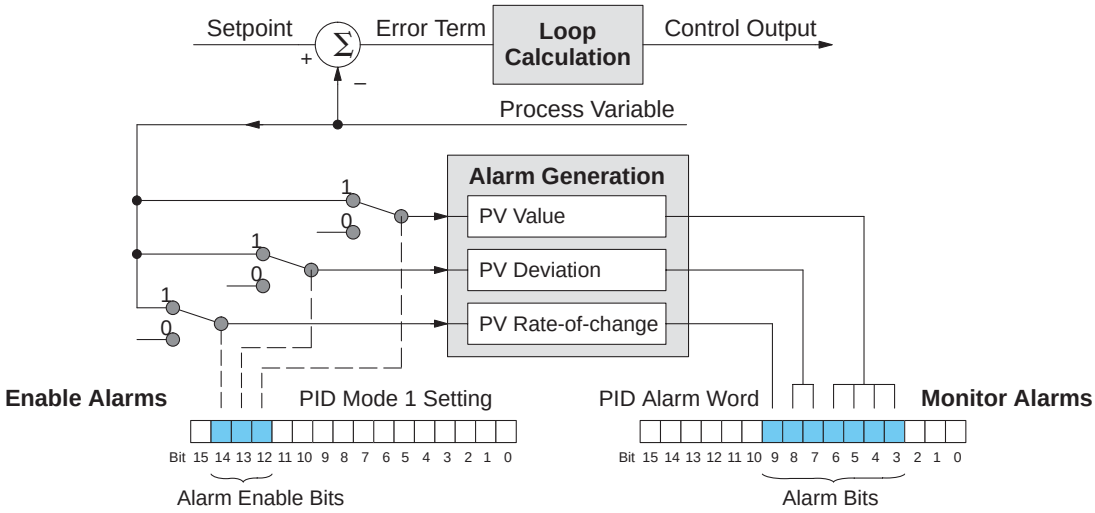
Process Alarms

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 DL06 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 addr+00 word in the loop parameter table to enable/disable the alarms. ***Direct***SOFT32'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 addr+06 in the

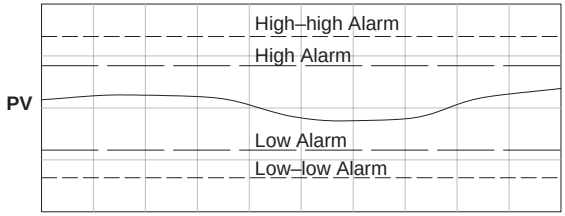


loop table.

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

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.

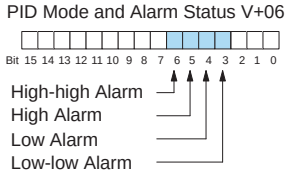


Loop Table

V+16	XXXX	High-high Alarm
V+15	XXXX	High Alarm
V+14	XXXX	Low Alarm
V+13	XXXX	Low-low Alarm

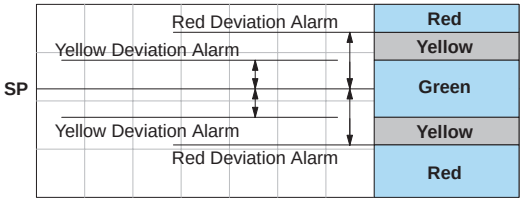
If the process remains out of control for some time, the PV will eventually cross one of the outer alarm thresholds, named High-high alarm and Low-low alarm. Their threshold values are programmed using the loop table registers listed above. A High-high or Low-low alarm indicates a serious condition exists, 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*SOFT32.



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 addr+17 and addr+20 as shown.

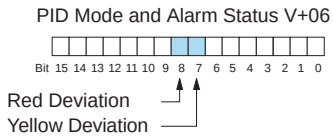


Loop Table

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*SOFT32.



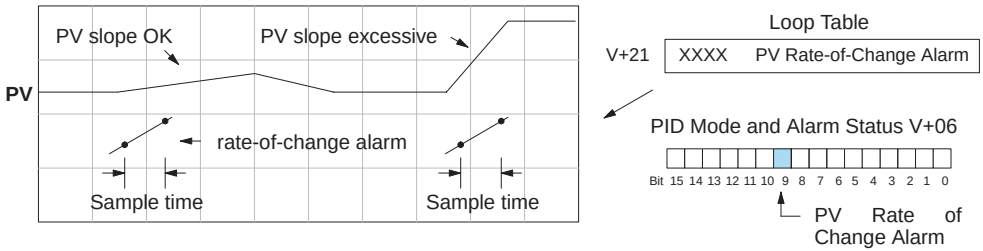
The PV Deviation Alarm can be independently enabled and disabled from the other PV alarms, using bit 13 of the PID Mode 1 Setting addr+00 word.

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 DL06 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 addr+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 addr+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:

$$\text{Alarm Rate-of-Change} = \frac{15 \text{ degrees}}{1 \text{ minute}} \times \frac{10 \text{ counts / degree}}{30 \text{ loop samples / min.}} = \frac{150}{30} = 5 \text{ counts / sample period}$$

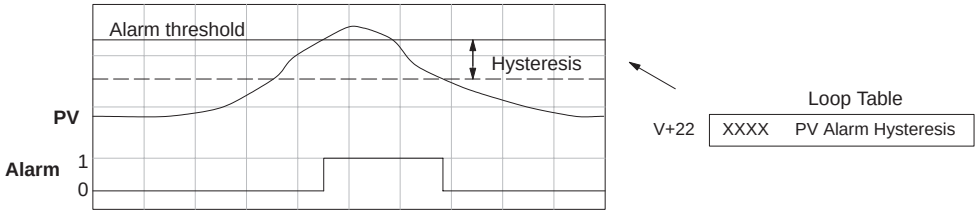
From the calculation result, we would program the value 5 in the loop table for the rate-of-change. The PV Rate-of-Change Alarm can be independently enabled and disabled from the other PV alarms, using bit 14 of the PID Mode 1 Setting addr+00 word.

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.

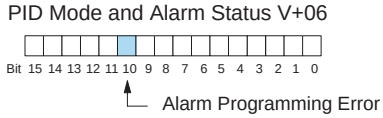
The PV Alarm Hysteresis amount is programmable from 1 to 200 (hex). When using the PV Deviation Alarm, the programmed hysteresis amount must be less than the programmed deviation amount. The figure below shows how the hysteresis is applied when the PV value goes past a threshold and descends back through 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 Programming 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.

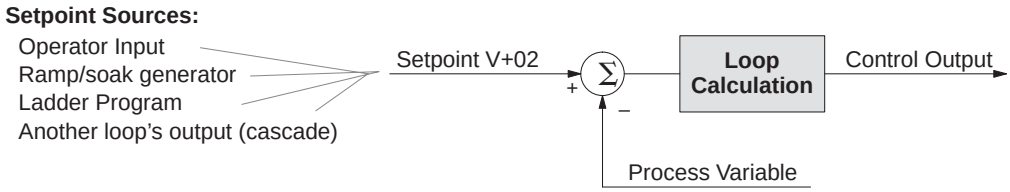


- PV Absolute Alarm value requirements:
  Low-low < Low < High < High-high
- PV Deviation Alarm requirements:
  Yellow < Red

Ramp/Soak Generator

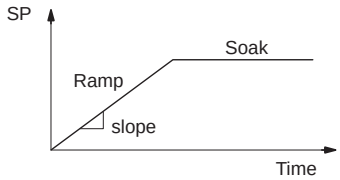
Introduction

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 addr+02 at any particular time.



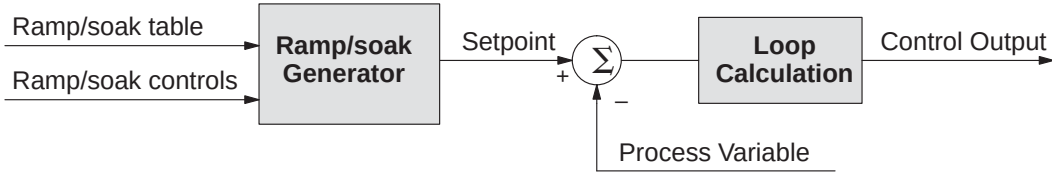
If the SP for your process rarely changes or can tolerate step changes, you probably will not need to use the ramp/soak generator. However, some processes require precisely-controlled SP value changes. *The ramp / soak generator can greatly reduce the amount of programming required for these applications.*

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.

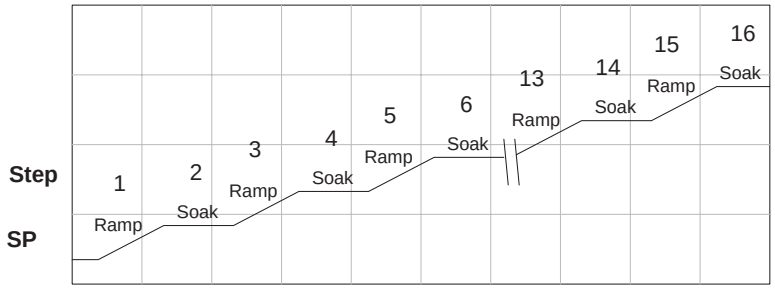
It is instructive to view the ramp/soak generator as a dedicated function to generate SP values, as shown below. It has two categories of inputs which determine the SP values generated. The ramp/soak table must be programmed in advance, containing the values that will define the ramp/soak profile. The 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.

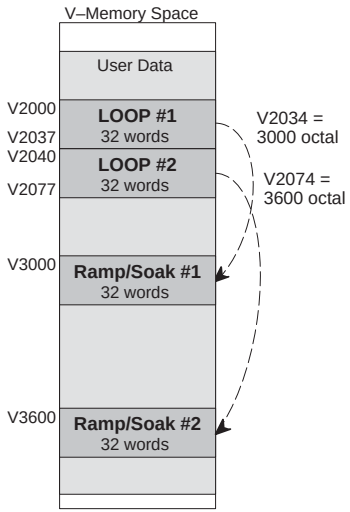
The following figure shows a SP profile consisting of ramp/soak segment pairs. The segments are individually numbered as steps from 1 to 16. The slope of each of the ramp may be either increasing or decreasing. The ramp/soak generator automatically knows whether to increase or decrease the SP based on the relative values of a ramp's end points. These values come from the ramp/soak table.



Ramp/Soak Table

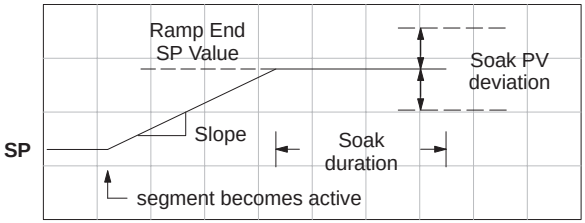
The parameters which define the ramp/soak profile for a loop are in a ramp/soak table. Each loop may have its own ramp/soak table, but it is optional. Recall the Loop Parameter table consists a 32-word block of memory for each loop, and together they occupy one contiguous memory area. However, the ramp/soak table for a loop is individually located, because it is optional for each loop. An address pointer in location addr+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*SOFT32, 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.

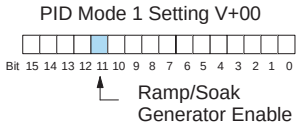
Ramp/Soak Table Flags

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	Off	On
8–15	Current Step in R/S Profile	read	decode as byte (hex)	

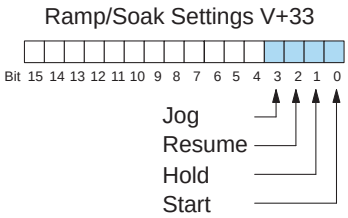
Ramp/Soak Generator Enable

The main enable control to permit ramp/soak generation of the SP value is accomplished with bit 11 in the PID Mode 1 Setting addr+00 word, as shown to the right. The other ramp/soak controls in addr+33 shown in the table above will not operate unless this bit=1 during the entire ramp/soak process.



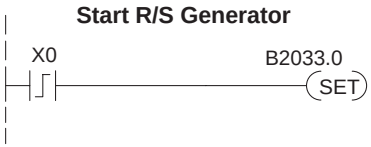
Ramp/Soak Controls

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*SOFT32 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.



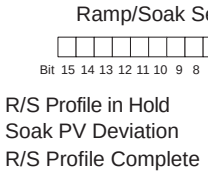
The normal state for the ramp/soak control bits is all zeros. Ladder logic must set only one control bit at a time.

- **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.

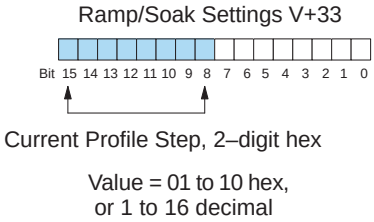
Ramp/Soak Profile Monitoring

You can monitor the Ramp/Soak profile status using other bits in the Ramp/Soak Settings addr+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. Ramp/Soak Settings addr+33

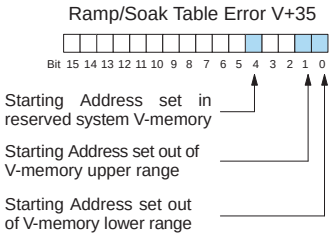


The number of the current step is available in the upper 8 bits of the Ramp/Soak Settings addr+33 word. The bits represent a 2-digit hex number, ranging from 1 to 10. Ladder logic can monitor these to synchronize other parts of the program with the ramp/soak profile. Load this word to the accumulator 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*SOFT32 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*SOFT32'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.
- 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 addr+31 is zero.
- 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. Remember, gains are entered in the loop table in BCD, while the SP and PV are in binary. If you are using DirectSOFT32, it displays the SP, PV, Bias and Control output in decimal (BCD), converting it to 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 addr+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 addr+27 is too low.
- Check your ladder program to verify it is not writing to the SP location (addr+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 addr+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 (addr+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 DirectSOFT32 work okay, but these values do not work properly when the ladder program writes the data.

A. The PID View in DirectSOFT32 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 DirectSOFT32 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 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, 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, Third 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 Author: Gregory K. McMillan Publisher: Instrument Society of America ISBN 1–55617–483-7
Instrument Engineer's Handbook, Volume 2: Process Control, Third Edition Author (Editor-in-Chief): Bela G. Liptak Publisher: Chilton. . . . . ISBN 0–8019–8242–1	Instrument Engineer's Handbook, Volume 1: Process Measurement, Third Edition 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 DL06'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.

Maintenance and Troubleshooting

In This Chapter...

Hardware System Maintenance 9–2
Diagnostics 9–2
CPU Indicators 9–6
Communications Problems 9–7
I/O Point Troubleshooting 9–8
Noise Troubleshooting 9–10
Machine Startup and Program Troubleshooting 9–11

Hardware System Maintenance

Standard Maintenance

No regular or preventative maintenance is required for this product (there are no internal batteries); however, a routine maintenance check (about every one or two months) of your PLC and control system is good practice, and should include the following items:

- Air Temperature – Monitor the air temperature in the control cabinet, so the operating temperature range of any component is not exceeded.
- Air Filter – If the control cabinet has an air filter, clean or replace it periodically as required.
- Fuses or breakers – Verify that all fuses and breakers are intact.
- Cleaning the Unit – Check that all air vents are clear. If the exterior case needs cleaning, disconnect the input power, and carefully wipe the case using a damp cloth. Do not let water enter the case through the air vents and do not use strong detergents because this may discolor the case.

Diagnostics

Diagnostics

Your DL06 Micro PLC performs many pre-defined diagnostic routines with every CPU scan. The diagnostics can detect various errors or failures in the PLC. The two primary error classes are *fatal and non-fatal.*

Fatal Errors

Fatal errors are errors which may cause the system to function improperly, perhaps introducing a safety problem. The CPU will automatically switch to Program Mode if it is in Run 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 allow you to transition to Run Mode until the error has been corrected.

Some examples of fatal errors are:

- Power supply failure
- Parity error or CPU malfunction
- Particular programming errors

Non-fatal Errors

Non-fatal errors are errors that need your attention, but should not cause improper operation. They do not cause or prevent any mode transitions of the CPU. The application program can use special relay contacts to detect non-fatal errors, and even take the system to an orderly shutdown or switch the CPU to Program Mode if desired. An example of a non-fatal error is:

- Particular programming errors - The programming devices will notify you of an error if one occurs while online.
- **Direct**SOFT provides the error number and an error message.
- The handheld programmer displays error numbers and short descriptions of the error.

Appendix B has a complete list of error messages in order by error number. Many error messages point to supplemental V-memory locations which contain related information. Special relays (SP contacts) also provide error indications (refer to Appendix D).

V-memory Error Code Locations

The following table names the specific memory locations that correspond to certain types of error messages.

Error Class	Error Category	Diagnostic V-memory
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

The special relay table also includes status indicators which can indicate errors. For a more detailed description of each of these special relays refer to Appendix D.

CPU Status Relays	
SP11	Forced Run mode
SP12	Terminal Run mode
SP13	Test Run mode
SP15	Test stop mode
SP16	Terminal Program mode
SP17	Forced stop
SP20	STOP instruction was executed
SP22	Interrupt enabled
System Monitoring Relays	
SP36	Override setup
SP37	Scan control error
SP40	Critical error
SP41	Non-critical error
SP42	Diagnostics error
SP44	Program memory error
SP45	I/O error
SP46	Communications error
SP50	Fault instruction was executed
SP51	Watchdog timeout
SP52	Syntax error
SP53	Cannot solve the logic
SP54	Communication error
SP56	Table instruction overrun
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

DL06 Micro PLC Error Codes

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 in the CPU)
E104	Write failed
E151	Invalid command
E311	Communications error 1
E312	Communications error 2
E313	Communications error 3
E316	Communications error 6
E320	Time out
E321	Communications error
E360	HP Peripheral port time-out
E501	Bad entry
E502	Bad address
E503	Bad command
E504	Bad reference / value
E505	Invalid instruction
E506	Invalid operation
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

Error Code	Description
E525	Mode Switch not in Term position
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
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	Mis-compare
E650	Handheld Programmer system error
E651	Handheld Programmer ROM error
E652	Handheld Programmer RAM error

Program Error Codes

The following table lists program syntax and runtime error codes. Error detection occurs during a Program-to-Run mode transition, 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
E421	Duplicate stage reference
E422	Duplicate SBR/LBL reference
E423	Nested loop
E431	Invalid ISG/SG address
E433	Invalid ISG / SG address
E434	Invalid RTC
E435	Invalid RT
E436	Invalid INT address
E437	Invalid IRTC

Error Code	Description
E438	Invalid IRT address
E440	Invalid Data Address
E441	ACON/NCON
E451	Bad MLS/MLR
E453	Missing T/C
E454	Bad TMRA
E455	Bad CNT
E456	Bad SR
E461	Stack Overflow
E462	Stack Underflow
E463	Logic Error
E464	Missing Circuit
E471	Duplicate coil reference
E472	Duplicate TMR reference
E473	Duplicate CNT reference
E499	Print instruction

CPU Indicators

The DL06 Micro PLCs have indicators on the front to help you determine potential problems with the system. In normal runtime operation only, the RUN and PWR indicators are on. The table below is a quick reference to potential problems.

Indicator Status	Potential Problems
PWR (Green LED off)	System voltage incorrect
	PLC power supply faulty
RUN (Green LED off)	CPU programming error
	(CPU in program mode)
CPU (Red LED on)	Electrical noise interference
	Internal CPU defective
CPU (Blinking Red LED)	Low backup battery (refer to page 4-8)

PWR Indicator

In general there are three reasons for the CPU power status LED (PWR) to be OFF:

1. Power to the unit is incorrect or is not applied.
2. PLC power supply is faulty.
3. Other component(s) have the power supply shut down.

If the voltage to the power supply is not correct, the PLC 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 external power.
2. Verify that all external circuit breakers or fuses are still intact.
3. Check all incoming wiring for loose connections. If you're using a separate termination block, check those connections for accuracy and integrity.
4. If the connections are acceptable, reconnect the system power and verify the voltage at the DL06 power input is within specification. If the voltage is not correct, shut down the system and correct the problem.
5. If all wiring is connected correctly and the incoming power is within the specifications, the PLC internal supply may be faulty.

The best way to check for a faulty PLC is to substitute a known good one to see if this corrects the problem. The removable connectors on the DL06 make this relatively easy. If there has been a major power surge, it is possible the PLC internal power supply has been damaged. If you suspect this is the cause of the power supply damage, consider installing an AC line conditioner to attenuate damaging voltage spikes in the future.

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.)

Both of the programming devices, Handheld Programmer and *Direct*SOFT32, will return an 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.

If the CPU indicator is blinking, the backup battery is low (refer to page 4-8).

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).
- The device connected to the port is sending data incorrectly, or another application is running on the device.
- A grounding difference exists between the two devices.
- Electrical noise is causing intermittent errors.
- The PLC has a bad communication port and should be replaced.

For problems in communicating with *Direct*SOFT32 on a personal computer, refer to the *Direct*SOFT32 manual. It includes a troubleshooting section that can help you diagnose PC problems in communications port setup, address or interrupt conflicts, etc.

I/O Point Troubleshooting

Possible Causes

If you suspect an I/O error, there are several things that could be causing the problem.

- High-Speed I/O configuration error
- A blown fuse in your machine or panel (the DL06 does not have internal I/O fuses)
- A loose terminal block
- The auxiliary 24 VDC supply has failed
- The Input or Output Circuit has failed

Some Quick Steps

When troubleshooting the DL06 Micro PLCs, please be aware of the following facts which may assist you in quickly correcting an I/O problem.

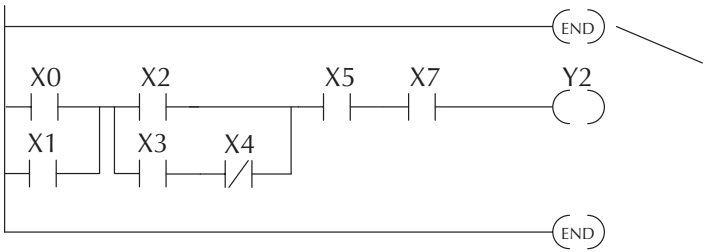
- HSIO configuration errors are commonly mistaken for I/O point failure during program development. If the I/O point in question is in X0–X2, or Y0–Y1, check all parameter locations listed in Chapter 3 that apply to the HSIO mode you have selected.
- The output circuits cannot detect shorted or open output points. If you suspect one or more faulty points, 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 point.
- The I/O point status indicators are logic-side indicators. This means the LED which indicates the on or off status reflects the status of the point with respect to the CPU. On an output point the status indicators could be operating normally while the actual output device (transistor, triac etc.) could be damaged. With an input point, if the indicator LED is on, the input circuitry is probably operating properly. Verify the LED goes off when the input signal is removed.
- Leakage current can be a problem when connecting field devices to an I/O point. 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 20K resistor will work. Verify the wattage rating of the resistor is correct for your application.
- Because of the removable terminal blocks on the DL06, the easiest method to determine if an I/O circuit has failed is to replace the unit if you have a spare. However, if you suspect a field device is defective, that device may cause the same failure in the replacement PLC as well. As a point of caution, you may want to check devices or power supplies connected to the failed I/O circuit before replacing the unit with a spare.

Output points can be set on or off in the DL06 series CPUs. If you want to do an I/O check-out independent of the application program, follow the procedure below:

Step	Action
1	Use a handheld programmer or ***Direct***SOFT32 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        Y0
□□□□□□□□□□□□□□□□
```

Use arrow keys to select point, then use ON and OFF to change the status

← ← SHFT ON INS

Y2 is now on

```
        Y 10        Y0
□□□□□□□□□□□□□■□□
```

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 they 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 an attached wire, panel connection ,etc. It may enter through an I/O circuit, 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. Ensure 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 can act as a large antenna, introducing noise into the system. Therefore, 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 PLC and I/O circuits. Installing an isolation transformer for all AC sources can correct this problem. DC sources should be well-grounded good quality supplies.
- Separate input wiring from output wiring. Never run low-voltage I/O wiring close to high voltage wiring.

Machine Startup and Program Troubleshooting

The DL06 Micro PLCs provide several features that can 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
- Special Instructions
- Run Time Edits
- Forcing I/O Points

Syntax Check

Even though the Handheld Programmer and *Direct*SOFT32 provide error checking during program entry, you may want to check a program that has been modified. 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*SOFT32. 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
?
```

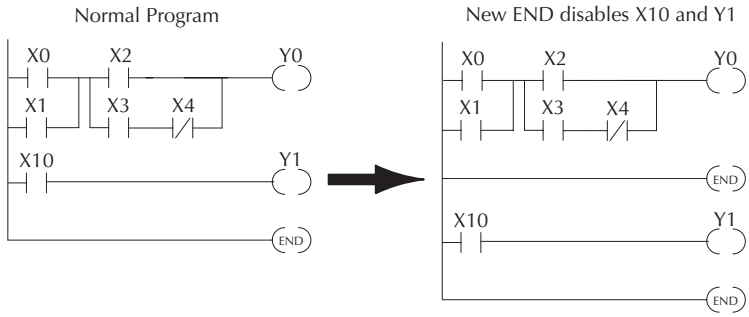
See the Error Codes Section for a complete listing of programming error codes. If you get an error, just press CLR and the Handheld will display the instruction where the error occurred. Correct the problem and continue running the Syntax check until the NO SYNTAX ERROR message appears.

Special Instructions

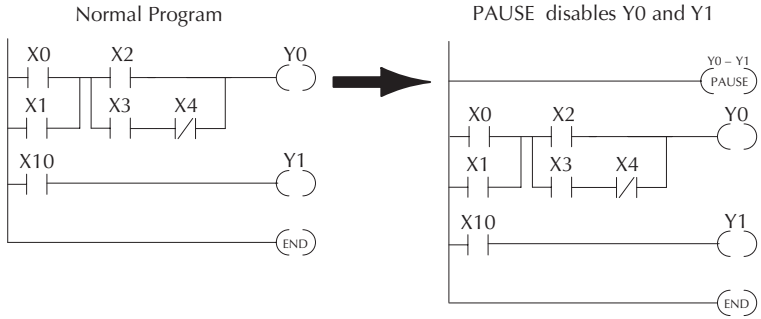
There are several instructions that can be used to help you debug your program during machine startup operations.

- END
- PAUSE
- STOP

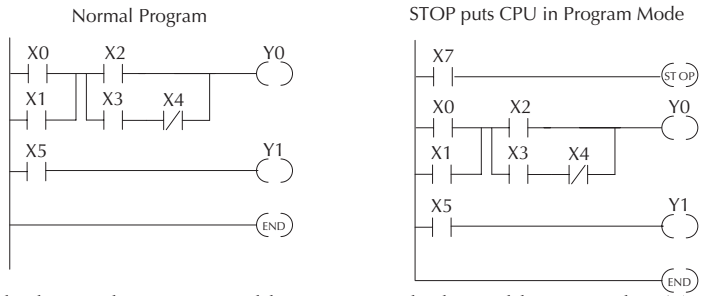
END Instruction: If you need a way to quickly disable part of the program, just insert an END statement prior to the portion that should be disabled. When the CPU encounters the END statement, it assumes that is the end of the program. The following diagram shows an example.



PAUSE Instruction: This instruction provides a quick way to allow the inputs (or other logic) to operate while disabling selected outputs. The output image register is still updated, but the output circuits are not. For example, you could make this conditional by adding an input contact or CR to control the instruction with a switch or a programming device. Or, you could just add the instruction without any conditions so the selected outputs would be disabled at all times.



STOP Instruction: Sometimes during machine startup you need a way to quickly turn off all the outputs and return to Program Mode. You can 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 X7 which would execute the STOP instruction. The CPU would enter Program Mode and all outputs would be turned off.

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***SOFT32. 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)
(shows location in question)

```
$00024 E471
DUP COIL REF
```

Syntax OK display

```
NO DUP REFS
?
```

If you get an error, just press CLR and the Handheld will display the instruction where the error 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 containing Stage instructions and / or OROUT instructions. The Duplicate Reference check will find occurrences, even though they are acceptable.*

Run Time Edits

The DL06 Micro PLC 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 operational 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.

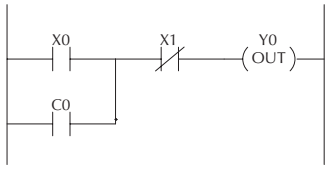
Not all instructions can be edited during a Run Time Edit session. The following list shows the instructions that can be edited.

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 (Boolean)
AND, ANDN	And, And not (Boolean)
OR, ORN	Or, Or not (Boolean)
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 (Comparative Boolean)
AND, ANDN	And greater than or equal And less than (Comparative Boolean)

Mnemonic	Description
OR, ORN	Or greater than or equal or less than (Comparative Boolean)
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

Run Time Edit Example

We'll use the program logic shown to describe how this process works. In the example, we'll 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 D2–HPP 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, that 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 DL06 CPUs process the forcing requests.

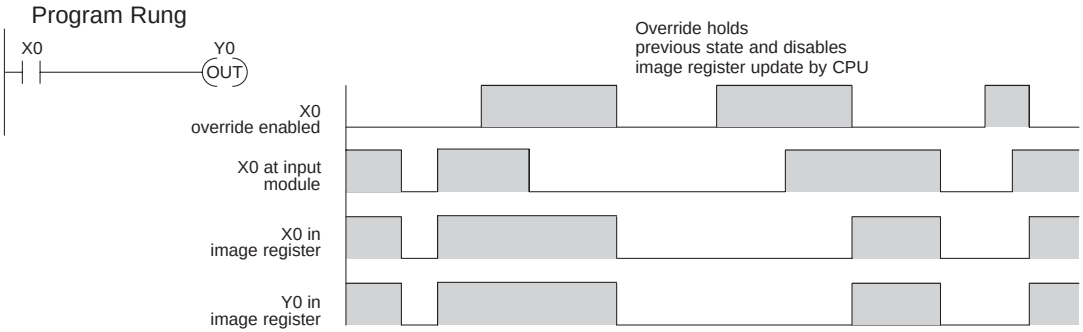
WARNING: Only authorized personnel fully familiar with all aspects of the application should make changes to the program. Make sure you thoroughly consider the impact of any changes to minimize the risk of personal injury or damage to equipment.

There are two types of forcing available with the DL06 CPUs. (Chapter 3 provides a detailed description of how the CPU processes each type of forcing request).

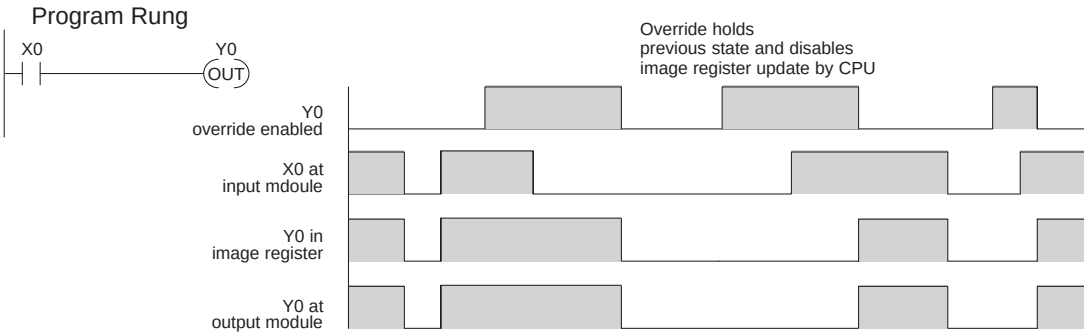
- Regular Forcing: This type of forcing can temporarily change the status of a discrete bit. For example, you may want to force an input on, even though it is really off. This allows you to change the point status that was stored in the image register. This value will be valid until the image register location is written to during the next scan. This is primarily useful during testing situations when you need to force a bit on to trigger another event.
- Bit Override : Bit override can be enabled on a point-by-point basis by using AUX 59 from the Handheld Programmer or by a menu option in DirectSOFT. You can use Bit Override with X, Y, C, T, CT, and S data types. Bit override basically disables any changes to the discrete point by the CPU. For example, if you enable bit override for X1, and X1 is off at the time, the CPU will not change the state of X1. This means that even if X1 comes on, the CPU will not acknowledge the change. Therefore, if you used X1 in the program, it would always be evaluated as "off" in this case. If X1 was on when the bit override was enabled, then X1 would always be evaluated as "on".

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

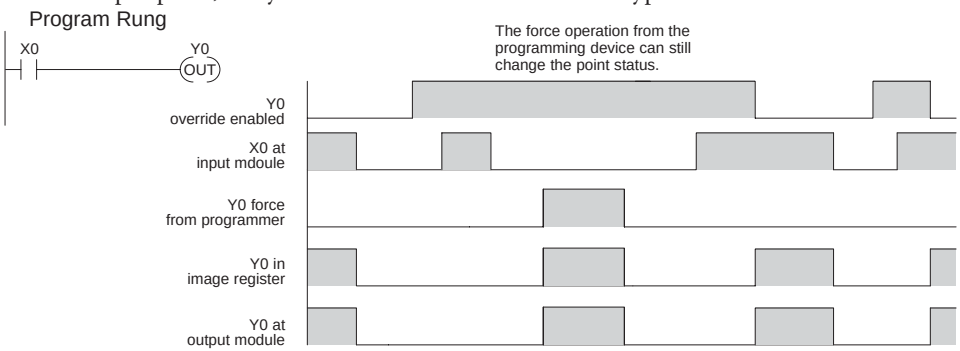
The following diagrams show how the bit override works for both input and output points. The example uses a simple rung, but the concepts are similar for any type of bit memory.



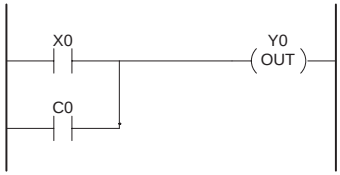
The following diagram shows how the bit override works for an output point. Notice the bit override maintains the output in the current state. If the output is on when the bit override is enabled, then the output stays on. If it is off, then the output stays off.



The following diagram shows how you can use a programming device in combination with the bit override to change the status of the point. Remember, bit override only disables CPU changes. You can still use a programming device to force the status of the point. Plus, since bit override maintains the current status, this enables true forcing. The example shown is for an output point, but you can also use the other bit data types.



The following diagrams show a brief example of how you could use the DL06 Handheld Programmer to force an I/O point. Remember, if you are using the Bit Override feature, the CPU will retain the forced value until you disable the Bit Override or until you remove the force. 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


```
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.)


```
   Y  10          Y   0
□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
```

Use arrow keys to select point, then use ON and OFF to change the status


```
   Y  10          Y   0
□ □ □ □ □ □ □ □ □ □ □ □ ■ □ □ □
```

Y2 is now on

Regular Forcing with Direct Access

From a clear display, use the following keystrokes to force Y10 ON Solid fill indicates point is on.

Solid fill indicates point is on.

```
BIT FORCE
Y10                ■
```

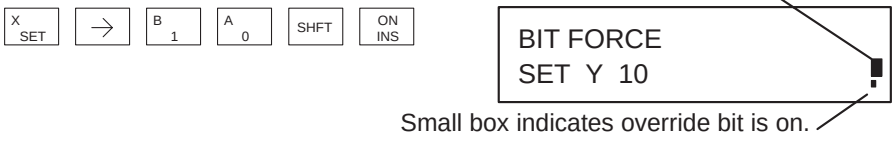
From a clear display, use the following keystrokes to force Y10 OFF No fill indicates point is off.

No fill indicates point is off.

```
BIT FORCE
Y10                □
```

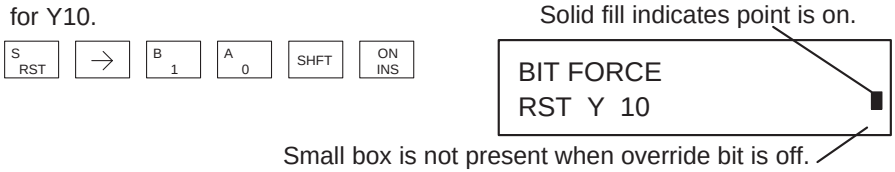
Bit Override Forcing

From a clear display, use the following keystrokes to turn on the override bit for Y10.



Note, at this point you can use the PREV and NEXT keys to move to adjacent memory locations and use the SHFT ON keys to set the override bit on.

From a clear display, use the following keystrokes to turn off the override bit for Y10. Solid fill indicates point is on.

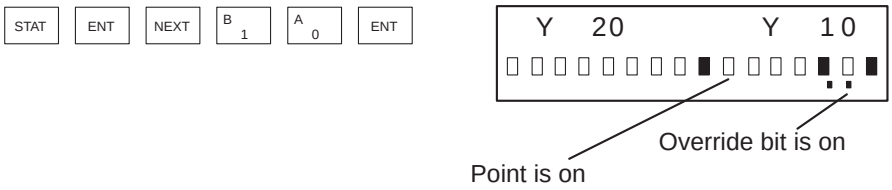


Like the example above, you can use the PREV and NEXT keys to move to adjacent memory locations and use the SHFT OFF keys to set the override bit off.

Bit Override Indicators

Override bit indicators are also shown on the handheld programmer status display. Below are the keystrokes to call the status display for Y10 – Y20.

From a clear display, use the following keystrokes to display the status of Y10 – Y20.



LCD Display Panel

In This Chapter...

Introduction to the DL06 LCD Display Panel 10–2
Keypad 10–2
Snap-in installation 10–3
Display Priority 10–4
Menu Navigation 10–5
Confirm PLC Type, Firmware Revision Level, Memory Usage, Etc. .10–6
Examining Option Slot Contents 10–8
Monitoring and Changing Data Values 10–10
Bit Monitor 10–13
Changing Date and Time 10–14
Setting Password and Locking 10–17
Reviewing Error History 10–20
Toggle Light and Beeper, Test Keys 10–21
PLC Memory Information for the Display 10–22
Changing the Default Screen 10–25
DL06 LCD Display Panel Instruction (LCD) 10–26

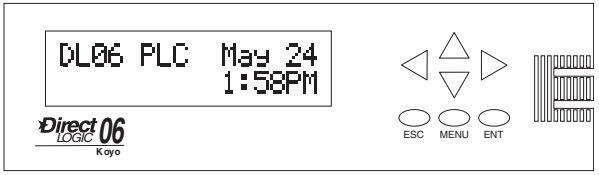
Introduction to the DL06 LCD Display Panel

The DL06 LCD Display Panel is a 16 character, two row display that mounts directly on the face of the DL06 PLC. The LCD is backlit for easy readability in most lighting situations. There are multiple ways of interacting with the LCD Display Panel:

- Built-in keypad
- LCD ladder instruction
- Using ladder instructions to write bit status changes to specified memory locations

The seven function keys on the face of the LCD Display Panel give the user access to clock and calendar setup, V-memory data values or I/O status, etc. Individuals with password authorization can:

- Change clock or calender settings or formats
- Monitor or change V-memory values (including DWord values)
- Force individual bits on or off (up to 16 per screen)
- Review error code history
- Set or change the password
- Turn the back light or buzzer on or off

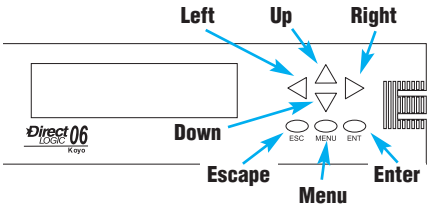


The potential uses for the DL06's LCD display vary widely. An operator can change values for setting up batch processes or machine timing for manufacturing different products. Maintenance personnel can interface in the control cabinet to identify machine problems. LCD messages can be preprogrammed for process events or alarms. The LCD can satisfy these and many other operator interface needs.

10

Keypad

The LCD Display Panel keypad has seven keys you can use to navigate through the menu hierarchy. Each screen displayed has a specific set of active keys associated with it. All other keys (those not associated with the current screen) are inactive.

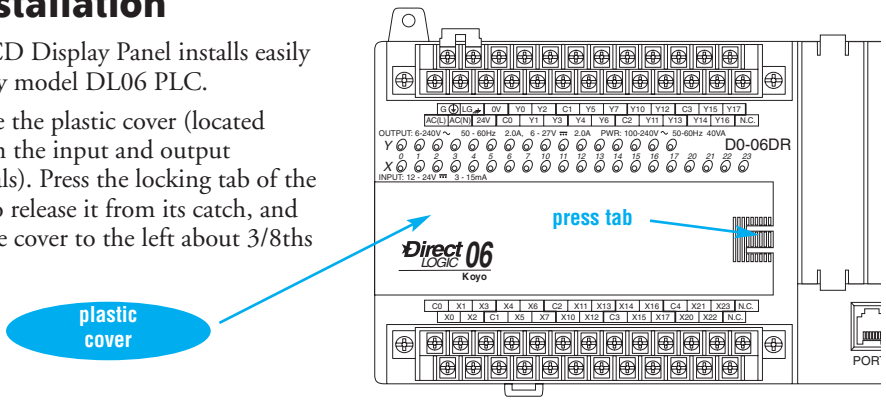


Function Keys		
Name	**Label**	**Function**
Up arrow	none	Move to selection above or increase value
Down arrow	none	Move to selection below or decrease value
Left arrow	none	Move to next digit to the left
Right arrow	none	Move to next digit to the right
Escape	ESC	Return to previous screen or next level up in the menu hierarchy
Menu	MENU	Scroll down through main menu or sub-menu selections
Enter	ENT	Enter the domain of the menu screen selected or save new value

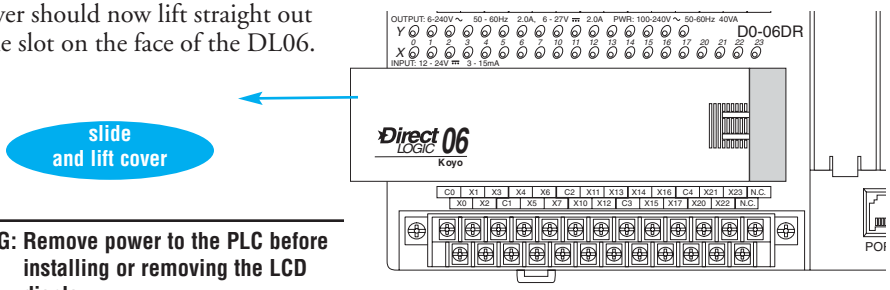
Snap-in installation

The LCD Display Panel installs easily into any model DL06 PLC.

Remove the plastic cover (located between the input and output terminals). Press the locking tab of the cover to release it from its catch, and slide the cover to the left about 3/8ths inch.

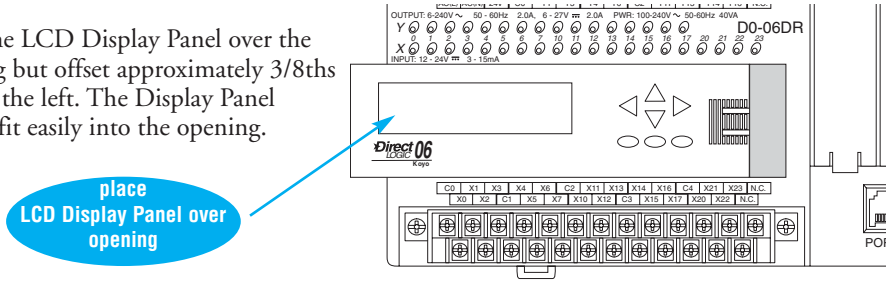


The cover should now lift straight out from the slot on the face of the DL06.

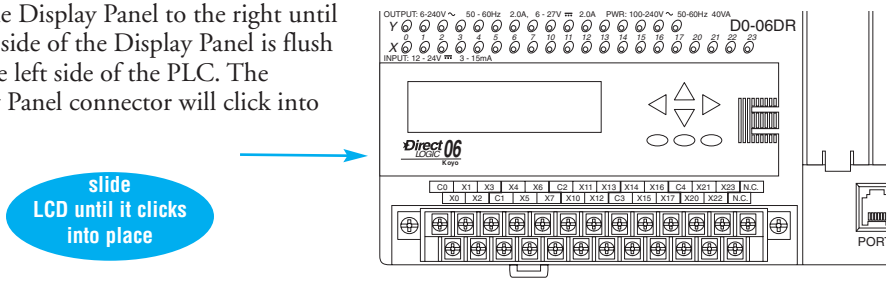


WARNING: Remove power to the PLC before installing or removing the LCD display.

Place the LCD Display Panel over the opening but offset approximately 3/8ths inch to the left. The Display Panel should fit easily into the opening.



Slide the Display Panel to the right until the left side of the Display Panel is flush with the left side of the PLC. The Display Panel connector will click into place.



10

Display Priority

The LCD Display Panel will show one of the following (unless power is removed from the PLC):

- Default screen (user defined or factory default)
- Menu selection
- Message from ladder program
- Error message

The built-in keypad allows you to navigate through these message displays.

On power-up the default message is normally displayed. The default message is set at the factory but can be customized by the user. Loading a custom default message is described later in this chapter.

If a system error occurs, the error message supercedes the default message (or other current display screen), and the appropriate error code is displayed for diagnostic purposes.

Menu Navigation

Beginning at the default screen, each time you press the MENU key the display will scroll to the next menu option. The up arrow and down arrow keys also scroll through the list of menus (in the direction indicated by the arrow), but ***you must initially press the MENU key (at the default screen) to activate the up and down arrow keys.***

There are seven built-in menus selections. Some of the menu items have sub-menus. The menus and sub-menus are described in this chapter. Each menu selection requires that you press the ENT key to view or change settings or values within the domain of that main menu selection.

Seven Menu Choices

Pressing and holding the MENU key will cause the display to scroll through the following menu options:

- M1 : PLC information
- M2 : System configuration
- M3 : Monitor
- M4 : Calendar read/write
- M5 : Password read/write
- M6 : Error history read
- M7 : LCD test and set

```
MENU SCREEN
>M1:PLC INFO.

>M2:SYSTEM CFG
>M3:MONITOR

>M4:CALENDAR R/W
>M5:PASSWORD R/W

>M6:ERR HISTORY
>M7:LCD TEST&SET
```

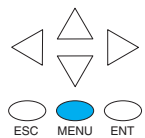

In this section we use illustrations of the LCD Display Panel keypad and display area to show how to navigate through the menu hierarchy. The example below shows the factory default screen as Screen 1 and the main menu entry screen as Screen 2.

The illustration of the keypad between the display screens indicates that pressing the MENU key causes a transition from Screen 1 to Screen 2. This type of representation is used throughout this section. When inside the menu hierarchy, the ESC key returns the display to the previous screen.

Screen 1 - factory default

```
DL06 PLC    May 08
      14:12:01
```

Press the highlighted key to transition from screen 1 to screen 2.



Screen 2

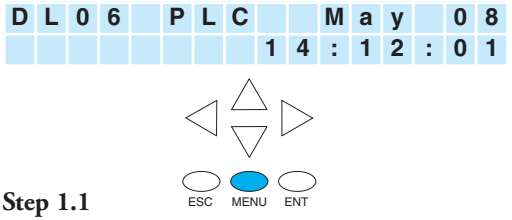
```
MENU SCREEN
>M1:PLC INFO.
```

Confirm PLC Type, Firmware Revision Level, Memory Usage, Etc.

Menu 1, M1:PLC INFO.

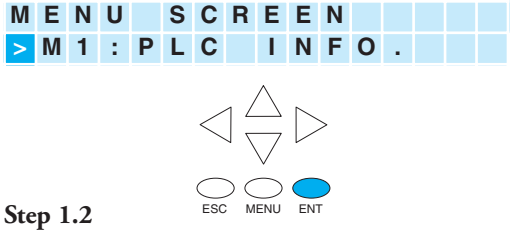
From the default screen, press the MENU key one time to arrive at the PLC INFO menu option.

Default screen

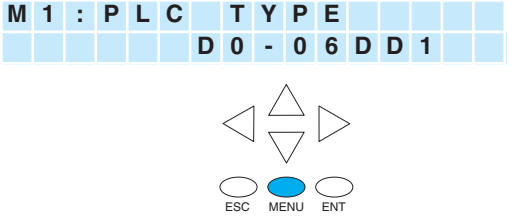


Step 1.1

Press ENT to enter this menu selection. The first screen inside the PLC INFO selection is M1:PLC TYPE. This selection displays the model number of the PLC.

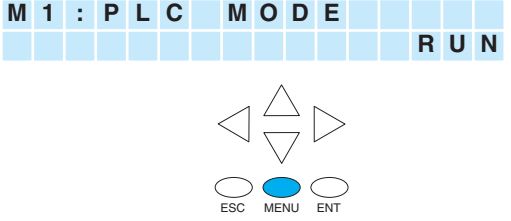


Step 1.2



Step 1.3

Press MENU again to sequence to PLC MODE. The PLC mode is either RUN, STOP (for Stop or Program Mode), TEST-STOP (for Test Stop Mode), or TEST-RUN (for Test Run Mode). You can put the DL06 in the Test Run Mode from the Test Stop Mode.

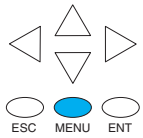


Note: The menu screen examples shown in this section assume the password/lock feature is not turned on. If the password/lock feature is turned on, the user will be prompted by a message on the Display Panel to enter the password at the appropriate time. Users without password authorization will have access to a limited number of screens.

Step 1.4

Press MENU again to sequence to FIRMWARE REV.

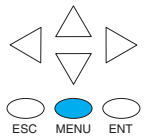
M 1 : F I R M W A R E		R E V .
		V 1 . 0 0



Step 1.5

Press MENU again to sequence to LADDER MEMORY USED. The number of words used and the total number available in the PLC are displayed.

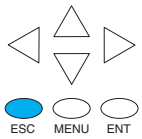
M 1 : L A D D E R	M E M O R Y
U S E D	2 1 / 7 6 8 0



Step 1.6

Press MENU again to sequence to LADDER PASSWORD, ACTIVATED OR NOT ACTIVATED. This is the last screen of the PLC INFO menu and is self-explanatory.

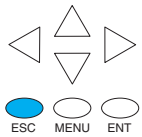
M 1 : L A D D E R	P A S S W D
N O T	A C T I V A T E D



Returns to Step 1.1

Press ESC to exit the M1 menu and return to the main menu.

M E N U	S C R E E N				
> M 1 : P L C	I N F O .				



Default screen

Press ESC once more to return to the default screen.

D L 0 6	P L C	M a y	0 8
		1 4 : 2 2 : 1 1	

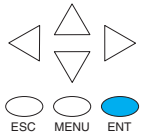
Examining Option Slot Contents

Menu 2, M2:SYSTEM CFG.

From the default screen, press MENU twice to arrive at the M2:SYSTEM CFG. (System Configuration) menu option.

Step 2.1

```
>M1:PLC  INFO.
>M2:SYSTEM  CFG.
```

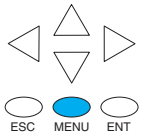


Step 2.2

Press ENT to enter the SYSTEM CFG. menu selection.

```
M2:OPTION  SLOT 1
  D0-DEVNETS
```

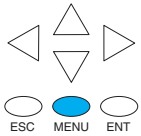
Note: This is an example only and may not represent the contents of this or any option slot on your system.



Step 2.3

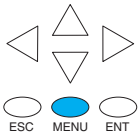
Pressing the MENU key four times will cycle through the four option slots. The model number of the option card in each slot is shown on line 2 or there is an indication that the slot is empty.

```
M2:OPTION  SLOT 2
  EMPTY  I/O  SLOT
```



Step 2.4

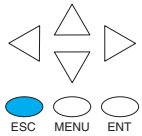
```
M2:OPTION  SLOT 3
  F0-04AD-1
```



Press the ESC key twice to return to the default screen.

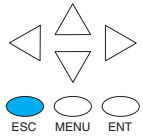
Step 2.5

```
M2:OPTION SLOT 4
  EMPTY I/O SLOT
```



Return to Step 2.1

```
>M1:PLC INFO.
>M2:SYSTEM CFG.
```



Return to default screen

```
DL06 PLC  May 08
          14:57:21
```

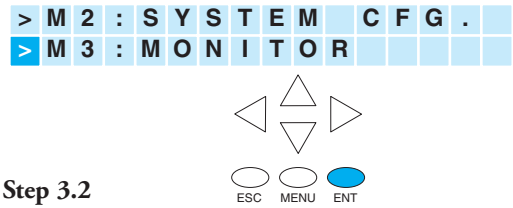
Monitoring and Changing Data Values

Menu 3, M3:MONITOR

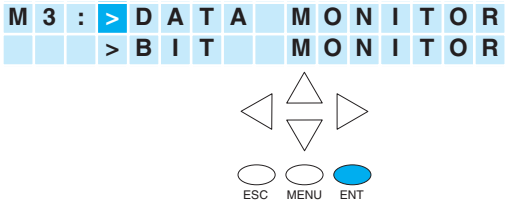
From the default screen, press MENU three times to arrive at the M3:MONITOR menu option.

The M3:MONITOR sub-menu contains the data monitor and the bit monitor. The data monitor allows you to examine the contents of memory registers or pointers to determine their contents. The default format is BCD/HEX, but the format can be changed to decimal by setting bit 8 of V7742. Please refer to the DL06 Memory Map for ranges.

Step 3.1



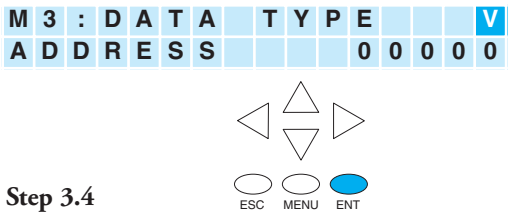
Step 3.2



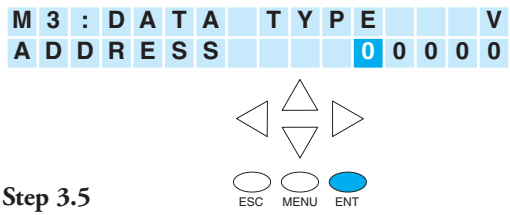
Data Monitor

Data type = V for V-memory or P for pointer. Press MENU to change data type, or press ENT to designate the register whose data you want to view or change.

Step 3.3



Step 3.4

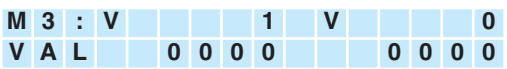


V-memory values

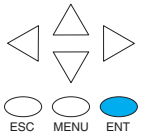
Use the right or left arrow key to move the cursor to the digit you want to change. Use the up or down arrow key to change the digit. The V-memory address is expressed as an octal number so you will not see 8's or 9's.

This screen allows you to view two adjacent V-memory locations in BCD format. The lower word is to the right. Pressing ENT makes it possible to change the value in the **lower word**. At this level of the menu hierarchy, you can also use the up and down arrow keys to scroll to other memory locations.

Step 3.5



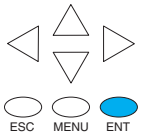
The data values on this screen will be four digits in length for BCD/HEX unless bit 8 of V7742 is set. Bit 8 of V7742 changes the data format to decimal (five digits).



Step 1.1

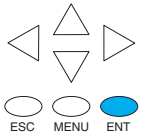
M	3	:	D	A	T	A				V					0
	C	H	G	=		0	0	0	0			0	0	0	0

Use the right or left arrow key to move the cursor to the digit you want to change. Use the up or down arrow key to move to another digit. The V-memory value is expressed as a BCD number so you will see values (in the range: 0 - F) available for each digit. The data format can be changed to decimal by setting bit 8 of V7742.



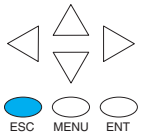
Step 1.1

M	3	:	D	A	T	A				V					0
	C	H	G	=		A	F	0	6			0	0	0	0



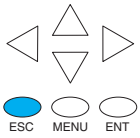
Step 1.1

M	3	:	V					1		V					0
V	A	L			0	0	0	0				A	F	0	6



Returns to Step 1.1

M	3	:	D	A	T	A		T	Y	P	E				V
A	D	D	R	E	S	S					0	0	0	0	0



Returns to default screen

Push the ESC key five (5) times to return to the default screen.

D	L	0	6		P	L	C			M	a	y		0	8
								1	5	:	0	2	:	1	3

Pointer values

Press ESC twice to return to the Step 3.3 screen with the cursor on the V, as shown. Use the up or down arrow key to change the V to P. Now, the pointer information is displayed.

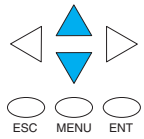
Use the up or down arrow keys to change the value of the current digit. Use the left or right arrow keys to move from one digit to the next.

At Step 3.7a, the up and down arrow keys can be used to cycle through data words. Each time you press the up or down arrow key, the address increments or decrements by one 16-bit word (addresses are expressed in octal).

To change from the data monitor to the bit monitor, press ESC three times to return to Step 3.2 (five times to return to the default screen).

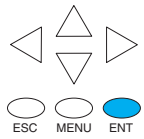
Return to Step 3.3

```
M 3 : D A T A   T Y P E       V
A D D R E S S               0 0 0 0 0
```



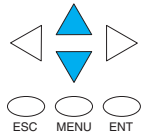
Step 3.4a

```
M 3 : D A T A   T Y P E       P
A D D R E S S               0 0 0 0 0
```



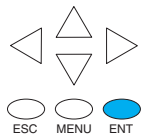
Step 3.5a

```
M 3 : D A T A   T Y P E       P
A D D R E S S               0 0 0 0 0
```



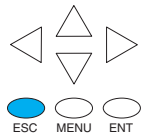
Step 3.6a

```
M 3 : D A T A   T Y P E       P
A D D R E S S               1 0 0 0 0
```



Step 3.7a

```
M 3 : D A T A         P 1 0 0 0 0
      ( V 0 0 0 0 0 )   2 0 0 0
```



Return to Step 3.3

```
M 3 : > D A T A   M O N I T O R
      > B I T     M O N I T O R
```

Bit Monitor

Bit status

From Step 3.3, press the up or down arrow key, then the ENT key. You will see one of eleven bit data types displayed. The data type that appears on the display is the last data type accessed. The address shown is also the last address accessed for that particular data type.

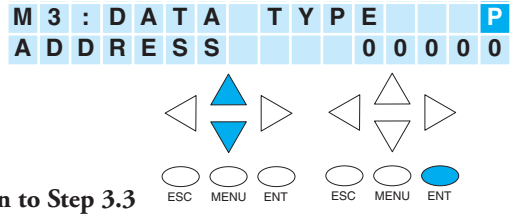
Press ENT to change the address.

Use the arrow keys to change the address as necessary.

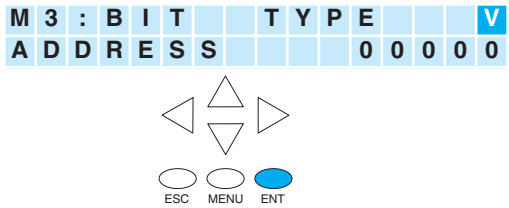
Press ENT to view the selected bits.

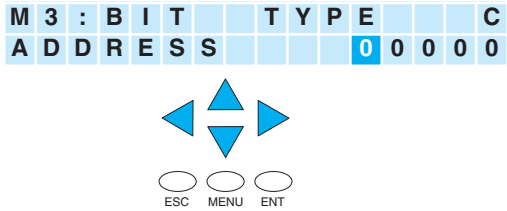
Use the left and right arrow keys to select a bit whose status you want to change. Press ENT once to see the change status screen. Press ENT again to change the status from OFF to ON or ON to OFF.

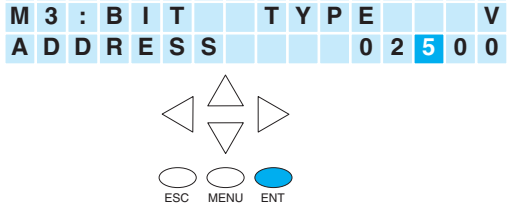
Return to Step 3.3

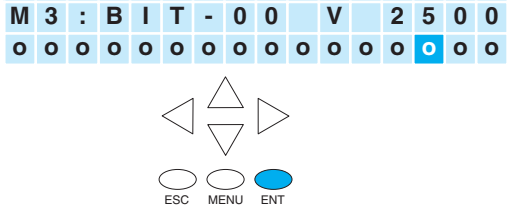


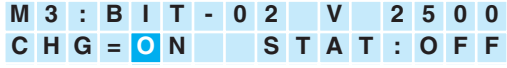
Return to Step 3.3











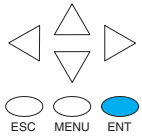
Changing Date and Time

Menu 4, M4 : CALENDAR R/W

From the default screen, press the MENU key four times to arrive at Step 4.1.

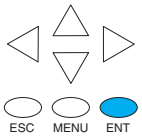
Step 4.1

```
>M3:DATA TYPE
>M4:CALENDAR R/W
```



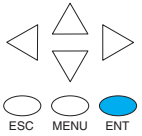
Step 4.2

```
M4:DATE  05-08-02
   TIME  01:21:28
```



Step 4.3

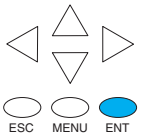
```
M4:>CHANGE DATE
  >CHANGE TIME
```



Step 4.4

```
M4:DATE  MM-DD-YY
   CHG=  05-08-02
```

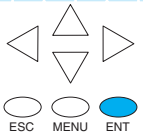
At Step 4.4, use the up and down arrows to change the value for month, day, or year. Use the left and right arrow keys to move between the different digits in the date. After making the necessary changes using the arrow keys, press the ENT key to register the changes.



Step 4.5

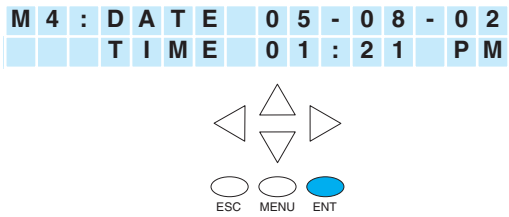
```
M4:DATE  MM-DD-YY
   SET?  05-08-02
```

You will be asked if you want to set the date to the chosen value. Press ENT again if the date is correct. You will automatically return to Step 4.2, and the new date will be displayed.

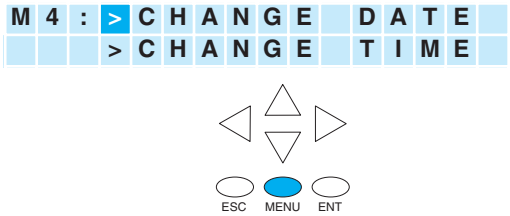


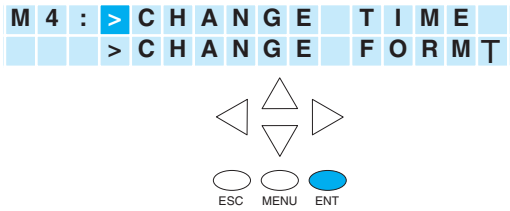
Returns to Step 4.2

In order to change the time or date/time format, press ENT again at Step 4.2.

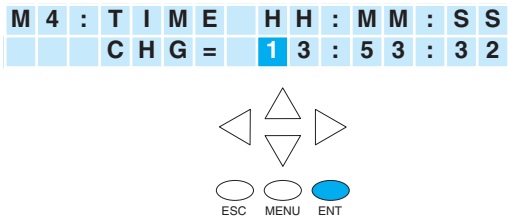


Use the up or down arrow keys or the MENU key to scroll through the sub-menu choices. At this point in our example, we will change the time setting.

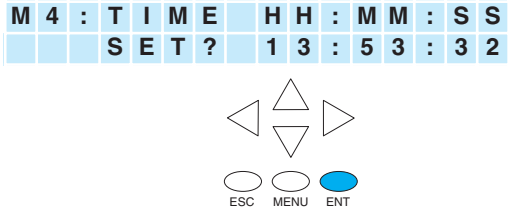




At Step 4.4, use the up and down arrows to change the value for hour, minute, or second. Use the left and right arrow keys to move between the different digits in the time. After making the necessary changes using the arrow keys, press the ENT key to register the changes.

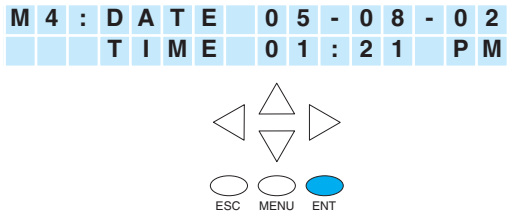


You will be asked if you want to set the date to the chosen value. Press ENT again if the date is correct. You will automatically return to Step 4.2, and the new date will be displayed.

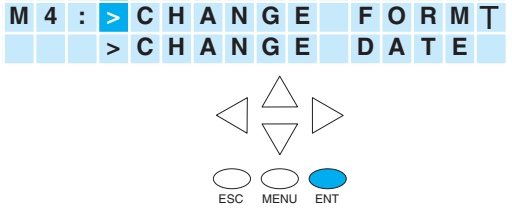


If you want to change the format for the date or time, return to Step 4.2 and press ENT.

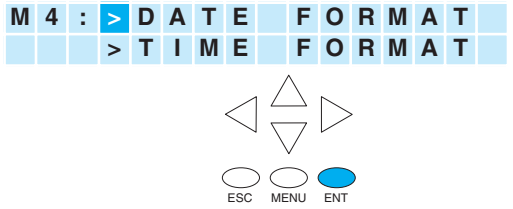
Returns to Step 4.2



Press ENT, MENU, MENU to arrive at the menu selection for changing the date or time formats. Press ENT again to enter the format changing location.



Press ENT again to enter the date format changing location, or press MENU, ENT to change the time format.



10

At Step 4.4, use the up and down arrow keys to scroll through the date formats. The choices are as follows:

MM-DD-YY (US format)

DD-MM-YY (European format)

YY-MM-DD (Asian format)

Press the ENT key to save the format changes.

If you have chosen to make a time format change, your choices are:

HH:MM US (12 hour 12:00 - 11:59AM/PM US format)

HH:MM AS (12 hour 00:00 - 11:59AM/PM Asian format)

HH:MM:SS (24 hour format)

Press the ENT key to save the format changes. Press ESC until the default screen reappears.

Date and Time Variables and Formats		
_date:us	US format	MM/DD/YY
_date:e	European format	DD/MM/YY
_date:a	Asian format	YY/MM/DD
_time:12	12 hour format	HH:MMAM/PM
_time:24	24 hour format	HH:MM:SS

Setting Password and Locking

Menu 5, M5 : PASSWORD R/W

The LCD Display Panel has its own password protection separate from the "ladder password" protection of the PLC. An LCD Display password can be used to prevent unauthorized changes to clock and calendar setup and V-memory data values. Individuals with password authorization can change clock, calender, V-memory values, force bits on or off, etc.

The LCD password inhibits unauthorized personnel from modifying the data in the DL06 with the LCD keypad. Even though the LCD password is locked, the user can still modify the data in the DL06 with DirectSOFT32 or the D2-HPP. The LCD Display Panel does not support the multi-level password.

Only menu 5 on the LCD Display can modify the LCD password.

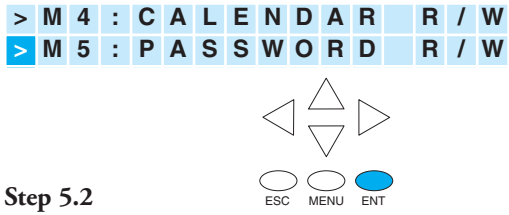
WARNING: The password protection available in DirectSOFT32 or the HPP does not prevent changes from the LCD Display Panel. To prevent changes from the LCD Display Panel, it is necessary to use the LCD password locking feature.

Step 5.1

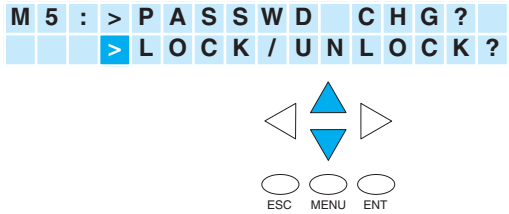
Use the MENU key to navigate to the M5 menu option. Press ENT to arrive at the display shown as Step 5.2.

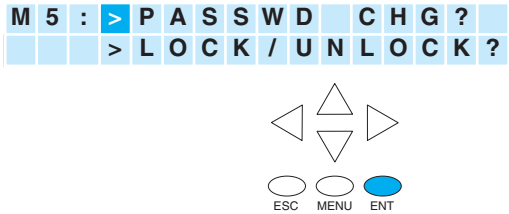
Assigning a password without locking the display allows access to all features and capabilities of the LCD.

Use the up arrow or down arrow keys to toggle between PASSWD CHG? and LOCK/UNLOCK? Eight zeroes removes the password. If the password is eight zeroes, the display will not LOCK.

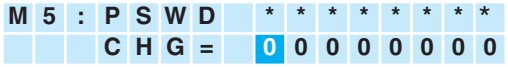


Step 5.2

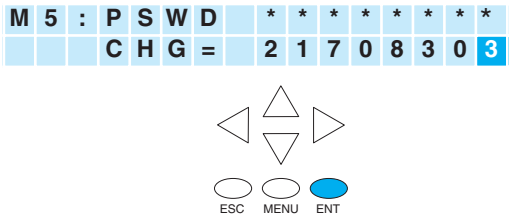


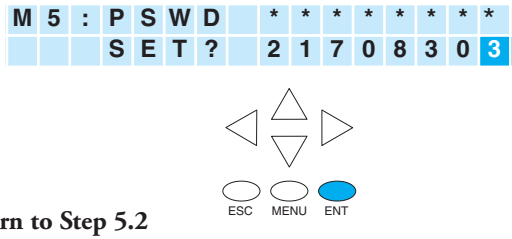


Use the up arrow or down arrow keys to scroll through number choices, and use the right arrow and left arrow keys to move from one digit position to another.

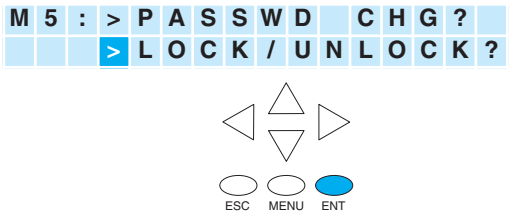


Note: It is important to record the password where it will not be forgotten and to issue the password only to qualified personnel. Full access to the LCD Display Panel gives access to change data values within the PLC.



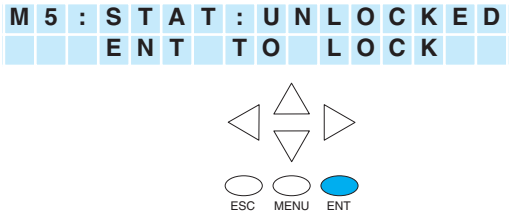


Return to Step 5.2



It is not possible to lock the display without assigning a password. It is possible to assign a password without locking the display, but doing so will not protect sensitive data.

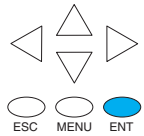
Press the ENT key at Step 5.2, and the display is now locked. If you do not wish to lock the display at this point, press ESC.



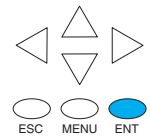
Before assigning a password, you can select "Lock/Unlock" by pressing ENT at Step 5.2.

Return to Step 5.2

M	5	:	>	P	A	S	S	W	D		C	H	G	?	
			>	L	O	C	K	/	U	N	L	O	C	K	?



M	5	:	S	T	A	T	:	U	N	L	O	C	K	E	D
			E	N	T		T	O		L	O	C	K		



Here, the display prompts you to enter a password.

M	5	:	P	S	W	D		*	*	*	*	*	*	*	*
			L	O	C	K		0	0	0	0	0	0	0	0

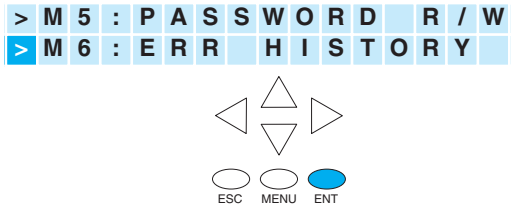
Reviewing Error History

Menu 6, M6 : ERR HISTORY

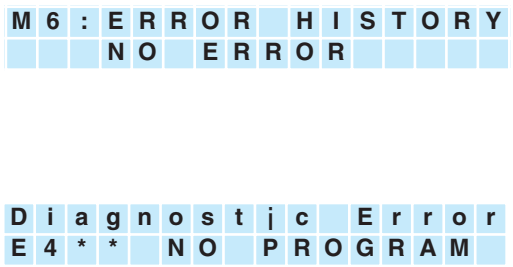
From the default screen, press the MENU key six times to arrive at Step 6.1.

Default screen

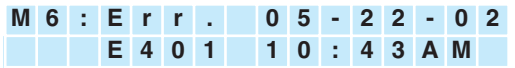
Step 6.1



The Error History screen will display "NO ERROR" if there is no record of errors. If errors have occurred, they can be identified by their Error Code. The Error Code table (see appendix B) will explain the source of the error message. The last 16 messages are displayed.Error messages are displaced when a new error message arrives



To review past error messages use the down arrow key to scroll through the historical record of error messages.



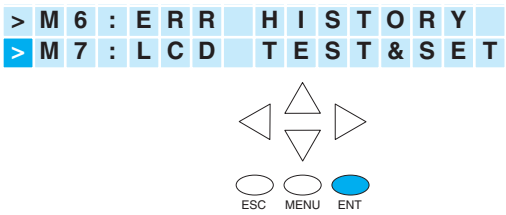
Toggle Light and Beeper, Test Keys

Menu 7, M7 : LCD TEST&SET

This menu selection gives you an opportunity to:

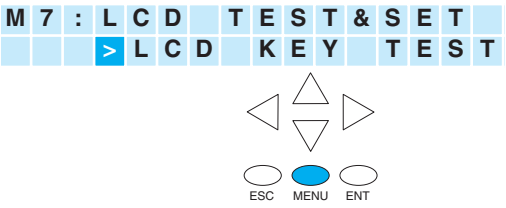
- Test each LCD key to assure that the PLC is receiving its input appropriately
- Turn the beep sound off or on
- Turn the LCD back light off or on

Make a menu selection by pressing the ENT key.

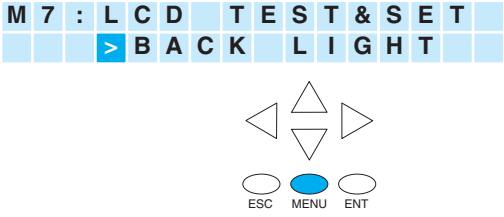


Press ENT to enter the LCD KEY TEST. All keys can be tested for proper function in this menu.

To return to the menu, press the ESC key twice or hold the ESC key down until the menu layer reappears.



Press ENT to enter the Back light test menu.



The piezo electric buzzer can be configured to provide pushbutton feedback.

PLC Memory Information for the LCD Display Panel

The valid memory ranges for storing text messages in the DL06 are:

V400 - V677 V1200 - V7577 V10000 - V17777

Data Format Suffixes for Embedded V-memory Data

Several data formats are available for displaying V-memory data on the LCD. The choices are shown in the table below. A colon is used to separate the embedded V-memory location from the data format suffix and modifier.

Data Format	Modifier		Example	Character Position/Content of the Output												
none (16 bit binary in HEX format)			V2000 = 0012	**1**	**2**	**3**	**4**									
			V2000	s	s	1	8									
	S	[:S]	V2000:S	1	8											
	C0	[:C0]	V2000:C0	0	0	1	8									
	0	[:0]	V2000:0	s	s	1	8									
:B (4 digit BCD)			V2000 = 0012	**1**	**2**	**3**	**4**									
		[:B]	V2000:B	0	0	1	2									
	S	[:BS]	V2000:BS	1	2											
	C0	[:BC0]	V2000:BC0	0	0	1	2									
	0	[:B0]	V2000:B0	s	s	1	2									
:D (32 bit binary)			V2000 = 0000													
			V2001 = 0001	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**		
		[:D]	V2000:D	s	s	s	s	s	s	6	5	5	3	6		
	S	[:DS]	V2000:DS	6	5	5	3	6								
	C0	[:DC0]	V2000:DC0	0	0	0	0	0	0	6	5	5	3	6		
	0	[:D0]	V2000:D0	s	s	s	s	s	s	6	5	5	3	6		
:DB (8 digit BCD)			V2000 = 0000													
			V2001 = 0001	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**					
		[:DB]	V2000:DB	0	0	0	1	0	0	0	0					
	S	[:DBS]	V2000:DBS	1	0	0	0	0								
	C0	[:DBC0]	V2000:DBC0	0	0	0	1	0	0	0	0					
	0	[:DB0]	V2000:DB0	s	s	s	1	0	0	0	0					
:R (Floating point number)			Value = 222.11111													
			V2000 = 1C72													
			V2001 = 435E	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
		[:R]	V2000:R	s	s	s	f	2	2	2	.	1	1	1	1	1
	S	[:RS]	V2000:RS	f	2	2	2	.	1	1	1	1	1			
	C0	[:RC0]	V2000:RC0	f	0	0	0	2	2	2	.	1	1	1	1	1
	0	[:R0]	V2000:R0	s	s	s	f	2	2	2	.	1	1	1	1	1
:E (Floating point number with exponent)			Value = 222.1													
			V2000 = 199A													
			V2001 = 435E	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
		[:E]	V2000:E	s	f	2	.	2	2	1	0	0	E	+	0	2
	S	[:ES]	V2000:ES	f	2	.	2	2	1	0	0	E	+	0	2	
	C0	[:EC0]	V2000:EC0	f	2	.	2	2	1	0	0	E	+	0	2	
	0	[:E0]	V2000:E0	f	2	.	2	2	1	0	0	E	+	0	2	

s = space f = plus/minus flag (plus = no symbol, minus = -)

The S, C0 and 0 modifiers alter the presentation of leading zeros and spaces. S removes leading spaces and left justifies the result. C0 replaces leading spaces with leading zeros. 0 is a modification of C0. 0 eleminates any leading zeros in the C0 format version and converts them to spaces.

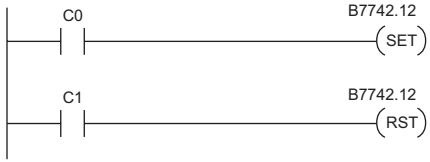
Reserved memory registers for the LCD Display Panel

Two V-memory registers are reserved for making changes to LCD functions via ladder logic. V7742 allows for bit flags to be set in the ladder program. The bit flags control such things as data formats, the back light, and the beeper. All V7742 bit flags are defined in the table on the next page.

The other reserved register is V7743. This register is used to write a customized default screen message to the LCD. A sample program for this purpose is illustrated later in this chapter.

V-memory address	Contents
V7742	**Various LCD flags** • Calendar date and time format • Default operation menu • Data format of data monitor • LCD password status flag • Key press acknowledgement buzzer on/off setting • Back light on/off setting
V7743	Default message location (writing 0 to this address returns the default message to the factory setting)

The following program segment uses the SET and RST coils to turn on and off bit 12 of V7742. When C0 is on, bit 12 is turned on. Bit 12 turns on the beeper in the LCD Display Panel. The C1 contact resets bit 12 to the off state.



V7742 bit definitions

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V7742	*	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit 1, 0	**Date display format (default = 00)**		
	00, 11	=	Month/Day/Year (US format)
	01	=	Day/Month/Year (EU format)
	10	=	Year/Month/Day (Asian format)
Bit 3, 2	**Time display format (default = 00)**		
	00, 11	=	HH:MM:SS (24 hour format)
	01	=	HH:MM PM/AM (12 hour US format - 12:00 - 11:59)
	10	=	HH:MM PM/AM (12 hour Asian format - 00:00 - 11:59)
Bit 6 - 4	**Default menu setting (default = 000)**		
	000	=	Default menu sequence, begins menu sequence with Menu 1
	001	=	Begin menu sequence with Menu 1
	010	=	Begin menu sequence with Menu 2
	011	=	Begin menu sequence with Menu 3
	100	=	Begin menu sequence with Menu 4
	101	=	Begin menu sequence with Menu 5
	110	=	Begin menu sequence with Menu 6
	111	=	Begin menu sequence with Menu 7
Bit 8	**Data monitor format (default = 0)**		
	0	=	BCD/HEX format (0000 - FFFF)
	1	=	Decimal format (00000 - 65535)
Bit 9	**New message overwrite (default = 0)**		
	0	=	New LCD message clears both lines of previous message
	1	=	New LCD message leaves previous message, overwrites specified char. only
Bit 11	**LCD password status flag (Read only)**		
	0	=	Password unlock
	1	=	Password lock
Bit 12	**Status flag beep on/off control (default = 0)**		
	0	=	Beep OFF
	1	=	Beep ON (LCD beeps continuously during ON status of this flag)
Bit 13	**Keypad beep on/off control (default = 0)**		
	0	=	Beep OFF
	1	=	Beep ON (LCD beeps when keys are pressed)
Bit 14	**LCD back light setting flag (default = 1)**		
	0	=	Light OFF
	1	=	Light ON
Bit 15	**LCD installed status flag (Read only)**		
	0	=	LCD is not installed
	1	=	LCD is installed

Changing the Default Screen

At power-up the default screen is displayed. The default screen message is set at the factory but can be customized by the user. One method of customizing the default message uses the VPRINT instruction. The VPRINT instruction is described in Chapter 5.

Factory default message

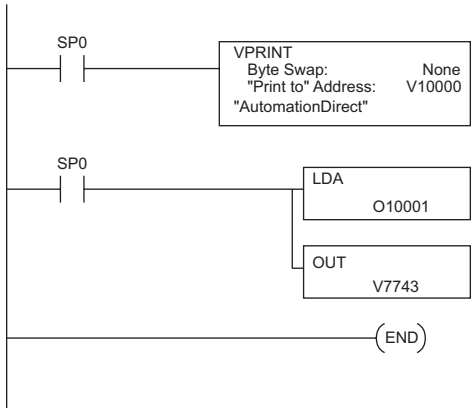
D	L	0	6		P	L	C			M	a	y		0	8
								1	4	:	2	0	:	4	9

Example program for setting the default screen message

The following program can be used to set up the default screen message. This program uses the VPRINT instruction to load ASCII text to a designated V-memory location and to embed a pointer to the current date.

The LDA and OUT instructions are used to point to the V-memory location (+1) where the text is located. The memory location V7743 is reserved for the pointer to the default message.

Note: The VPRINT instruction adds a one word (2 bytes) non-printing header to the text. For this reason, the LDA instruction points to the V-memory location V10001 rather than V10000.



V10000	00h	16h
V10001	u	A
V10002	o	t
V10003	a	m
V10004	i	t
V10005	n	o
V10006	i	D
V10007	e	r
V10010	t	c
V10011		
V10012		
V10013		
V10014		
V10015		
V10016		
V10017		
V10020		

After running this program, press MENU, then ESC or cycle power. The new default message should look as indicated. See Menu 4 instructions for changing date and time information.

A	u	t	o	m	a	t	i	o	n	D	i	r	e	c	t

Note: It is possible to return to the factory default screen by writing 0 to V7743 and cycling power.

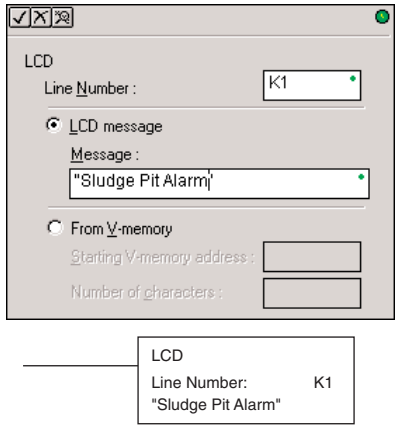
DL06 LCD Display Panel Instruction (LCD)

From the DirectSOFT32 project folder, use the Instruction Browser to locate the LCD instruction. When you select the LCD instruction and click OK, the LCD dialog will appear.

The LCD Display Panel instruction is inserted into the ladder program via the set-up dialog box shown to the right. The dialog is used to specify a message to be displayed on line 1 or line 2 LCD Display Panel.

S	l	u	d	g	e		P	i	t		A	l	a	r	m									
E	f	f	l	u	e	n	t		O	v	e	r	f	l	o									

figure A

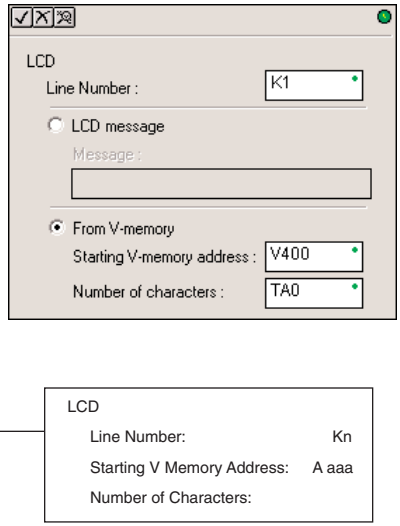


Source of message

The text of the message can originate from one of two places. It can be input directly from the instruction as a literal text string (see figure A), or it can originate as ASCII text stored in a V-memory location (figure B). In the latter case, it is necessary to specify its beginning V-memory location and length within the dialog box.

Display text strings can include embedded data. Any V-memory value or date and time settings can be embedded in the displayed text.

figure B



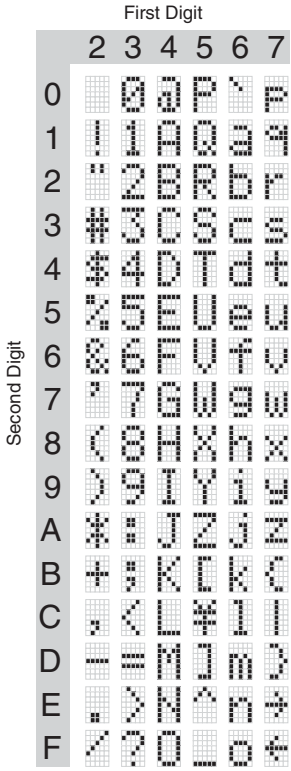
Note: The LCD Display Panel instruction is supported by DirectSOFT32, Ver. 4 or later. It is not supported by the D2-HPP handheld programmer.

ASCII Character Codes

ASCII characters can be written directly to V-memory locations and then displayed using the LCD instruction. The table to the right shows the two-digit BCD/HEX code for each character available for display.

Example:
To display an upper case A, write 41 HEX to the memory location identified by the LCD instruction.

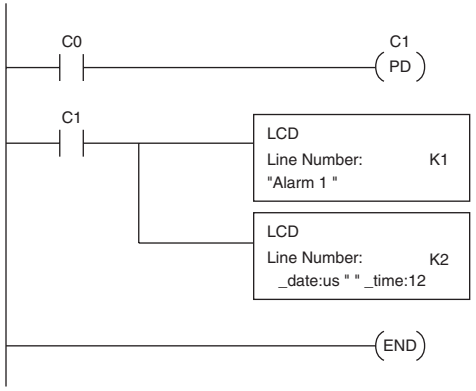
ASCII
Character Codes
(BCD/HEX)



Example program: alarm with embedded date/time stamp

The following program will display the message “Alarm 1”and the time on line K1 of the display screen with the date on line K2.

The one-shot, or positive differential (PDd), is used so that the message displays but does not block other messages or menu options. Pressing MENU or ESC will cause the alarm message text to disappear.



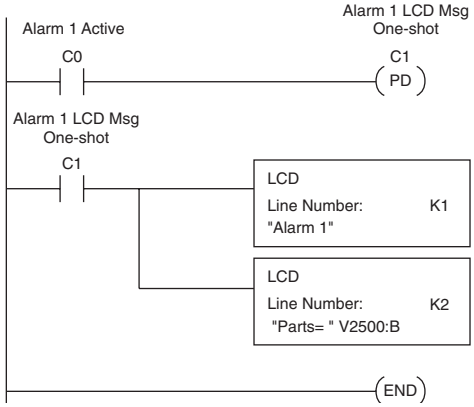
A	l	a	r	m		1								
0	5	/	0	8	/	0	2			5	:	2	3	P	M

Example program: alarm with embedded V-memory data

In this program example, the alarm notification text is displayed along with the contents of V2500. The suffix "B" is added to the memory location (V2500:B) to cause the data to be displayed as a BCD number.

In the first example, the alarm text is loaded directly via the LCD instruction. In the second example, the alarm text is loaded into V-memory and the LCD instruction is used to point to that text.

Note: When using the LCD instruction to display V2000:R, there is a limit of three characters of text because V2000:R uses 13 characters.

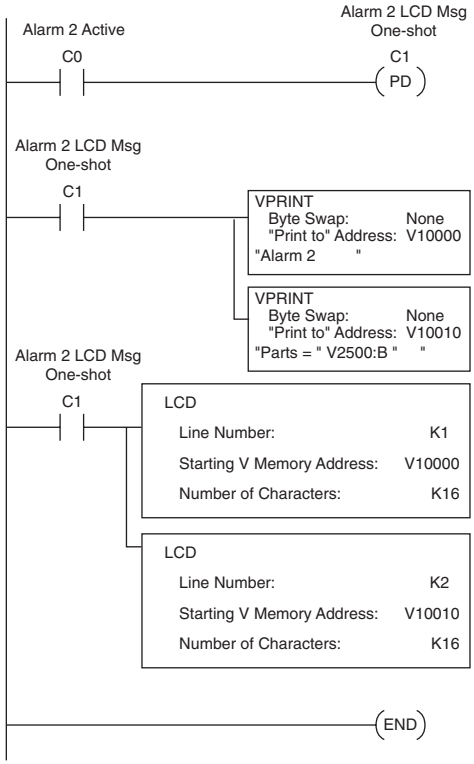


A	l	a	r	m		1								
P	a	r	t	s		=		2	4	3	7			

10

Example program: alarm text from V-memory with embedded V-memory data

This program example uses the VPRINT instruction to write ASCII text (in the appropriate character sequence) to V10000 and V10010. The LCD instruction is used as a pointer to the V-memory location where the text for each line of the display resides.



A	l	a	r	m		2								
P	a	r	t	s		=		3	5	8	9			

Auxiliary Functions

In This Appendix...

Introduction A–2
AUX 2* — RLL Operations A–4
AUX 3* — V-memory Operations A–4
AUX 4* — I/O Configuration A–4
AUX 5* — CPU Configuration A–5
AUX 6* — Handheld Programmer Configuration A–8
AUX 7* — EEPROM Operations A–8
AUX 8* — Password Operations A–9

Introduction

A

Purpose of Auxiliary Functions

Many CPU setup tasks involve the use of Auxiliary (AUX) Functions. The AUX Functions perform many different operations, including clearing ladder memory, displaying the scan time, and copying programs to EEPROM in the handheld programmer. They are divided into categories that affect different system resources. You can access the AUX Functions from *Direct*SOFT32 or from the D2–HPP Handheld Programmer. The manuals for those products provide step-by-step procedures for accessing the AUX Functions. Some of these AUX Functions are designed specifically for the Handheld Programmer setup, so they will not be needed (or available) with the *Direct*SOFT32 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 DL06 PLCs.*

AUX Function and Description		DL06
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	*
AUX 5* — CPU Configuration		
51	Modify Program Name	*
53	Display Scan Time	*
54	Initialize Scratchpad	*
55	Set Watchdog Timer	*
56	Set Communication Port 2	*
57	Set Retentive Ranges	*
58	Test Operations	*
59	Override Setup	*
5B	HSIO Interface Configuration	*
5D	Scan Control Setup	*

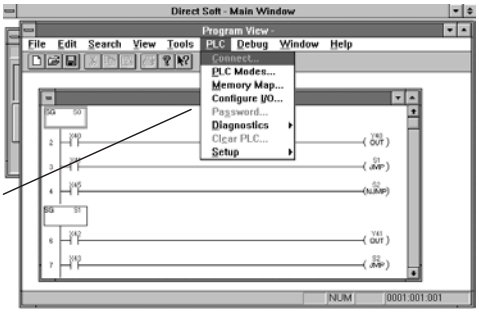
AUX Function and Description		DL06
AUX 6* — Handheld Programmer Configuration		
61	Show Revision Numbers	*
62	Beeper On / Off	HP
65	Run Self Diagnostics	HP
AUX 7* — EEPROM Operations		
71	Copy CPU memory to HPP EEPROM	HP
72	Write HPP EEPROM to CPU	HP
73	Compare CPU to HPP EEPROM	HP
74	Blank Check (HPP EEPROM)	HP
75	Erase HPP EEPROM	HP
76	Show EEPROM Type (CPU and HPP)	HP
AUX 8* — Password Operations		
81	Modify Password	*
82	Unlock CPU	*
83	Lock CPU	*

* - Supported
HP - Handheld Programmer function

A

Accessing AUX Functions via *Direct*SOFT32

*Direct*SOFT32 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*SOFT32.



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, just 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

RLL Operations auxiliary functions allow you to perform various operations on the ladder program.

AUX 21 Check Program

Both the Handheld and *Direct*SOFT32 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. Two types of checks are available:

- Syntax
- Duplicate References

The Syntax check will find a wide variety of programming errors, such as missing END statements. 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*SOFT32.

AUX 22 Change Reference

There will probably 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 we've taken existing programs and added or removed certain portions to solve new application problems. By using AUX 23 you can select and delete a portion of the program. *Direct*SOFT32 does not have a menu option for this AUX function, but you can just 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*SOFT32.

AUX 3* — V-memory Operations

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*SOFT32.

AUX 4* — I/O Configuration

AUX 41 Show I/O Configuration

This AUX function allows you to display the current I/O configuration on the DL06. Both the Handheld Programmer and DirectSOFT32. will show the I/O configuration.

AUX 5* — CPU Configuration

The following auxiliary AUX functions allow you to setup, view, or change the CPU configuration.

AUX 51 Modify Program Name

DL06 PLCs 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*SOFT32. 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 effects of AUX 54 before you use it!

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*SOFT32 by using the PLC/Diagnostics sub-menu.

AUX 54 Initialize Scratchpad

The 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*SOFT32 by using the PLC/Setup sub-menu.

AUX 55 Set Watchdog Timer

DL06 PLCs have 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*SOFT32 by using the PLC/Setup sub-menu.

AUX 56 CPU Network Address

Since the DL06 CPU has an additional communication port, you can use the Handheld to set the network address for port 2 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*SOFT32, or, as a communication port for *Direct*NET and MODBUS. Refer to *Direct*NET and MODBUS manuals for additional information about communication settings required for network operation.

A

NOTE: You will only need to use this procedure if you have port 2 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***SOFT32 by using the PLC/Setup sub-menu.

AUX 57 Set Retentive Ranges

DL06 CPUs provide certain ranges of retentive memory by default. Some of the retentive memory locations are backed up by a super-capacitor, and others are in non-volatile FLASH memory. The FLASH memory locations are V7400 to V7577. The default ranges are suitable for many applications, but you can change them if your application requires additional retentive ranges or no retentive ranges at all. The default settings are:

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

Use AUX 57 to change the retentive ranges. You can also perform this operation from within DirectSOFT32 by using the PLC/Setup sub-menu.

WARNING: The DL06 CPUs do not have battery-backed RAM. The super-capacitor will retain the values in the event of a power loss, but only up to 3 weeks. (The retention time may be as short as 4 1/2 days in 60 degree C operating temperature.)

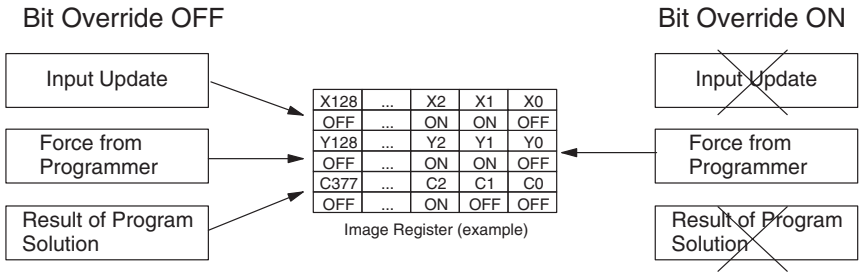
AUX 58 Test Operations

AUX 58 is used to override the output disable function of the Pause instruction. Use AUX 58 to program a single output or a range of outputs which will operate normally even when those points are within the scope of the pause instruction.

AUX 59 Bit Override

Bit override can be enabled on a point-by-point basis by using AUX 59 from the Handheld Programmer or, by a menu option from within ***Direct***SOFT36. Bit override basically disables any changes to the discrete point by the CPU. For example, if you enable bit override for X1, and X1 is off at the time, then the CPU will not change the state of X1. This means that even if X1 comes on, the CPU will not acknowledge the change. So, if you used X1 in the program, it would always be evaluated as "off" in this case. Of course, if X1 was on when the bit override was enabled, then X1 would always be evaluated as "on".

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



AUX 5B Counter Interface Configuration

AUX 5B is used with the High-Speed I/O (HSIO) function to select the configuration. You can choose the type of counter, set the counter parameters, etc. See Chapter 3 for a complete description of how to select the various counter features.

AUX 5D Select PLC Scan Mode

The DL06 CPU has two program scan modes: fixed and variable. In fixed mode, the scan time is lengthened to the time you specify (in milliseconds). If the actual scan time is longer than the fixed scan time, then the error code 'E504 BAD REF/VAL' is displayed. In variable scan mode, the CPU begins each scan as soon as the previous scan's activities complete.

A

AUX 6* — Handheld Programmer Configuration

The following auxiliary functions allow you to setup, view, or change the 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*SOFT32 from the PLC/Diagnostics sub-menu.

AUX 62 Beeper On/Off

The Handheld has a beeper that provides confirmation of keystrokes. You can use Auxiliary (AUX) Function 62 to turn off the beeper.

AUX 65 Run Self Diagnostics

If you think the Handheld Programmer is not operating correctly, you can use AUX 65 to run a self diagnostics program. You can check the following items.

- Keypad
- Display
- LEDs and Backlight
- Handheld Programmer EEPROM check

AUX 7* — EEPROM Operations

The following auxiliary functions allow you to move the ladder program from one area to another and perform other program maintenance tasks.

Transferrable Memory Areas

Many of these AUX functions allow you to copy different areas of memory to and from the CPU and handheld programmer. The following table shows the areas that may be mentioned.

Option and Memory Type	DL06 Default Range
1:PGM — Program	$00000 – $02047
2:V — V memory	$00000 – $07777
3:SYS — System	Non-selectable copies system parameters
4:etc (All)— Program, System and non-volatile Vmemory only	Non-selectable

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.

AUX 72 HPP EEPROM to CPU

AUX 72 copies information from the EEPROM installed in the Handheld Programmer to CPU memory in the DL06. You can copy different portions of EEPROM (HP) memory to the 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.

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

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. The password 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.)

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 82 and AUX 83 to lock and unlock the CPU.

You can also enter or modify a password from within DirectSOFT32 by using the PLC/Password sub-menu. This feature works slightly differently in DirectSOFT32. 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 DL06 CPUs support multi-level password protection of the ladder program. This allows password protection while not locking the communication port to an operator interface. The multi-level password can be invoked by creating a password with an upper case "A" followed by seven numeric characters (e.g. A1234567).

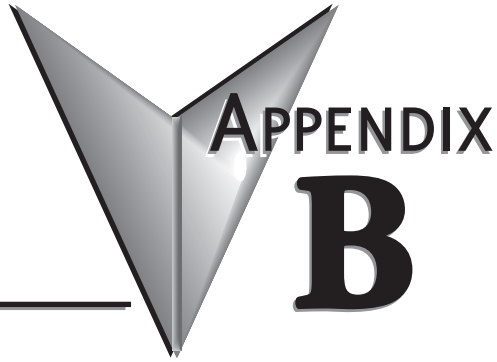
AUX 82 Unlock CPU

AUX 82 can be used to unlock a CPU that has been password protected. ***Direct***SOFT32 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***SOFT32 since the CPU is automatically locked whenever you exit the software package.

DL06 Error codes



In This Appendix...

DL06 Error Codes B–2
DL06 Error Codes, continued B–3

DL06 Error Codes

DL06 Error Code	Description
E001 CPU FATAL ERROR	You may possibly clear the error by power cycling the CPU. If the error returns, replace the DL06.
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 use AUX 55 to extend the time allotted to the watchdog timer.
E041 CPU BATTERY LOW	The DL06 battery is low and should be replaced. SP43 will be on and the error code will be stored in V7757.
E104 WRITE FAILED	A write to the DL06 was not successful. Power cycle the DL06. If the error returns, replace the DL06.
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 Micro DL06.
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 DL06.
E2** I/O MODULE FAILURE	An I/O module has failed. Run AUX42 to determine the actual error.
E202 MISSING I/O MODULE	An I/O module has failed to communicate with the DL06 or is missing from the slot. 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 DL06.
E252 NEW I/O CFG	This error occurs when the auto configuration check is turned on in the DL06 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.
E263 CONFIGURED I/O ADDRESS OUT OF RANGE	Out of range addresses have been assigned while manually configuring the I/O. Correct the address assignments using AUX46.
E311 HP COMM ERROR 1	A request from the handheld programmer could not be processed by the DL06. Clear the error and retry the request. If the error continues replace the DL06 SP46 will be on and the error code will be stored in V7756.

DL06 Error Codes, continued

DL06 Error Code	Description
E312 HP COMM ERROR 2	A data error was encountered during communications with the DL06. Clear between the two devices, replace the handheld programmer, then if necessary replace the DL06. The error code will be stored in V7756.
E313 HP COMM ERROR 3	An address error was encountered during communications with the DL06. 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 DL06 The error code will be stored in V7756.
E316 HP COMM ERROR 6	A mode error was encountered during communications with the DL06. Clear the error and retry the request. If the error continues replace the handheld programmer, then if necessary replace the DL06. The error code will be stored in V7756.
E320 HP COMM TIME-OUT	The DL06 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 DL06 first and then the handheld programmer if necessary.
E321 COMM ERROR	A data error was encountered during communication with the DL06. Check to insure cabling is correct and not defective. Power cycle the system and if the error continues replace the DL06 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 MOVMC or LDLBL instruction was used without the appropriate label. Refer to Chapter 5 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.
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>256	There is greater than 256 SBR or DLBL instructions in the program. This error is also returned if there is greater than 4 INT instructions used in the 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 LBL REFERENCE	Two or more LBL instructions exist in the application program with the same number. A unique number must be allowed for each 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) are not allowed. SP52 will be on and the error code will be stored in V7755.
E431 INVALID ISG/SG ADDRESS	An ISG or SG instruction must not be placed after the end statement (such as inside a subroutine). SP52 will be on and the error code will be stored in V7755.

DL06 Error Code	Description
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.
E434 INVALID RTC ADDRESS	A RTC 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.
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.
E437 INVALID IRTC ADDRESS	An IRTC 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/UDC	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.

DL06 Error Code	Description
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	An STR/STRN 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.
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 INSTRUCTION	An instruction exists between convergence stages.
E482 MAX. CV INSTRUCTIONS EXCEEDED	Number of CV instructions exceeds 17.
E483 INVALID CV JUMP 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 MISSING REQUIRED INSTRUCTION	A CV JMP instruction is not placed between the CV and the [SG, ISG, ST BLK, END BLK, END] instruction.
E486 INVALID CALL BLK ADDRESS	CALL BLK is used in a subroutine or a program interrupt routine. The CALL BLK instruction may only be used in the main program area (before the END statement).
E487 MISSING ST BLK INSTRUCTION	The CALL BLK instruction is not followed by a ST BLK instruction.
E488 INVALID ST BLK ADDRESS	The ST BLK instruction is used in a subroutine or a program interrupt. Another ST BLK instruction is used between the CALL BLK and the END BLK instructions.
E489 DUPLICATE CR REFERENCE	The control relay used for the BLK instruction is being used as an output elsewhere.
E490 MISSING SG INSTRUCTION	The BLK instruction is not immediately followed by the SG instruction.

DL06 Error Code	Description
E491 INVALID ISG INSTRUCTION ADDRESS	There is an ISG instruction between the ST BLK and END BLK instructions.
E492 INVALID END BLK ADDRESS	The END BLK instruction is used in a subroutine or a program interrupt routine. The END BLK instruction is not followed by a ST BLK instruction.
E493 MISSING END REQUIRED INSTRUCTION	A [CV, SG, ISG, ST BLK, END] instruction must immediately follow the END BLK instruction.
E494 MISSING END BLK INSTRUCTION	The ST BLK instruction is not followed by a END BLK instruction.
E499 PRINT INSTRUCTION	Invalid PRINT instruction 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 command 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.
E525 MODE SWITCH	An operation was attempted by the handheld programmer while the DL06 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 key.
E528 CPU MODE	The operation attempted is not allowed during a Run Time Edit.
E540 CPU LOCKED	The DL06 has been password locked. To unlock the DL06 use AUX82 with the password.
E541 WRONG PASSWORD	The password used to unlock the DL06 with AUX82 was incorrect.
E542 PASSWORD RESET	The DL06 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 DL06.
E602 INSTRUCTION MISSING	A search function was performed and the instruction was not found.

DL06 Error Code	Description
E603 DATA MISSING	A search function was performed and the data 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	An attempt to transfer more data between the DL06 and handheld programmer than the receiving device can hold.
E621 EEPROM NOT BLANK	An attempt to write to a non-blank EEPROM in the handheld programmer 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 in the handheld programmer which contains system information only.
E624 V-MEMORY ONLY	A function was requested with an EEPROM in the handheld programmer which contains V-memory data only.
E625 PROGRAM ONLY	A function was requested with an EEPROM in the handheld programmer which contains program data only.
E627 BAD WRITE	An attempt to write to a faulty EEPROM in the handheld programmer was made. Replace the EEPROM if necessary.
E628 EEPROM TYPE ERROR	The wrong size EEPROM is being used.
E640 COMPARE ERROR	A compare between the EEPROM handheld programmer and the DL06 was found to be in error.
E642 CHECKSUM ERROR	An error was detected while data was being transferred to the handheld programmer's EEPROM. Check cabling and retry the operation.
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.

Instruction Execution Times

In This Appendix...

Introduction C–2
Instruction Execution Times C–3

Introduction

This appendix contains several tables that provide the instruction execution times for DL06 Micro PLCs. Many of the execution times depend on the type of data used with the instruction. Registers may be classified into the following types:

- Data (word) Registers
- Bit Registers

V-Memory Data Registers

Some V-memory locations are considered data registers, such as timer or counter current values standard user V memory is classified as a V-memory data register. Note that you can load a bit pattern into these types of registers, even though their primary use is for data registers. The following locations are data registers:

Data Registers	DL06
Timer Current Values	V0 - V377
Counter Current Values	V1000 - V1177
User Data Words	V400 - V677 V1200 - V7377 V10000 - V17777

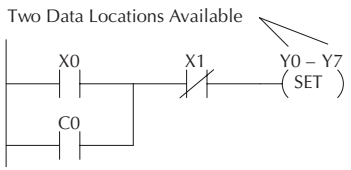
V-Memory Bit Registers

You may recall that some of the discrete points such as X, Y, C, etc. are automatically mapped into V memory. The following bit registers contain this data:

Bit Registers	DL06
Input Points (X)	V40400 - V40437
Output Points (Y)	V40500 - V40537
Control Relays (C)	V40600 - V40677
Stages (S)	V41000 - V41077
Timer status Bits	V41100 - V41177
Counter status Bits	V41140 - V41147
Special Relays (SP)	V41200 - V41237

How to Read the Tables

Some instructions can have more than one parameter. For example, the SET instruction shown in the ladder program to the right can set a single point or a range of points.



In these cases, execution times depend on the amount and type of parameters. The execution time tables list execution times for both situations, as shown below:

SET	1st #: X, Y, C, S		9.2 µs
	2nd #: X, Y, C, S	(N pt)	9.6 µs + 0.9 µs x N
RST	1st #: X, Y, C, S		9.2 µs
	2nd #: X, Y, C, S	(N pt)	9.6 µs + 0.9 µs x N

Execution depends on numbers of locations and types of data used

Instruction Execution Times

Boolean Instructions

Boolean Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
STR	X, Y, C, T, CT, S,SP, GX, GY	0.67 µs	0.00 µs
STRN	X, Y, C, T, CT, S,SP, GX, GY	0.67 µs	0.0 µs
OR	X, Y, C, T, CT, S,SP, GX, GY	0.51 µs	0.51 µs
ORN	X, Y, C, T, CT, S,SP, GX, GY	0.55 µs	0.55 µs
AND	X, Y, C, T, CT, S,SP, GX, GY	0.42 µs	0.42 µs
ANDN	X, Y, C, T, CT, S,SP, GX, GY	0.51 µs	0.51 µs
ANDSTR	None	0.37 µs	0.37 µs
ORSTR	None	0.37 µs	0.37 µs
OUT	X, Y, C, GX, GY	1.82 µs	1.82 µs
OROUT	X, Y, C, GX, GY	2.09 µs	2.09 µs
NOT	None	1.04 µs	1.04 µs
SET	1st #: X, Y, C, S, 2nd #: X, Y, C, S (N pt)	9.2 µs 9.6 µs+0.9 µs x N	1.0 µs 1.1 µs
RST	1st #: X, Y, C,S, GX, GY 2nd #: X, Y, C,S (N pt), GX, GY	9.2 µs 9.6 µs+0.9 µs x N	1.0 µs 1.1 µs
	1st #: T, CT, GX, GY 2nd #: T, CT (N pt), GX, GY	25.7 µs 16.8 µs + 2.7 µs x N	1.1 µs 1.4 µs
PAUSE	1wd: Y 2wd: Y (N points)	5.6 µs 9.2 µs + 0.3 µs x N	5.4 µs 4.8 µs

C

Comparative Boolean Instructions

Comparative Boolean Instructions			DL06	
Instruction	**Legal Data Types**		**Execute**	**Not Execute**
STRE	***1st***	***2nd***		
	V Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.7 µs	27.7 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.7 µs	27.7 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
STRNE	***1st***	***2nd***		
	V: Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	30.3 µs	30.3 µs
		V:Bit Reg.	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg.	30.3 µs	30.3 µs
		V:Bit Reg.	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

Comparative Boolean Instructions (cont.)			DL06	
Instruction	**Legal Data Types**		**Execute**	**Not Execute**
ORE	***1st***	***2nd***		
	V Data Reg	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg	30.3 µs	30.3 µs
		V:Bit Reg	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	50.4 µs	50.4 µs
		P:Indir. (Bit)	50.4 µs	50.4 µs
	P:Indir. (Bit)	V:Data Reg	30.3 µs	30.3 µs
		V:Bit Reg	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	50.4 µs	50.4 µs
		P:Indir. (Bit)	50.4 µs	50.4 µs
ORNE	***1st***	***2nd***		
	Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

C

Comparative Boolean Instructions (cont.)			DL06	
Instruction	**Legal Data Types**		**Execute**	**Not Execute**
ANDE	***1st***	***2nd***		
	V Data Reg.	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
ANDNE	***1st***	***2nd***		
	V: Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

Comparative Boolean Instructions			DL06	
Instruction	**Legal Data Types**		**Execute**	**Not Execute**
STR	***1st***	***2nd***		
	T, CT	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V Data Reg	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
STRN	***1st***	***2nd***		
	T, CT	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit	30.2 µs	30.2 µs

Comparative Boolean Instructions			DL06	
Instruction	**Legal Data Types**		**Execute**	**Not Execute**
STRN (cont.)	***1st***	***2nd***		
	V: Bit Reg	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

Comparative Boolean Instructions (cont.)			DL06	
Instruction	**Legal Data Types**		**Execute**	**Not Execute**
OR	***1st***	***2nd***		
	T, CT	V Data Reg	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

C

Comparative Boolean Instructions (cont.)			DL06	
Instruction	**Legal Data Types**		**Execute**	**Not Execute**
ORN	***1st***	***2nd***		
	T, CT	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Data Reg	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

Comparative Boolean Instructions (cont.)			DL06	
Instruction	**Legal Data Types**		**Execute**	**Not Execute**
AND	***1st***	***2nd***		
	T, CT	V Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V Data Reg.	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

C

Comparative Boolean Instructions (cont.)			DL06	
Instruction	**Legal Data Types**		**Execute**	**Not Execute**
ANDN	***1st***	***2nd***		
	T, CT	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Data Reg.	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

Bit of Word Boolean Instructions

Bit of Word Boolean Instructions		DL06	
Instruction	**Legal Data Types**	**Execute**	**Not Execute**
STRB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	3.1 µs 3.1 µs 30.0 µs 30.0 µs	3.1 µs 3.1 µs 30.0 µs 30.0 µs
STRNB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	3.0 µs 3.0 µs 29.8 µs 29.8 µs	3.0 µs 3.0 µs 29.8 µs 29.8 µs
ORB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.9 µs 2.9 µs 29.9 µs 29.9 µs	2.9 µs 2.9 µs 29.9 µs 29.9 µs
ORNB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.8 µs 2.8 µs 29.6 µs 29.6 µs	2.8 µs 2.8 µs 29.6 µs 29.6 µs
ANDB	V:Data Reg V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.8 µs 2.8 µs 29.6 µs 29.6 µs	2.8 µs 2.8 µs 29.6 µs 29.6 µs
ANDNB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.7 µs 2.7 µs 29.6 µs 29.6 µs	2.7 µs 2.7 µs 29.6 µs 29.6 µs
OUTB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	3.1 µs 3.1 µs 30.3 µs 30.3 µs	3.4 µs 3.4 µs 30.7 µs 30.7 µs
SETB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.4 µs 13.4 µs 41.1 µs 41.1 µs	3.4 µs 3.4 µs 29.1 µs 29.1 µs
RSTB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.5 µs 13.5 µs 41.3 µs 41.3 µs	1.4 µs 1.4 µs 29.1 µs 29.1 µs

Immediate Instructions

Immediate Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
LDI	V	20.6 µs	1.1 µs
LDIF	1st #: Y 2nd #: K Constant	26.6 µs+0.9µs x N	1.4 µs
STRI	X	19.3 µs	19.3 µs
STRNI	X	19.4 µs	19.4 µs
ORI	X	19.1 µs	18.7 µs
ORNI	X	19.2 µs	18.9 µs
ANDI	X	18.7 µs	18.7 µs
ANDNI	X	18.8 µs	18.8 µs
OUTI	Y	25.5 µs	25.5 µs
OROUTI	Y	25.7 µs	25.7 µs
OUTIF	1st #: Y 2nd #: Y (N pt)	66.1 µs+0.9µs x N	1.4 µs
SETI	1st #: Y 2nd #: K Constant	23.1 µs, 22.8 µs+1.4µsxN	0.9 µs, 0.9 µs
RSTI	1st #: Y 2nd #: Y (N pt)	23.2 µs, 22.8 µs+1.4µsxN	0.9 µs, 0.9 µs

Timer, Counter and Shift Register

Timer, Counter and Shift Register			DL06	
Instruction	Legal Data Types		Execute	Not Execute
	1st	***2nd***		
TMR	T	V Data Reg.	26.8 µs	7.3 µs
		V:Bit Reg	26.8 µs	7.3 µs
		K:Constant	20.0 µs	4.8 µs
		P:Indir. (Data)	45.6 µs	30.2 µs
		P:Indir. (Bit)	45.6 µs	30.2 µs
TMRF	T	V:Data Reg.	51.4 µs	7.3 µs
		V:Bit Reg	51.4 µs	7.3 µs
		K:Constant	48.4 µs	4.6 µs
		P:Indir. (Data)	75.9 µs	30.2 µs
		P:Indir. (Bit)	75.9 µs	30.2 µs
TMRA	T	V:Data Reg.	48.9 µs	7.3 µs
		V:Bit Reg	48.9 µs	7.3 µs
		K:Constant	45.0 µs	4.6 µs
		P:Indir. (Data)	75.9 µs	30.2 µs
		P:Indir. (Bit)	75.9 µs	30.2 µs

Timer, Counter and Shift Register (cont.)			DL06	
Instruction	**Legal Data Types**		**Execute**	**Not Execute**
	1st	***2nd***		
TMRAF	T	V Data Reg.	54.2 µs	7.3 µs
		V:Bit Reg	54.2 µs	7.3 µs
		K:Constant	50.3 µs	4.6 µs
		P:Indir. (Data)	81.2 µs	30.2 µs
		P:Indir. (Bit)	81.2 µs	30.2 µs
CNT	CT	V:Data Reg.	25.8 µs	7.3 µs
		V:Bit Reg	25.8 µs	7.3 µs
		K:Constant	22.2 µs	4.6 µs
		P:Indir. (Data)	53.5 µs	30.2 µs
		P:Indir. (Bit)	53.5 µs	30.2 µs
SGCNT	CT	V:Data Reg.	27.3 µs	7.3 µs
		V:Bit Reg	27.3 µs	7.3 µs
		K:Constant	23.5 µs	4.6 µs
		P:Indir. (Data)	54.9 µs	30.2 µs
		P:Indir. (Bit)	54.9 µs	30.2 µs
UDC	CT	V:Data Reg	39.8 µs	7.3 µs
		V:Bit Reg	39.8 µs	7.3 µs
		K:Constant	35.4 µs	4.6 µs
		P:Indir. (Data)	67.8 µs	30.2 µs
		P:Indir. (Bit)	67.8 µs	30.2 µs
SR	C (N points to shift)		17.8 µs + 0.9 µs x N	9.8 µs

C

Accumulator Data Instructions

Accumulator / Stack Load and Output Data Instructions			DL06	
Instruction	**Legal Data Types**		**Execute**	**Not Execute**
LD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)		11.8 µs 11.8µs 9.0 µs 33.9 µs 33.9 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
LDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)		12.2 µs 12.2 µs 9.0 µs 37.8 µs 37.8 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
LDF	***1st*** X, Y, C, S T, CT, SP	***2nd*** K:Constant	20.5 µs+0.9 µsxN	0.9 µs
LDA	O: (Octal constant for address)		10.4 µs	1.0 µs
LDR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)		29.5 µs 29.5 µs 25.5 µs 54.9 µs 54.9 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
LDSX	K: Constant		14.6 µs	1.0 µs
LDX	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		10.8 µs 10.8 µs 45.2 µs 45.2 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
OUT	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		9.3 µs 9.3 µs 35.2 µs 35.2 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs

Accumulator / Stack Load and Output Data Instructions (continued)			DL06	
Instruction	Legal Data Types		Execute	Not Execute
OUTD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		10.2 µs 10.2 µs 35.8 µs 35.8 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
OUTF	***1st*** X, Y, C	***2nd*** K:Constant	54 µs+1.0 µsxN	0.9 µs
OUTL	V:Data Reg. V:Bit Reg.		13.5 µs 13.5 µs	1.0 µs 1.0 µs
OUTM	V:Data Reg. V:Bit Reg.		13.7 µs 13.7 µs	1.0 µs 1.0 µs
OUTX	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		17.2 µs 17.2 µs 43.4 µs 43.4 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
POP	None		8.4 µs	1.0 µs

Logical Instructions

Logical (Accumulator) Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
AND	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	7.9 µs 7.9 µs 33.4 µs 33.4 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
ANDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	8.9 µs 8.9 µs 5.7 µs 34.4 µs 34.4 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
ANDF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	21.6 µs + 0.9 µs x N	1.0 µs
ANDS	None	10.0 µs	1.0 µs
OR	V:Data Reg V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	8.1 µs 8.1 µs 33.8 µs 33.8 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
ORD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	9.0 µs 9.0 µs 5.8 µs 34.5 µs 34.5 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs

Logical (Accumulator) Instructions (cont.)		DL06	
Instruction	**Legal Data Types**	**Execute**	**Not Execute**
ORF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	20.9 µs + 0.9 µs x N	1.0 µs
ORS	None	10.2 µs	1.0 µs
XOR	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	8.0 µs 8.0 µs 33.6 µs 33.6 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
XORD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	9.0 µs 9.0 µs 5.4 µs 34.4 µs 34.4 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
XORF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	20.9 µs + 0.9 µs x N	1.0 µs
XORS	None	10.1 µs	1.0 µs
CMP	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	9.4 µs 9.4 µs 34.9 µs 34.9 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
CMPD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	9.9 µs 9.9 µs 6.7 µs 35.4 µs 35.4 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
CMPF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	20.9 µs + 1.0 µs x N	1.0 µs
CMPR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	42.8 µs 42.8 µs 38.4 µs 69.0 µs 69.0 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
CMPS	None	11.2 µs	1.0 µs

Math Instructions

Math Instructions (Accumulator)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
ADD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	78.4 µs 78.4 µs 101.2 µs 101.2 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs
ADDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Daa) P:Indir. (Bit)	83.3 µs 83.3 µs 67.7 µs 101.2 µs 101.2 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
SUB	V:Data Reg. V:Bit Reg P:Indir. (Data) P:Indir. (Bit)	77.4 µs 77.4 µs 95.1 µs 95.1 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs
SUBD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	82.5 µs 82.5 µs 66.0 µs 99.7 µs 99.7 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
MUL	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	266.1 µs 266.1 µs 286.9 µs 290.0 µs 290.0 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
MULD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	839.1 µs 839.1 µs 863.1 µs 863.1 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs
DIV	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	363.9 µs 363.9 µs 384.4 µs 419.8 µs 419.8 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
DIVD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	398.3 µs 398.3 µs 390.9 µs 390.9 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs
INC	V:Data Reg V:Bit Reg P:Indir. (Data) P:Indir. (Bit)	48.5 µs 48.5 µs 74.7 µs 74.7 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
DEC	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	47.5 µs 47.5 µs 71.5 µs 71.5 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
INCB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.2 µs 13.2 µs 38.6 µs 38.6 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
DECB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.2 µs 13.2 µs 38.0 µs 38.0 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs

Math Instructions (Accumulator) (continued)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
ADDB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.9 µs 24.9 µs 23.5 µs 51.1 µs 51.1 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
ADDBD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.4 µs 24.4 µs 20.7 µs 50.7 µs 50.7 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
SUBB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.7 µs 24.7 µs 23.3 µs 50.6 µs 50.6 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
SUBBD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.2 µs 24.2 µs 20.2 µs 50.2 µs 50.2 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
MULB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	10.8 µs 10.8 µs 8.2 µs 37.1 µs 37.1 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
DIVB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	28.7 µs 28.7 µs 26.1 µs 54.9 µs 54.9 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
ADDR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	48.1 µs 48.1 µs 41.7 µs 74.3 µs 74.3 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
SUBR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	50.1 µs 50.1 µs 58.7 µs 76.3 µs 76.3 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
MULR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	54.2 µs 54.2 µs 42.7 µs 80.4 µs 80.4 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
DIVR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	50.1 µs 50.1 µs 58.7 µs 76.3 µs 76.3 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
ADDF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	109.3 µs + 0.9 µs x N	1.0 µs

Math Instructions (Accumulator) (continued)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
SUBF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	107.3 µs + 0.9 µs x N	1.0 µs
MULF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	352.5 µs + 0.9 µs x N	1.0 µs
DIVF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	477.3 µs + 0.8 µs x N	1.0 µs
ADDS	None	99.5 µs	1.0 µs
SUBS	None	97.5 µs	1.0 µs
MULS	None	342.5 µs	1.0 µs
DIVS	None	467.3 µs	1.0 µs
ADDBS	None	24.3 µs	1.0 µs
SUBBS	None	23.7 µs	1.0 µs
MULBS	None	11.7 µs	1.0 µs
DIVBS	None	29.7 µs	1.0 µs
SQRTR	None	87.9 µs	1.0 µs
SINR	None	226.8 µs	1.0 µs
COSR	None	213.1 µs	1.0 µs
TANR	None	285.5 µs	1.0 µs
ASINR	None	489.8 µs	1.0 µs
ACOSR	None	508.3 µs	1.0 µs
ATANR	None	317.1 µs	1.0 µs

Differential Instructions

Differential Instructions		DL06	
Instruction	**Legal Data Types**	**Execute**	**Not Execute**
PD	X, Y, C	14.4 µs	14.4 µs
STRPD	X, Y, C, S, T, CT	5.4 µs	5.4 µs
STRND	X, Y, C, S, T, CT	7.3 µs	7.3 µs
ORPD	X, Y, C, S, T, CT	6.8 µs	5.2 µs
ORND	X, Y, C, S, T, CT	7.1 µs	4.9 µs
ANDPD	X, Y, C, S, T, CT	6.8 µs	5.2 µs
ANDND	X, Y, C, S, T, CT	7.1 µs	4.9 µs

Bit Instructions

Bit Instructions (Accumulator)		DL06	
Instruction	**Legal Data Types**	**Execute**	**Not Execute**
SUM	None	6.7 µs	1.0 µs
SHFR	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	12.1 µs + 0.1 x N 8.4 µs + 0.1 x N	0.9 µs
SHFL	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	12.1 µs + 0.1 x N 8.4 µs + 0.1 x N	0.9 µs
ROTR	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	16.4 µs 16.4 µs 12.9 µs	1.0 µs 1.0 µs 1.0 µs
ROTL	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	16.4 µs 16.4 µs 12.7 µs	1.0 µs 1.0 µs 1.0 µs
ENCO	None	33.9 µs	0.9 µs
DECO	None	5.7 µs	1.0 µs

Number Conversion Instructions

Number Conversion Instructions (Accumulator)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
BIN	None	100.2 µs	0.9 µs
BCD	None	95.2 µs	0.9 µs
INV	None	2.5 µs	1.0 µs
BCDPL	None	75.6 µs	1.0 µs
ATH	V	25.4 µs	1.0 µs
HTA	V	25.4 µs	1.0 µs
GRAY	None	110.8 µs	1.0 µs
SFLDGT	None	23.1 µs	1.0 µs
BTOR	None	18.6 µs	1.0 µs
RTOB	None	8.6 µs	1.0 µs
RADR	None	51.4 µs	1.0 µs
DEGR	None	81.5 µs	1.0 µs

Table Instructions

Table Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
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	60.2 µs+9.5 x N	0.9 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	35 µs + 10.4 µs x N	0.9 µs
LDLBL	K	6.4 µs	1.3 µs
FILL	V: Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (bit)	29.4 µs + 8.0 µs x N 26.2 µs + 8.0 µs x N 55.1 µs + 8.0 µs x N	1.0 µs 1.0 µs 1.0 µs
FIND	V: Data Reg (N bits) V:Bit Reg. (N bits) K:Constant(N bits)	66.8 µs 66.8 µs 64.0 µs	1.0 µs 1.0 µs 1.0 µs

Table Instructions (cont.)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
FDGT	V: Data Reg (N bits) V:Bit Reg. (N bits) K:Constant(N bits)	66.1 µs 66.1 µs 55.2 µs	1.0 µs 1.0 µs 1.0 µs
FINDB	V: Data Reg (N bits) V:Bit Reg. (N bits) P:Indir. (Data) P:Indir. (Bit)	210.8 µs 210.8 µs 237.0 µs 237.0 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
TTD	V: Data Reg V:Bit Reg	66.9 µs 66.9 µs	1.0 µs 1.0 µs
RFB	V: Data Reg V:Bit Reg	66.8 µs 66.8 µs	1.0 µs 1.0 µs
STT	V: Data Reg V:Bit Reg K:Constant	67.8 µs 67.8 µs 65.0 µs	1.0 µs 1.0 µs 1.0 µs
RFT	V: Data Reg V:Bit Reg	51.1 µs 51.1 µs	1.0 µs 1.0 µs
ATT	V: Data Reg V:Bit Reg K:Constant	53.5 µs 53.5 µs 50.8 µs	1.0 µs 1.0 µs 1.0 µs
TSHFL	V: Data Reg V:Bit Reg	134.0 µs 134.0 µs	1.0 µs 1.0 µs
TSHFR	V: Data Reg V:Bit Reg	133.9 µs 133.9 µs	1.0 µs 1.0 µs
ANDMOV	V: Data Reg V:Bit Reg	80.2 µs 80.2 µs	1.0 µs 1.0 µs
ORMOV	V: Data Reg V:Bit Reg	80.4 µs 80.4 µs	1.0 µs 1.0 µs
XORMOV	V: Data Reg V:Bit Reg	80.4 µs 80.4 µs	1.0 µs 1.0 µs
SWAP	V: Data Reg V:Bit Reg	84.1 µs 84.1 µs	1.0 µs 1.0 µs
SETBIT	V: Data Reg (N bits) V:Bit Reg. (N bits)	59.5 µs 59.5 µs	1.0 µs 1.0 µs
RSTBIT	V: Data Reg (N bits) V:Bit Reg. (N bits)	59.5 µs 59.5 µs	1.0 µs 1.0 µs

CPU Control Instructions

CPU Control Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
NOP	None	1.1 µs	1.1 µs
END	None	24.0 µs	24.0 µs
STOP	None	10.0 µs	1.1 µs
RSTWT	None	5.9 µs	2.2 µs

Program Control Instructions

Program Control Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
GOTO	K	5.1 µs	4.8 µs
LBL	K	5.7 µs	0.0 µs
FOR	V, K	125.9 µs	14.5 µs
NEXT	None	64.4 µs	64.4 µs
GTS	K	27.5 µs	14.8 µs
SBR	K	1.5 µs	1.5 µs
RTC	None	25.7 µs	12.1 µs
RT	None	21.2 µs	21.2 µs
MLS	K	(1–7) 35.2 µs	35.2 µs
MLR	K	(0–7) 30.9 µs	30.9 µs

Interrupt Instructions

Interrupt Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
ENI	None	24.2 µs	2.7 µs
DISI	None	9.4 µs	2.3 µs
INT	O(0,1)	7.5 µs	–
IRTC	None	0.9 µs	1.3 µs
IRT	None	6.6 µs	–

Network Instructions

Network Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
RX	X, Y, C, T, CT, SP, S, $ V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	852.0 µs 852.0 µs 852.0 µs 868.2 µs 868.2 µs	4.4 µs 4.4 µs 4.4 µs 4.2 µs 4.2 µs
WX	X, Y, C, T, CT, SP, S, $ V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	1614.0 µs 1614.0 µs 1614.0 µs 1630.0 µs 1630.0 µs	4.4 µs 4.4 µs 4.4 µs 4.4 µs 4.4 µs

Intelligent I/O Instructions

Network Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
RD	V:Data Reg. V:Bit Reg.	385.7 µs 385.7 µs	1.2 µs 1.2 µs
WT	V:Data Reg. V:Bit Reg.	385.6 µs 385.6 µs	1.2 µs 1.2 µs

Message Instructions

Message Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
FAULT	V:Data Reg. V:Bit Reg. K:Constant	65.0 µs 65.0 µs 204.7 µs	4.4 µs 4.4 µs 4.4 µs
DLBL	K	–	–
NCON	K	–	–
ACON	A	–	–
PRINT	ASCII	631.0 µs	3.6 µs

RLL *plus* Instructions

RLL*plus* Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
ISG	S	44.0 µs	41.1 µs
SG	S	44.0 µs	41.1 µs
JMP	S	76.0 µs	9.3 µs
NJMP	S	77.4 µs	9.3 µs
CV	S	42.1 µs	27.5 µs
CVJMP	S	89.5 µs	17.6 µs
BCALL	C	22.1 µs	22.6 µs
BLK	C	17.1 µs	14.6 µs
BEND	None	8.7 µs	0.0 µs

Drum Instructions

Drum Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
DRUM	CT	840.0 µs	339.6 µs
EDRUM	CT	753.2 µs	357.0 µs
MDRMD	CT	411.3 µs	216.4 µs
MDRMW	CT	378.6 µs	147.0 µs

Clock / Calender Instructions

Clock / Calender Instructions		DL06	
Instruction		Execute	Not Execute
DATE	V:Data Reg. V:Bit Reg.	24.0 µs	1.2 µs
TIME	V:Data Reg. V:Bit Reg.	50.8 µs	1.2 µs

MODBUS Instructions

Clock / Calender Instructions		DL06	
Instruction		Execute	Not Execute
MRX	Input, Input Register Coil, Holding Register	120.2 µs	1.3 µs
MWX	Input, Input Register Coil, Holding Register	21.3 µs	1.3 µs

ASCII Instructions

ASCII Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
AIN	V	13.9 µs	12.0 µs
AFIND	V	111.5 µs	1.3 µs
AEX	V	111.7 µs	1.3 µs
CMPV	V	12.2 µs	1.3 µs
SWAPB	V	109.8 µs	1.3 µs
VPRINT	Text Data	161.6 µs	1.3 µs
PRINTV	V	163.3 µs	1.3 µs
ACRB	V	3.9 µs	1.1 µs

Special Relays

In This Appendix...

DL06 PLC Special Relays D–2

DL06 PLC Special Relays

"Special Relays" are just contacts that are set by the CPU operating system to indicate a particular system event has occurred. These contacts are available for use in your ladder program. Knowing just the right special relay contact to use for a particular situation can save a lot of programming time. Since the CPU operating system sets and clears special relay contacts, the ladder program only has to use them as inputs in ladder logic.

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 when the mode switch is in the run position and the CPU is running.
SP12	Terminal run mode	On when the mode switch is in the TERM position and the CPU is in the run mode.
SP13	Test run mode	On when the CPU is in the test run mode.
SP15	Test stop mode	On when the CPU is in the test stop mode.
SP16	Terminal PGM mode	On when the mode switch is in the TERM position and the CPU is in program mode.
SP17	Forced stop	On when the mode switch is in the STOP position.
SP20	Forced stop mode	On when the STOP instruction is executed.
SP22	Interrupt enabled	On when interrupts have been enabled using the ENI instruction.

System Monitoring

SP36	Override setup relay	On when the override function is used.
SP37	Scan controller	On when the actual scan time runs over the prescribed scan time.
SP40	Critical error	On when a critical error such as I/O communication loss has occurred.
SP41	Warning	On when a non critical error has occurred.
SP42	Diagnostics error	On when a diagnostics error or a system error occurs.
SP43	Low battery error	On when the CPU battery voltage is low.
SP44	Program memory error	On when a memory error such as a memory parity error has occurred.
SP45	I/O error	On when an I/O error such as a blown fuse occurs.
SP46	Communications error	On when a communication error occurs on any of the CPU ports.
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 will hold the exact error code.
SP53	Solve logic error	On if CPU cannot solve the logic.
SP54	Communication error	On when RX, WX, instructions are executed with the wrong parameters.
SP56	Table instruction overrun	On if a table instruction with a pointer is executed and the pointer value is outside the table boundary.

Accumulator Status

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	On when the 32 bit addition instruction results in a carry.
SP70	Sign	On anytime the value in the accumulator is negative.
SP71	Pointer reference error	On when the V-memory specified by a pointer (P) is not valid.
SP72	Floating point number	On anytime the value in the accumulator is a valid floating point number.
SP73	Overflow	On if overflow occurs in the accumulator when a signed addition or subtraction results in an incorrect sign bit.
SP74	Underflow	On anytime a floating point math operation results in an underflow error.
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.

HSIO Input Status

SP100	X0 status	On when X0 is on
SP101	X1 status	On when X1 is on

HSIO Pulse Output Relay

SP104	Profile Complete	On when the pulse output profile is completed. (Mode 30)

Communication Monitoring Relay

SP116	CPU port busy Port 2	On when port 2 is the master and sending data.
SP117	Communications error Port 2	On when port 2 is the master and has a communication error.

Option Slot Communication Monitoring Relay

SP120	Slot 1 busy	H0-ECOM/D0-DCM Port2
SP121	Slot 1 error	H0-ECOM/D0-DCM Port2
SP122	Slot 2 busy	H0-ECOM/D0-DCM Port2
SP123	Slot 2 error	H0-ECOM/D0-DCM Port2
SP124	Slot 3 busy	H0-ECOM/D0-DCM Port2
SP125	Slot 3 error	H0-ECOM/D0-DCM Port2
SP126	Slot 4 busy	H0-ECOM/D0-DCM Port2
SP127	Slot 4 error	H0-ECOM/D0-DCM Port2

Option Slot Special Relay

SP140-237	Slot 1	SP relay for option card
SP240-337	Slot 2	SP relay for option card
SP340-437	Slot 3	SP relay for option card
SP430-537	Slot 4	SP relay for option card

Counter 1 Mode 10 Equal Relays

SP540	Current = target value	On when the counter current value equals the value in V3631/3630
SP541	Current = target value	On when the counter current value equals the value in V3633/3632
SP542	Current = target value	On when the counter current value equals the value in V3635/3634
SP543	Current = target value	On when the counter current value equals the value in V3637/3636
SP544	Current = target value	On when the counter current value equals the value in V3641/3640
SP545	Current = target value	On when the counter current value equals the value in V3643/3642
SP546	Current = target value	On when the counter current value equals the value in V3645/3644
SP547	Current = target value	On when the counter current value equals the value in V3647/3646
SP550	Current = target value	On when the counter current value equals the value in V3651/3650
SP551	Current = target value	On when the counter current value equals the value in V3653/3652
SP552	Current = target value	On when the counter current value equals the value in V3655/3654
SP553	Current = target value	On when the counter current value equals the value in V3657/3656
SP554	Current = target value	On when the counter current value equals the value in V3661/3660
SP555	Current = target value	On when the counter current value equals the value in V3663/3662
SP556	Current = target value	On when the counter current value equals the value in V3665/3664
SP557	Current = target value	On when the counter current value equals the value in V3667/3666
SP560	Current = target value	On when the counter current value equals the value in V3671/3670
SP561	Current = target value	On when the counter current value equals the value in V3673/3672
SP562	Current = target value	On when the counter current value equals the value in V3675/3674
SP563	Current = target value	On when the counter current value equals the value in V3677/3676
SP564	Current = target value	On when the counter current value equals the value in V3771/3770
SP565	Current = target value	On when the counter current value equals the value in V3703/3702
SP566	Current = target value	On when the counter current value equals the value in V3705/3704
SP567	Current = target value	On when the counter current value equals the value in V3707/3706

Counter 2 Mode 10 Equal Relays

SP570	Current = target value	On when the counter current value equals the value in V3711/3710
SP571	Current = target value	On when the counter current value equals the value in V3713/3712
SP572	Current = target value	On when the counter current value equals the value in V3715/3714
SP573	Current = target value	On when the counter current value equals the value in V3717/3716
SP574	Current = target value	On when the counter current value equals the value in V3721/3720
SP575	Current = target value	On when the counter current value equals the value in V3723/3722
SP576	Current = target value	On when the counter current value equals the value in V3725/3724
SP577	Current = target value	On when the counter current value equals the value in V3727/3726
SP600	Current = target value	On when the counter current value equals the value in V3731/3730
SP601	Current = target value	On when the counter current value equals the value in V3733/3732
SP602	Current = target value	On when the counter current value equals the value in V3735/3734
SP603	Current = target value	On when the counter current value equals the value in V3737/3736
SP604	Current = target value	On when the counter current value equals the value in V3741/3740
SP605	Current = target value	On when the counter current value equals the value in V3743/3742
SP606	Current = target value	On when the counter current value equals the value in V3745/3744
SP607	Current = target value	On when the counter current value equals the value in V3747/3746
SP610	Current = target value	On when the counter current value equals the value in V3751/3750
SP611	Current = target value	On when the counter current value equals the value in V3753/3752
SP612	Current = target value	On when the counter current value equals the value in V3755/3754
SP613	Current = target value	On when the counter current value equals the value in V3757/3756
SP614	Current = target value	On when the counter current value equals the value in V3761/3760
SP615	Current = target value	On when the counter current value equals the value in V3763/3762
SP616	Current = target value	On when the counter current value equals the value in V3765/3764
SP617	Current = target value	On when the counter current value equals the value in V3767/3766

Product Weights

In this Appendix

Product Weight Table E–2

Product Weight Table

PLC	Weight
D0–06AR	1.78 lb./807g.
D0–06DR	1.76 lb./798 g.
D0–06DR-D	1.72 lb./780 g.
D0–06AA	1.78 lb./807 g.
D0–06DA	1.76 lb./798 g.
D0–06DD1	1.68 lb./762 g.
D0–06DD1-D	1.64 lb./743 g.
D0–06DD2-D	1.64 lb./743 g.
D0–06DD2	1.68 lb./798 g.
D0–06LCD	0.12 lb./54.4g.

PLC Memory

In this Appendix

DL06 PLC Memory F–2

DL06 PLC Memory

When designing a PLC application, it is important for the PLC user to understand the different types of memory in the PLC. Two types of memory are used by the DL06: CPU, RAM and EEPROM. This memory can be configured by the PLC user as either retentive or non-retentive memory.

Retentive memory is memory that is configured by the user to maintain values through a power cycle or a PROGRAM to RUN transition. Non-retentive memory is memory that is configured by the PLC user to clear data after a power cycle or a PROGRAM to RUN transition. The retentive ranges can be configured with either the handheld programmer using AUX57 or ***Direct***SOFT32 (PLC Setup).

The contents of RAM memory can be written to and read from an infinite number of times, but RAM requires a power source to maintain the contents of memory. The contents of RAM are maintained by the internal power supply (5VDC) only while the PLC is powered by an external source, normally 120VAC. When power to the PLC is turned off, the contents of RAM are maintained by a"Super-Capacitor". If the Super-Capacitor ever discharges, the contents of RAM will be lost. The data retention time of the Super-Capacitor backed RAM is 3 weeks maximum, and 4 1/2 days minimum (at 60° C).

The contents of EEPROM memory can be read from an infinite number of times, but there is a limit to the number of times it can be written to (typical specification is 100,000 writes). EEPROM does not require a power source to maintain the memory contents. It will retain the contents of memory indefinately.

PLC user V-memory is stored in both volatile RAM and non-volatile EEPROM memory. Data being stored in RAM uses V400-V677, V1200-V7377 and V10000-V17777. Data stored in EEPROM uses V7400-V7577 and V700-V777, V7600-V7777 and V36000-V37777.

Data values that must be retained for long periods of time, when the PLC is powered off, should be stored in EEPROM based V-memory.

Data values that are continually changing or which can be initialized with program logic should be stored in RAM based V-memory.

ASCII TABLE

In this Appendix

ASCII Conversion Table G–2

DECIMAL TO HEX TO ASCII CONVERTER											
DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII
0	00	NUL	32	20	space	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	“	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	‘	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	TAB	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

European Union Directives (CE)

In This Appendix...

European Union (EU) Directives H–2
Basic EMC Installation Guidelines H–4

European Union (EU) Directives

NOTE: The information contained in this section is intended as a guideline and is based on our interpretation of the various standards and requirements. Since the actual standards are issued by other parties, and in some cases governmental agencies, the requirements can change over time without advance warning or notice. Changes or additions to the standards can possibly invalidate any part of the information provided in this section.

This area of certification and approval is absolutely vital to anyone who wants to do business in Europe. One of the key tasks that faced the EU member countries and the European Economic Area (EEA) was the requirement to bring several similar yet distinct standards together into one common standard for all members. The primary purpose of a single standard was to make it easier to sell and transport goods between the various countries and to maintain a safe working and living environment. The Directives that resulted from this merging of standards are now legal requirements for doing business in Europe. Products that meet these Directives are required to have a CE mark to signify compliance.

Member Countries

As of July 23, 2002, the members of the EU are Austria, Belgium, Denmark, Finland, France, Germany, Greece, Ireland, Italy, Luxembourg, The Netherlands, Portugal, Spain, Sweden, and the United Kingdom. Iceland, Liechtenstein, and Norway together with the EU members make up the European Economic Area (EEA) and all are covered by the Directives.

Applicable Directives

There are several Directives that apply to our products. Directives may be amended, or added, as required.

- **Electromagnetic Compatibility Directive (EMC)** — this Directive attempts to ensure that devices, equipment, and systems have the ability to function satisfactorily in an electromagnetic environment without introducing intolerable electromagnetic disturbance to anything in that environment.
- **Machinery Safety Directive** — this Directive covers the safety aspects of the equipment, installation, etc. There are several areas involved, including testing standards covering both electrical noise immunity and noise generation.
- **Low Voltage Directive** — this Directive is also safety related and covers electrical equipment that has voltage ranges of 50–1000VAC and/or 75–1500VDC.
- **Battery Directive** — this Directive covers the production, recycling, and disposal of batteries.

Compliance

Certain standards within each Directive already require mandatory compliance. The EMC Directive, which has gained the most attention, became mandatory as of January 1, 1996. The Low Voltage Directive became mandatory as of January 1, 1997.

Ultimately, we are all responsible for our various pieces of the puzzle. As manufacturers, we must test our products and document any test results and/or installation procedures that are necessary to comply with the Directives. As a machine builder, you are responsible for installing the products in a manner which will ensure compliance is maintained. You are also responsible for testing any combinations of products that may (or may not) comply with the Directives when used together.

The end user of the products must comply with any Directives that may cover maintenance, disposal, etc. of equipment or various components. *Although we strive to provide the best assistance available, it is impossible for us to test all possible configurations of our products with respect to any specific Directive. Because of this, it is ultimately your responsibility to ensure that your machinery (as a whole) complies with these Directives and to keep up with applicable Directives and/or practices that are required for compliance.*

As of January 1, 1999, the DL05, DL06, DL205, DL305, and DL405 PLC systems manufactured by Koyo Electronics Industries or FACTS Engineering, when properly installed and used, conform to the Electromagnetic Compatibility (EMC), Low Voltage Directive, and Machinery Directive requirements of the following standards.

- **EMC Directive Standards Relevant to PLCs**
  EN50081–1 Generic emission standard for residential, commercial, and light industry
  EN50081–2 Generic emission standard for industrial environment.
  EN50082–1 Generic immunity standard for residential, commercial, and light industry
  EN50082–2 Generic immunity standard for industrial environment.
- **Low Voltage Directive Standards Applicable to PLCs**
  EN61010–1 Safety requirements for electrical equipment for measurement, control, and laboratory use.
- **Product Specific Standard for PLCs**
  EN61131–2 Programmable controllers, equipment requirements and tests. This standard replaces the above generic standards for immunity and safety. However, the generic emissions standards must still be used in conjunction with the following standards:
  -EN 61000-3-2 Harmonics
  -EN 61000-3-2 Fluctuations
- **Warning on Electrostatic Discharge (ESD)**
  We recommend that all personnel take necessay precautions to avoid the risk of transferring static charges to inside the control cabinet, and clear warnings and instructions should be provided on the cabinet exterior. Such precautions may include, the use of earth straps, simila devices or the powering off of the equipment inside the enclosure before the door is opened.
- **Warning on Radio Interference (RFI)**
  This is a class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.

General Safety

- External switches, circuit breaker or external fusing, are required for thes devices.
- The switch or circuit breaker should be mounted near the PLC equipment.

AutomationDirect is currently in the process of changing their testing procedures from the generic standards to the product specific standards.

Special Installation Manual

The installation requirements to comply with the requirements of the Machinery Directive, EMC Directive and Low Voltage Directive are slightly more complex than the normal installation requirements found in the United States. To help with this, we have published a special manual which you can order:

- **DA–EU–M** – EU Installation Manual that covers special installation requirements to meet the EU Directive requirements. Order this manual to obtain the most up-to-date information.

Other Sources of Information

Although the EMC Directive gets the most attention, other basic Directives, such as the Machinery Directive and the Low Voltage Directive, also place restrictions on the control panel builder. Because of these additional requirements it is recommended that the following publications be purchased and used as guidelines:

- BSI publication TH 42073: February 1996 – covers the safety and electrical aspects of the Machinery Directive
- EN 60204–1:1992 – General electrical requirements for machinery, including Low Voltage and EMC considerations
- IEC 1000–5–2: EMC earthing and cabling requirements
- IEC 1000–5–1: EMC general considerations

It may be possible for you to obtain this information locally; however, the official source of applicable Directives and related standards is:

The Office for Official Publications of the European Communities L–2985 Luxembourg; quickest contact is via the World Wide Web at http://euro–op.eu.int/indexn.htm

Another source is:

British Standards Institution – Sales Department

Linford Wood

Milton Keynes

MK14 6LE

United Kingdom; the quickest contact is via the World Wide Web at http://www.bsi.org.uk

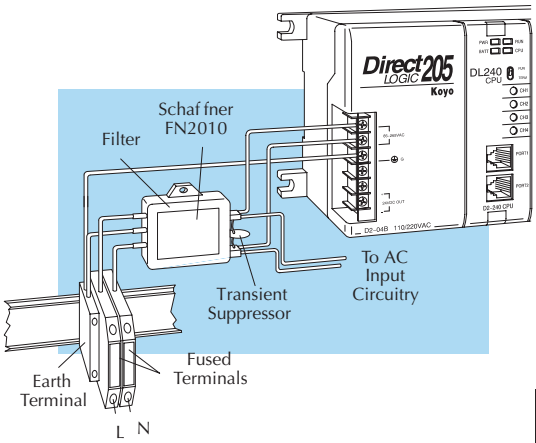
Basic EMC Installation Guidelines

Enclosures

The simplest way to meet the safety requirements of the Machinery and Low Voltage Directives is to house all control equipment in an industry standard lockable steel enclosure. This normally has an added benefit because it will also help ensure that the EMC characteristics are well within the requirements of the EMC Directive. Although the RF emissions from the PLC equipment, when measured in the open air, are well below the EMC Directive limits, certain configurations can increase emission levels. Holes in the enclosure, for the passage of cables or to mount operator interfaces, will often increase emissions.

AC Mains Filters

DL05, DL06, DL205 and DL305 AC powered base power supplies require extra mains filtering to comply with the EMC Directive on conducted RF emissions. All PLC equipment has been tested with filters from Schaffner, which reduce emissions levels if the filters are properly grounded (earth ground). A filter with a current rating suitable to supply all PLC power supplies and AC input modules should be selected. We suggest the FN2010 for DL05/DL06/DL205 systems and the FN2080 for DL305 systems. DL405 systems do not require extra filtering.



NOTE: Very few mains filters can reduce problem emissions to negligible levels. In some cases, filters may increase conducted emissions if not properly matched to the problem emissions.

Suppression and Fusing

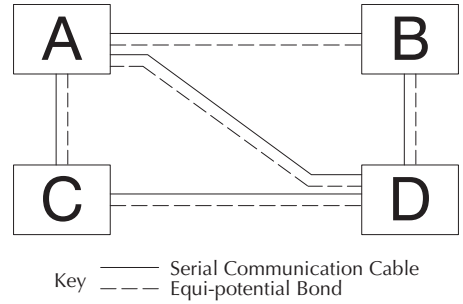
In order to comply with the fire risk requirements of the Low Voltage and Machinery Directive electrical standards EN 61010–1, and EN 60204–1, by limiting the power into "unlimited" mains circuits with power leads reversed, it is necessary to fuse both AC and DC supply inputs. You should also install a transient voltage suppressor across the power input connections of the PLC. Choose a suppressor such as a metal oxide varistor, with a rating of 275VAC working voltage for 230V nominal supplies (150VAC working voltage for 115V supplies) and high energy capacity (eg. 140 joules).

Transient suppressors must be protected by fuses and the capacity of the transient suppressor must be greater than the blow characteristics of the fuses or circuit breakers to avoid a fire risk. A recommended AC supply input arrangement for Koyo PLCs is to use twin 3 amp TT fused terminals with fuse blown indication, such as DINnectors DN–F10L terminals, or twin circuit breakers, wired to a Schaffner FN2010 filter or equivalent, with high energy transient suppressor soldered directly across the output terminals of the filter. PLC system inputs should also be protected from voltage impulses by deriving their power from the same fused, filtered, and surge-suppressed supply.

Internal Enclosure Grounding

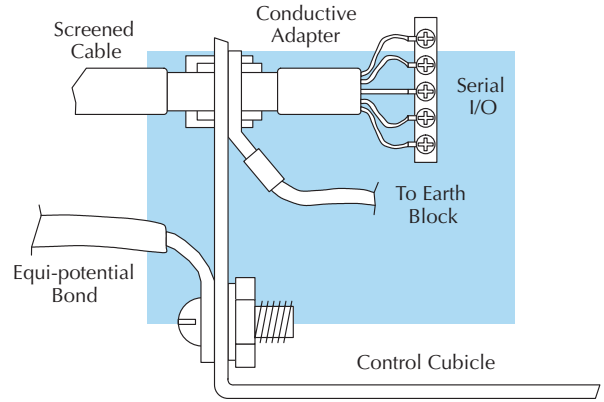
A heavy-duty star earth terminal block should be provided in every cubicle for the connection of all earth ground straps, protective earth ground connections, mains filter earth ground wires, and mechanical assembly earth ground connections. This should be installed to comply with safety and EMC requirements, local standards, and the requirements found in IEC 1000–5–2.The Machinery Directive also requires that the common terminals of PLC input modules, and common supply side of loads driven from PLC output modules should be connected to the protective earth ground terminal.

Equi–potential Grounding



Adequate site earth grounding must be provided for equipment containing modern electronic circuitry. The use of isolated earth electrodes for electronic systems is forbidden in some countries. Make sure you check any requirements for your particular destination. IEC 1000–5–2 covers equi-potential bonding of earth grids adequately, but special attention should be given to apparatus and control cubicles that contain I/O devices, remote I/O racks, or have inter-system communications with the primary PLC system enclosure. An equi-potential bond wire must be provided alongside all serial communications cables, and to any separate items of the plant which contain I/O devices connected to the PLC. The diagram shows an example of four physical locations connected by a communications cable.

Communications and Shielded Cables



Good quality 24 AWG minimum twisted-pair shielded cables, with overall foil and braid shields are recommended for analog cabling and communications cabling outside of the PLC enclosure. To date it has been a common practice to only provide an earth ground for one end of the cable shield in order to minimize the risk of noise caused by earth ground loop currents between apparatus. The procedure of only grounding one end, which primarily originated as a result of trying to reduce hum in audio systems, is no longer applicable to the complex industrial environment. Shielded cables are also efficient emitters of RF noise from the PLC system, and can interact in a parasitic manner in networks and between multiple sources of interference.

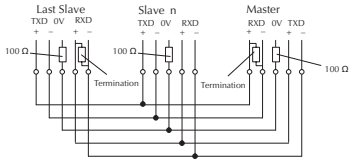
The recommendation is to use shielded cables as electrostatic "pipes" between apparatus and systems, and to run heavy gauge equi-potential bond wires alongside all shielded cables. When a shielded cable runs through the metallic wall of an enclosure or machine, it is recommended in IEC 1000–5–2 that the shield should be connected over its full perimeter to the wall, preferably using a conducting adapter, and not via a pigtail wire connection to an earth ground bolt. Shields must be connected to every enclosure wall or machine cover that they pass through.

Analog and RS232 Cables

Providing an earth ground for both ends of the shield for analog circuits provides the perfect electrical environment for the twisted pair cable as the loop consists of signal and return, in a perfectly balanced circuit arrangement, with connection to the common of the input circuitry made at the module terminals. RS232 cables are handled in the same way.

Multidrop Cables

RS422 twin twisted pair, and RS485 single twisted pair cables also require a 0V link, which has often been provided in the past by the cable shield. It is now recommended that you use triple twisted pair cabling for RS422 links, and twin twisted pair cable for RS485 links. This is because the extra pair can be used as the 0V inter-system link. With loop DC power supplies earth grounded in both systems, earth loops are created in this manner via the inter-system 0v link. The installation guides encourage earth loops, which are maintained at a low impedance by using heavy equi-potential bond wires. **To account for non–European installations using single-end earth grounds, and sites with far from ideal earth ground characteristics, we recommend the addition of 100 ohm resistors at each 0V link connection in network and communications cables.**



Shielded Cables within Enclosures

When you run cables between PLC items within an enclosure which also contains susceptible electronic equipment from other manufacturers, remember that these cables may be a source of RF emissions. There are ways to minimize this risk. Standard data cables connecting PLCs and/or operator interfaces should be routed well away from other equipment and their associated cabling. You can make special serial cables where the cable shield is connected to the enclosure's earth ground at both ends, the same way as external cables are connected.

Network Isolation

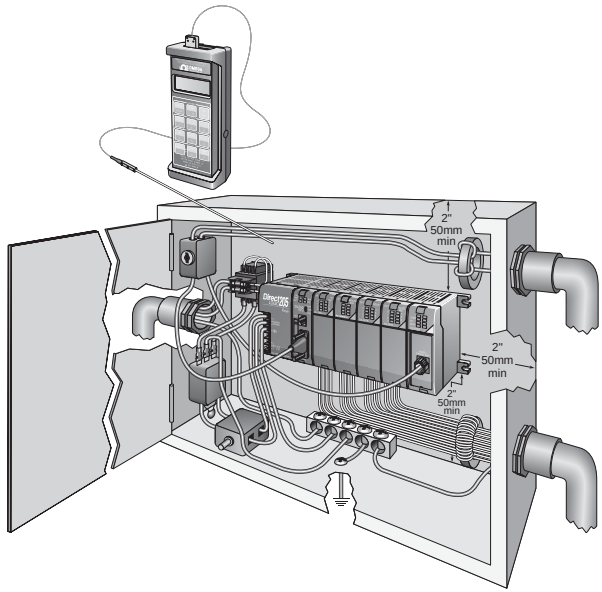
For safety reasons, it is a specific requirement of the Machinery Directive that a keyswitch must be provided that isolates any network input signal during maintenance, so that remote commands cannot be received that could result in the operation of the machinery. The FA–ISONET does not have a keyswitch! Use a keylock and switch on your enclosure which when open removes power from the FA–ISONET. To avoid the introduction of noise into the system, any keyswitch assembly should be housed in its own earth grounded steel box and the integrity of the shielded cable must be maintained.

Again, for further information on EU directives we recommend that you get a copy of our EU Installation Manual (DA–EU–M). Also, if you are connected to the World Wide Web, you can check the EU Commision's official site at: http://eur–op.eu.int/

DC Powered Versions

Due to slightly higher emissions radiated by the DC powered versions of the DL06, and the differing emissions performance for different DC supply voltages, the following stipulations must be met:

- The PLC must be housed within a metallic enclosure with a minimum amount of orifices.
- I/O and communications cabling exiting the cabinet must be contained within metallic conduit/trunking.



Items Specific to the DL06

- The rating between all circuits in this product are rated **as basic insulation only**, as appropriate for single fault conditions.
- There is no isolation offered between the PLC and the analog inputs of this product.
- It is the responsibility of the system designer to earth one side of all control and power circuits, and to earth the braid of screened cables.
- This equipment must be properly installed while adhering to the guidelines of the in house PLC

Manufacturer	Mfg. Part No.	ODmm	ID mm	L mm	1 Turn L/25 MHz	1 Turn L/100 MHz	2 Turn L/25 MHz	2 Turn L/100 MHz
RS Online	260–6795	17.5	9.5	28.5	153	210	649	632
Fair–Rite	2643665702	17.5	9.5	28.5	153	210	649	632
Wurth Elektronik	742 700 9	17.5	9.5	28.5	153	210	649	632

installation manual DA–EU–M, and the installation standards IEC 1000–5–1, IEC 1000–5–2 and IEC 1131–4.

- It is a requirement that all PLC equipment must be housed in a protective steel enclosure, which limits access to operators by a lock and power breaker. If access is required by operators or untrained personnel, the equipment must be installed inside an internal cover or secondary enclosure.
- It should be noted that the safety requirements of the machinery directive standard EN60204–1 state that all equipment power circuits must be wired through isolation transformers or isolating power supplies, and that one side of all AC or DC control circuits must be earthed.
- Both power input connections to the PLC must be separately fused using 3 amp T type anti–surge fuses, and a transient suppressor fitted to limit supply overvoltages.
- If the user is made aware by notice in the documentation that if the equipment is used in a manner not specified by the manufacturer the protection provided by the equipment may be impaired.

INDEX

A

Accumulating Fast Timer instruction, 5–42
Accumulating Timer (TMRA) instruction, 5–42
Accumulator, 5–69
Accumulator Instructions, 5–52
Add (ADD) instruction, 5–86
Add Binary Double instruction, 5–100
Add Binary instruction, 5–99
Add Binary Top of Stack instruction, 5–114
Add Double (ADDD) instruction, 5–87
Add Formatted instruction, 5–106
Add Real (ADDR) instruction, 5–88
Add to Top instruction, 5–162
Add Top of Stack instruction, 5–110
ADDR, 5–88
And (AND) instruction, 5–14, 5–31, 5–69
AND Bit-of-Word (ANDB) instruction, 5–15
And Double (ANDD) instruction, 5–70
And Formatted (ANDF) instruction, 5–71
And If Equal (ANDE) instruction, 5–28
And If Not Equal (ANDNE) instruction, 5–28
And Immediate (ANDI) instruction, 5–33
AND Move instruction, 5–167
And Negative Differential (ANDND) instruction, 5–22
And Not (ANDN) instruction, 5–14, 5–31
And Not Bit-of-Word (ANDNB) instruction, 5–15
And Not Immediate (ANDNI) instruction, 5–33
And Positive Differential (ANDPD) instruction, 5–22
And Store (AND STR) instruction, 5–16
And with Stack (ANDS) instruction, 5–72
Approvals, 2–9, F–2
Arc Cosine Real instruction, 5–119
Arc Sine Real instruction, 5–118
Arc Tangent Real instruction, 5–119
ASCII Constant instruction, 5–187
ASCII In/Out and PRINT, 4–50
ASCII Instructions, 5–207
ASCII Print from V–memory instruction, 5–223
ASCII Swap Bytes instruction, 5–224
ASCII to HEX instruction, 5–134
Automatic I/O Configuration, 4–43
Automatic Trapezoidal Profile, 3–47
Auxiliary Functions, 4–9, A–2

B

Battery Backup, 4–8
BCD instruction, 5–128
Binary Coded Decimal instruction, 5–128
Binary instruction, 5–127
Binary to Real Conversion instruction, 5–131
Bit Override, 9–19
Boolean Instructions, 5–5, 5–10

C

C Data Type, 4–26
Cables
  programming, 1–8
Changing Date and Time, 10–14
Comm Port 1, 4–4
Comm Port 2, 4–4
Comm Ports, configuring, 4–46
Communications Problems, 9–7
Comparative Boolean Instructions, 5–26
Compare (CMP) instruction, 5–81
Compare Double (CMPD) instruction, 5–82
Compare Formatted (CMPF) instruction, 5–83
Compare Real Number (CMPR) instruction, 5–85
Compare with Stack (CMPS) instruction, 5–84
Components, 1–6
Configuration, 4–43
Connecting DC I/O, 2–17
Connections
  power input, 1–8
  programming device, 1–8
  toggle switches, 1–7
Control Relay Bit Map, 4–35
Converge Jump instruction, 7–23
Converge Stage instruction, 7–23
Convergence Jump instruction, 7–20
Convergence Stages, 7–19
Cosine Real instruction, 5–118
Counter (CNT) instruction, 5–45
Counter Example Using Discrete Status Bits instruction, 5–46
Counter Status Bit Map, 4–37
CPU Features, 4–2
CPU Operation, 4–12
CPU Specifications, 4–3
CT Data type, 4–27

D

Data Label instruction, 5–187
Date and Time, 10–14
Date instruction, 5–171
DC input wiring, 2–22
DC output wiring, 2–23
Decode instruction, 5–126
Decrement Binary instruction, 5–105
Decrement instruction, 5–98
Degree Real Conversion instruction, 5–133
Diagnostics, 9–2
Dimensions, 2–6
DIN rail mounting, 2–8
DirectNET, 4–48
DirectNET Port Configuration, 4–49
Disable Interrupts instruction, 5–184
Discrete Inputs with Filter, 3–73
Divide Binary by Top OF Stack instruction, 5–117
Divide Binary instruction, 5–104
Divide by Top of Stack instruction, 5–113
Divide Double instruction, 5–96
Divide Formatted instruction, 5–109
Divide instruction, 5–95
Divide Real instruction, 5–97
Drum Instruction, 6–2, 6–12
  chart representation, 6–3
  counter assignments, 6–6
  drum control techniques, 6–10
  event drum (EDRUM), 6–14
  handheld programmer mnemonics, 6–16
  masked event drum (MDRMD), 6–19, 6–21
  overview, 6–8
  powerup state, 6–9
  self-resetting, 6–11

step transitions, 6–11
Drum sequencer programming, 1–11
Duplicate Reference Check, 9–13

E

Edits, 9–14
Electrical Noise, 9–10
Enable Interrupts instruction, 5–183
Encode instruction, 5–125
End instruction, 5–173, 9–12
END Statement, 5–5
Error Code Locations, 9–3
Error Codes, 9–4, 9–5, B–2
Errors, 9–2
European Union Directives, F–2
Exclusive Or (XOR) instruction, 5–77
Exclusive Or Double (XORD) instruction, 5–78
Exclusive Or Formatted (XORF) instruction, 5–79
Exclusive OR Move instruction, 5–167
Exclusive Or with Stack (XORS) instruction, 5–80
Execution times, C–3

F

Fault instruction, 5–186
Fill instruction, 5–146
Find Block instruction, 5–169
Find Greater Than instruction, 5–148
Find instruction, 5–147
For / Next instruction, 5–176
Force I/O, 4–14
Forcing I/O Points, 9–16
Front Panel, 2–4
Fuse protection, 2–10

G

Goto Label instruction, 5–175
Goto Subroutine instruction, 5–178
Gray Code instruction, 5–138

H

HEX to ASCII instruction, 5–135
High-Speed Counter, 3–7
High-speed I/O wiring, 2–24
High-Speed Interrupts, 3–64
HSIO Features, 3–2
HSIO Operating Mode, 3–4

I

I/O type selection, 1–5
Immediate Instructions, 5–32
Increment Binary instruction, 5–105
Increment instruction, 5–98
Indicators, 9–6
Inductive loads, 2–20
Initial Stage (ISG), 7–22
Initial Stages, 7–5
Instruction execution times, C–3
Instruction index, 5–2
Instructions, 5–2
drum, 6–2, 6–12
stage, 7–21
stage programming, 7–2
Instructions, by category
Accumulator / Stack Load, 5–52
Bit Operation, 5–120
Boolean, 5–10
Clock / Calendar, 5–171
Comparative, 5–26
CPU Control, 5–173

Immediate, 5–32
Instructions, by category, 5–175
Logical, 5–69
Math, 5–86
Number Conversion, 5–127
Table, 5–141
Timer, Counter and Shift Register, 5–39
Transcendental, 5–118
Interrupt instruction, 5–183
Interrupt Instructions, 5–183
Interrupt Return Conditional instruction, 5–183
Interrupt Return instruction, 5–183
Invert instruction, 5–129

I

J

Jump instruction, 7–7

L

LCD Display Panel, 10–2
LCD Installation, 10–3
LCD instruction, 5–197, 10–26
LCD Keypad, 10–2
LCD Menu Navigation, 10–5
Load (LD) instruction, 5–57
Load Accumulator Indexed (LDX) instruction, 5–61
Load Accumulator Indexed from Data Constants (LDSX) instruction, 5–62
Load Address (LDA) instruction, 5–60
Load Double (LDD) instruction, 5–58
Load Formatted (LDF) instruction, 5–59
Load Immediate (LDI) instruction, 5–37
Load Immediate Formatted (LDIF) instruction, 5–38
Load Label instruction, 5–142
Load Real Number (LDR) instruction, 5–63
Logical Instructions, 5–69

M

Maintenance, 9–2
Manual, documentation, 1–2
Master Line Reset instruction, 5–181
Master Line Set instruction, 5–181
Math Instructions, 5–86
Memory
EEPROM, 1–12
FLASH, 1–12
Memory Map, 4–25, 4–31
Message Instructions, 5–186
Mnemonics. See Instruction Mnemonics
MODBUS, 4–48, 5–201
MODBUS RTU Instructions, 5–201
Mode 10, 3–7
Mode 20, 3–24
Mode 30, 3–38
Mode 40, 3–64
Mode 50, 3–69
Mode 60, 3–73
Mode Switch, 4–6
Motion Control, 3–2
Mounting Guidelines, 2–6
Clearances, 2–7
Move instruction, 5–141
MRX instruction, 4–60
Multiply Binary instruction, 5–103
Multiply Binary Top of Stack instruction, 5–116
Multiply Double instruction, 5–93
Multiply Formatted instruction, 5–108
Multiply instruction, 5–92
Multiply Real instruction, 5–94
Multiply Top of Stack instruction, 5–112
MWX instruction, 4–60

N

Network Master, 4–56
Network Slave, 4–51
Network Specification, 4–47
No Operation instruction, 5–173
Noise, 9–10
Not (NOT) instruction, 5–19
Not Jump, 7–22
Numerical Constant instruction, 5–187

O

Or (OR) instruction, 5–12, 5–30, 5–73
Or Bit-of-Word (ORB) instruction, 5–13
Or Double (ORD) instruction, 5–74
Or Formatted (ORF) instruction, 5–75
Or If Equal (ORE) instruction, 5–27
Or If Not Equal (ORNE) instruction, 5–27
Or Immediate (ORI) instruction, 5–32
OR Move instruction, 5–167
Or Negative Differential (ORND) instruction, 5–21
Or Not (ORN) instruction, 5–12, 5–30
Or Not Bit-of-Word (ORNB) instruction, 5–13
Or Not Immediate (ORNI) instruction, 5–32
Or Out (OR OUT) instruction, 5–17
Or Out Immediate (OROUTI) instruction, 5–34
Or Positive Differential (ORPD) instruction, 5–21
Or Store (OR STR) instruction, 5–16
Or with Stack (ORS) instruction, 5–76
Out (OUT) instruction, 5–17, 5–64
Out Bit-of-Word (OUTB) instruction, 5–18
Out Double (OUTD) instruction, 5–64
Out Formatted (OUTF) instruction, 5–65
Out Immediate (OUTI) instruction, 5–34
Out Immediate Formatted (OUTIF) instruction, 5–35
Out Indexed (OUTX) instruction, 5–67
Out Least (OUTL) instruction, 5–68
Out Most (OUTM) instruction, 5–68

P

Parallel Processing, 7–19
Password, 4–11, 10–17
Pause (PAUSE) instruction, 5–25
PAUSE Instruction, 9–12
Pop (POP) instruction, 5–65
Port 1, 4–4
Port 2, 4–4
Positive Differential (PD) instruction, 5–19
Power Budgeting, 4–44
Power supply, 2–11
Print Message instruction, 5–189
Product weights, E–2
Program Mode, 4–13
Programming Devices, 2–14
Pulse Catch Input, 3–69
Pulse Output, 3–38

Q

Quick Start, 1–6

R

Radian Real Conversion instruction, 5–133
Read from Network instruction, 5–193
Real to Binary Conversion instruction, 5–132
Relay outputs, 2–19
Remote I/O Bit Map, 4–38
Remove from Bottom instruction, 5–153
Remove from Table instruction, 5–159
Reset (RST) instruction, 5–23

Reset Bit-of-Word (RST) instruction, 5–24
Reset Immediate (RSTI) instruction, 5–36
Reset Watch Dog Timer instruction, 5–174
Response Time, 4–17
Retentive Memory Ranges, 4–10
Rotate Left instruction, 5–123
Rotate Right instruction, 5–124
RSTBIT instruction, 5–144
RUN Indicator, 9–7
Run Mode, 4–13
Run Time Edits, 9–14

S

S Data type, 4–28
Safety, 2–2
Scan Time, 4–20
Segment instruction, 5–137
Set (SET) instruction, 5–23
Set Bit-of-Word (SET) instruction, 5–24
Set Immediate (SETI) instruction, 5–36
SETBIT instruction, 5–144
Shift Left instruction, 5–121
Shift Register (SR) instruction, 5–51
Shift Right instruction, 5–122
Shuffle Digits instruction, 5–139
Sine Real instruction, 5–118
Sinking / sourcing concepts, 2–15
Slot Numbering, 4–42
Source to Table instruction, 5–156
SP Data Type, 4–28
Special Instructions, 9–12
Special Relays, D–2
Special Relays, Error Codes, 9–3
Specifications, 2–26
Specifications, environmental, 2–9
Square Root Real instruction, 5–119
Stack Load and Output Data Instructions, 5–52
Stage Control / Status Bit Map, 4–33
Stage Counter (SGCNT) instruction, 5–47
Stage Counter instruction, 7–17
Stage instructions, 7–21
Stage Programming, 7–2, 7–15
  convergence, 7–19
  emergency stop, 7–14
  four steps to stage programmig, 7–9
  garage door opener example, 7–10
  initial stages, 7–5
  introduction, 7–2
  jump instruction, 7–7
  mutually exclusive transitions, 7–14
  parallel processes, 7–12
  parallel processing concepts, 7–19
  power flow transition, 7–18
  program organization, 7–15
  questions and answers, 7–27
  stage instruction characteristics, 7–6
  stage view, 7–18
  state transition diagrams, 7–3
  supervisory process, 7–17
  timer inside stage, 7–13
Startup, 9–11
State Diagram, 7–11
Status Indicators, 4–6
Step Transitions, 6–4
Step Trapezoidal Profile, 3–46
Stop instruction, 5–173, 9–12
Store (STR) instruction, 5–10, 5–29
Store Bit-of-Word (STRB) instruction, 5–11
Store If Equal (STRE) instruction, 5–26
Store If Not Equal (STRNE) instruction, 5–26
Store Immediate (STRI) instruction, 5–32

Store Negative Differential (STRND) instruction, 5–20
Store Not (STRN) instruction, 5–10, 5–29
Store Not Bit-of-Word (STRNB) instruction, 5–11
Store Not Immediate (STRNI) instruction, 5–32
Store Positive Differential (STRPD) instruction, 5–20
Subroutine Return Conditional instruction, 5–178
Subroutine Return instruction, 5–178
Subtract (SUB) instruction, 5–89
Subtract Binary Double instruction, 5–102
Subtract Binary instruction, 5–101
Subtract Binary Top of Stack instruction, 5–115
Subtract Double instruction, 5–90
Subtract Formatted instruction, 5–107
Subtract Real instruction, 5–91
Subtract Top of Stack instruction, 5–111
Sum instruction, 5–120
Swap instruction, 5–170
Syntax Check, 9–11
System Design, 1–10

T

T Data Type, 4–26
Table Shift Left instruction, 5–165
Table Shift Right, 5–165
Table to Destination instruction, 5–150
Tangent Real instruction, 5–118
Technical Support, 1–2
Ten's Complement instruction, 5–130
Terminal Block Removal, 2–5
Time instruction, 5–172
Timer (TMR) and Timer Fast (TMRF) instruction, 5–40
Timer Example Using Discrete Status Bits instruction, 5–41
Timer Status Bit Map, 4–37
Timer, Counter and Shift Register Instructions, 5–39
Troubleshooting, 9–8, 9–11

U

Understanding Master Control Relays instruction, 5–181
Up Down Counter (UDC) instruction, 5–49
Up/Down Counter, 3–24

V

V Data Type, 4–27
V-memory, 4–29

W

Web site, 1–2
Weight table, E–2
Wiring Diagrams, 2–26
  D0–06AA I/O Wiring Diagram, 2–26
  D0–06AR I/O Wiring Diagram, 2–28
  D0–06DA I/O Wiring Diagram, 2–30
  D0–06DD1 I/O Wiring Diagram, 2–32
  D0–06DD1–D I/O Wiring Diagram, 2–38
  D0–06DD2 I/O Wiring Diagram, 2–34
  D0-06DD2-D I/O Wiring Diagram 2-40
  D0–06DR I/O Wiring Diagram, 2–36
  D0–06DR–D I/O Wiring Diagram, 2–42
Wiring Guidelines, 2–10
Write to Network instruction, 5–195

X

X Data Type, 4–26
X Input / Y Output Bit Map, 4–32

Y

Y Data Type, 4–26