

# DL05 User Manual



---

# Manual Revisions

---

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

**Title:** DL05 Micro PLC User Manual

**Manual Number:** D0–USER–M

Edition/Rev	Date	Description of Changes
Original	12/98	original issue
2nd Edition	2/00	added pid chapter, analog module chapter, and memory cartridge chapter
2nd Edition, Rev. A	7/00	added DC power
3rd Edition	11/01	removed MC and analog module chapters, corrected drum instruction, several minor corrections, added PLC weights, EU directive additions
3rd Edition, Rev. A	7/02	Added new discrete option modules

# Table of Contents

---

## Chapter 1: Getting Started

<b>Introduction</b>	<b>1-2</b>
The Purpose of this Manual	1-2
Where to Begin	1-2
Supplemental Manuals	1-2
Technical Support	1-2
<b>Conventions Used</b>	<b>1-3</b>
Key Topics for Each Chapter	1-3
<b>DL05 Micro PLC Components</b>	<b>1-5</b>
The DL05 Micro PLC Family	1-5
<b>Programming Methods</b>	<b>1-5</b>
DirectSOFT Programming for Windows	1-5
Handheld Programmer	1-6
<b>I/O Selection Quick Chart</b>	<b>1-6</b>
<b>Quick Start for PLC Checkout and Programming</b>	<b>1-7</b>
Step 1: Unpack the DL05 Equipment	1-7
Step 2: Connect Switches to Input Terminals	1-8
Step 3: Connect the Power Wiring	1-9
Step 4: Connect the Programming Device	1-9
Step 5: Switch on the System Power	1-10
Step 6: Initialize Scratchpad Memory	1-10
Step 7: Enter a Ladder Program	1-10
<b>Steps to Designing a Successful System</b>	<b>1-11</b>
Step 1: Review the Installation Guidelines	1-11
Step 2: Understand the PLC Setup Procedures	1-11
Step 3: Review the I/O Selection Criteria	1-11
Step 4: Choose a System Wiring Strategy	1-11
Step 5: Understand the System Operation	1-11
Step 6: Review the Programming Concepts	1-12
Step 7: Choose the Instructions	1-12
Step 8: Understand the Maintenance and Troubleshooting Procedures	1-12
<b>Questions and Answers about DL05 Micro PLCs</b>	<b>1-13</b>

## Chapter 2: Installation, Wiring, and Specifications

<b>Safety Guidelines</b>	<b>2-2</b>
Plan for Safety	2-2
Three Levels of Protection	2-2
Orderly System Shutdown	2-3

---

System Power Disconnect .....	2-3
Emergency Stop .....	2-3
<b>Orientation to DL05 Front Panel .....</b>	<b>2-4</b>
Connector Removal .....	2-5
<b>Mounting Guidelines .....</b>	<b>2-6</b>
Unit Dimensions .....	2-6
Enclosures .....	2-6
Panel Layout & Clearances .....	2-7
Using Mounting Rails .....	2-8
Environmental Specifications .....	2-9
Agency Approvals .....	2-9
<b>Wiring Guidelines .....</b>	<b>2-10</b>
Fuse Protection for Input Power .....	2-10
External Power Source .....	2-11
Planning the Wiring Routes .....	2-11
Fuse Protection for Input and Output Circuits .....	2-12
I/O Point Numbering .....	2-12
<b>System Wiring Strategies .....</b>	<b>2-13</b>
PLC Isolation Boundaries .....	2-13
Connecting Operator Interface Devices .....	2-14
Connecting Programming Devices .....	2-14
Sinking / Sourcing Concepts .....	2-15
I/O “Common” Terminal Concepts .....	2-16
Connecting DC I/O to “Solid State” Field Devices .....	2-17
Solid State Input Sensors .....	2-17
Solid State Output Loads .....	2-17
Relay Output Wiring Methods .....	2-19
Surge Suppresion For Inductive Loads .....	2-20
Prolonging Relay Contact Life .....	2-21
DC Input Wiring Methods .....	2-22
DC Output Wiring Methods .....	2-23
High-Speed I/O Wiring Methods .....	2-24
<b>Glossary of Specification Terms .....</b>	<b>2-25</b>
<b>Wiring Diagrams and Specifications .....</b>	<b>2-26</b>
D0-05AR I/O Wiring Diagram .....	2-26
D0-05AR General Specifications .....	2-27
AC Input Specifications X0 – X7 .....	2-27
Relay Output Specifications Y0 – Y5 .....	2-27
D0-05DR I/O Wiring Diagram .....	2-28
D0-05DR General Specifications .....	2-29
DC Input Specifications .....	2-29
Relay Output Specifications .....	2-29
D0-05AD I/O Wiring Diagram .....	2-30
D0-05AD General Specifications .....	2-31
AC Input Specifications .....	2-31
DC Output Specifications .....	2-31
D0-05DD I/O Wiring Diagram .....	2-32

D0-05DD General Specifications .....	2-33
DC Input Specifications .....	2-33
DC Output Specifications .....	2-33
D0-05AA I/O Wiring Diagram .....	2-34
D0-05AA General Specifications .....	2-35
AC Input Specifications .....	2-35
AC Output Specifications .....	2-35
D0-05DA I/O Wiring Diagram .....	2-36
D0-05DA General Specifications .....	2-37
DC Input Specifications .....	2-37
AC Output Specifications .....	2-37
D0-05DR-D I/O Wiring Diagram .....	2-38
D0-05DR-D General Specifications .....	2-39
DC Input Specifications .....	2-39
Relay Output Specifications .....	2-39
D0-05DD-D I/O Wiring Diagram .....	2-40
D0-05DD-D General Specifications .....	2-41
DC Input Specifications .....	2-41
DC Output Specifications .....	2-41
<b>D0-16ND3 DC Input .....</b>	<b>2-43</b>
<b>D0-10TD1 DC Output .....</b>	<b>2-44</b>
<b>D0-16TD1 DC Output .....</b>	<b>2-45</b>
<b>D0-10TD2 DC Output .....</b>	<b>2-46</b>
<b>D0-16TD2 DC Output .....</b>	<b>2-47</b>
<b>D0-07CDR DC Input and Output .....</b>	<b>2-48</b>
<b>D0-08TR Relay Output .....</b>	<b>2-49</b>
<b>D0-08CDD1 DC Input and Output .....</b>	<b>2-50</b>
<b>I/O Addressing .....</b>	<b>2-51</b>
Module I/O Points and Addressing .....	2-51

## Chapter 3: High-Speed Input and Pulse Output Features

<b>Introduction .....</b>	<b>3-2</b>
Built-in Motion Control Solution .....	3-2
Availability of HSIO Features .....	3-2
Dedicated High-Speed I/O Circuit .....	3-3
Wiring Diagrams for Each HSIO Mode .....	3-3
<b>Choosing the HSIO Operating Mode .....</b>	<b>3-4</b>
Understanding the Six Modes .....	3-4
Default Mode .....	3-4
Configuring the HSIO Mode .....	3-5
Configuring Inputs X0 – X2 .....	3-5
<b>Mode 10: High-Speed Counter .....</b>	<b>3-6</b>
Purpose .....	3-6

Functional Block Diagram .....	3-6
Wiring Diagram .....	3-7
Interfacing to Counter Outputs .....	3-7
Setup for Mode 10 .....	3-8
Presets and Special Relays .....	3-8
Preset Data Starting Location .....	3-9
Using Fewer than 24 Presets .....	3-9
Equal Relay Numbers .....	3-9
Calculating Your Preset Values .....	3-10
X Input Configuration .....	3-10
Writing Your Control Program .....	3-11
Program Example: Counter Without Preset .....	3-12
Program Example Cont'd .....	3-13
Counter With Presets Program Example .....	3-14
Counter With Preload Program Example .....	3-16
Troubleshooting Guide for Mode 10 .....	3-17
<b>Mode 20: Quadrature Counter .....</b>	<b>3-18</b>
Purpose .....	3-18
Functional Block Diagram .....	3-18
Quadrature Encoder Signals .....	3-18
Wiring Diagram .....	3-19
Interfacing to Encoder Outputs .....	3-19
Setup for Mode 20 .....	3-20
X Input Configuration .....	3-20
Writing Your Control Program .....	3-21
Quadrature Counter w/Preload Program Example .....	3-21
Program Example Cont'd .....	3-22
Counter Preload Program Example .....	3-23
Troubleshooting Guide for Mode 20 .....	3-23
<b>Mode 30: Pulse Output .....</b>	<b>3-24</b>
Purpose .....	3-24
Functional Block Diagram .....	3-25
Wiring Diagram .....	3-26
Interfacing to Drive Inputs .....	3-26
Motion Profile Specifications .....	3-27
Physical I/O Configuration .....	3-27
Logical I/O Functions .....	3-27
Setup for Mode 30 .....	3-28
Profile / Velocity Select Register .....	3-28
Profile Parameter Table .....	3-29
Trapezoidal Profile .....	3-29
Registration Profile .....	3-29
Velocity Profile .....	3-29
Choosing the Profile Type .....	3-30
Trapezoidal Profile Defined .....	3-30
Registration and Home Search Profiles Defined .....	3-30
Velocity Profile Defined .....	3-30
Trapezoidal Profile Operation .....	3-31

Trapezoidal Profile Applications .....	3-31
Trapezoidal Profile Program Example .....	3-32
Program Example Cont'd .....	3-33
Preload Position Value .....	3-33
Registration Profile Operation .....	3-34
Registration Applications .....	3-34
Registration Profile Program Example .....	3-35
Program Example Cont'd .....	3-36
Home Search Program Example .....	3-37
Velocity Profile Operation .....	3-39
Velocity Profile Applications .....	3-39
Velocity Profile Program Example .....	3-40
Program Example Cont'd .....	3-41
Pulse Output Error Codes .....	3-42
Troubleshooting Guide for Mode 30 .....	3-42
<b>Mode 40: High-Speed Interrupts .....</b>	<b>3-44</b>
Purpose .....	3-44
Functional Block Diagram .....	3-44
Setup for Mode 40 .....	3-45
Interrupts and the Ladder Program .....	3-45
External Interrupt Timing Parameters .....	3-46
Timed Interrupt Parameters .....	3-46
X Input / Timed INT Configuration .....	3-46
Independent Timed Interrupt .....	3-46
External Interrupt Program Example .....	3-47
Timed Interrupt Program Example .....	3-48
<b>Mode 50: Pulse Catch Input .....</b>	<b>3-49</b>
Purpose .....	3-49
Functional Block Diagram .....	3-49
Pulse Catch Timing Parameters .....	3-49
Setup for Mode 50 .....	3-50
X Input Configuration .....	3-50
Pulse Catch Program Example .....	3-51
<b>Mode 60: Discrete Inputs with Filter .....</b>	<b>3-52</b>
Purpose .....	3-52
Functional Block Diagram .....	3-52
Input Filter Timing Parameters .....	3-52
Setup for Mode 60 .....	3-53
X Input Configuration .....	3-53
Filtered Inputs Program Example .....	3-54

## Chapter 4: CPU Specifications and Operation

<b>Introduction .....</b>	<b>4-2</b>
DL05 CPU Features .....	4-2
<b>CPU Specifications .....</b>	<b>4-3</b>
<b>CPU Hardware Setup .....</b>	<b>4-4</b>



Communication Port Pinout Diagrams .....	4-4
Connecting the Programming Devices .....	4-5
CPU Setup Information .....	4-5
Status Indicators .....	4-6
Mode Switch Functions .....	4-6
Changing Modes in the DL05 PLC .....	4-7
Mode of Operation at Power-up .....	4-7
Auxiliary Functions .....	4-8
Clearing an Existing Program .....	4-8
Initializing System Memory .....	4-8
Setting Retentive Memory Ranges .....	4-9
Using a Password .....	4-10
<b>CPU Operation .....</b>	<b>4-11</b>
CPU Operating System .....	4-11
Program Mode .....	4-12
Run Mode .....	4-12
Read Inputs .....	4-13
Service Peripherals and Force I/O .....	4-13
Update Special Relays and Special Registers .....	4-14
Solve Application Program .....	4-14
Write Outputs .....	4-15
Diagnostics .....	4-15
<b>I/O Response Time .....</b>	<b>4-15</b>
Is Timing Important for Your Application? .....	4-15
Normal Minimum I/O Response .....	4-15
Normal Maximum I/O Response .....	4-16
Improving Response Time .....	4-17
<b>CPU Scan Time Considerations .....</b>	<b>4-18</b>
Reading Inputs .....	4-18
Writing Outputs .....	4-18
Application Program Execution .....	4-19
<b>PLC Numbering Systems .....</b>	<b>4-20</b>
PLC Resources .....	4-20
V-Memory .....	4-21
Binary-Coded Decimal Numbers .....	4-21
Hexadecimal Numbers .....	4-21
<b>Memory Map .....</b>	<b>4-22</b>
Octal Numbering System .....	4-22
Discrete and Word Locations .....	4-22
V Memory Locations for Discrete Memory Areas .....	4-22
Input Points (X Data Type) .....	4-23
Output Points (Y Data Type) .....	4-23
Control Relays (C Data Type) .....	4-23
Timers and Timer Status Bits (T Data type) .....	4-23
Timer Current Values (V Data Type) .....	4-24
Counters and Counter Status Bits (CT Data type) .....	4-24
Counter Current Values (V Data Type) .....	4-24

Word Memory (V Data Type) .....	4-25
Stages (S Data type) .....	4-25
Special Relays (SP Data Type) .....	4-25
<b>DL05 System V-memory .....</b>	<b>4-26</b>
System Parameters and Default Data Locations (V Data Type) .....	4-26
DL05 Memory Map .....	4-28
<b>X Input Bit Map .....</b>	<b>4-29</b>
<b>Y Output Bit Map .....</b>	<b>4-29</b>
<b>Stage Control / Status Bit Map .....</b>	<b>4-29</b>
<b>Control Relay Bit Map .....</b>	<b>4-30</b>
<b>Timer Status Bit Map .....</b>	<b>4-31</b>
<b>Counter Status Bit Map .....</b>	<b>4-31</b>
<b>Network Configuration and Connections .....</b>	<b>4-32</b>
Configuring the DL05's Comm Ports .....	4-32
Networking DL05 to DL05 RS-232C .....	4-32
Networking PC to DL05s RS-422 .....	4-33
Networking DL05 Master to Other PLCs .....	4-33
MODBUS Port Configuration .....	4-34
DirectNET Port Configuration .....	4-35
<b>Network Slave Operation .....</b>	<b>4-36</b>
MODBUS Function Codes Supported .....	4-36
Determining the MODBUS Address .....	4-36
If Your Host Software Requires the Data Type and Address... ..	4-37
Example 1: V2100 .....	4-38
Example 2: Y20 .....	4-38
Example 3: T10 Current Value .....	4-38
Example 4: C54 .....	4-38
If Your MODBUS Host Software Requires an Address ONLY .....	4-39
Example 1: V2100 584/984 Mode .....	4-40
Example 2: Y20 584/984 Mode .....	4-40
Example 3: T10 Current Value 484 Mode .....	4-40
Example 4: C54 584/984 Mode .....	4-40
Determining the DirectNET Address .....	4-40
<b>Network Master Operation .....</b>	<b>4-41</b>
Step 1: Identify Master Port # and Slave # .....	4-42
Step 2: Load Number of Bytes to Transfer .....	4-42
Step 3: Specify Master Memory Area .....	4-43
Step 4: Specify Slave Memory Area .....	4-43
Communications from a Ladder Program .....	4-44
Multiple Read and Write Interlocks .....	4-44

## Chapter 5: Standard RLL Instructions

Introduction .....	5-2
--------------------	-----

<b>Using Boolean Instructions</b>	<b>5-4</b>
END Statement	5-4
Simple Rungs	5-4
Normally Closed Contact	5-4
Contacts in Series	5-5
Midline Outputs	5-5
Parallel Elements	5-5
Joining Series Branches in Parallel	5-6
Joining Parallel Branches in Series	5-6
Combination Networks	5-6
Comparative Boolean	5-6
Boolean Stack	5-7
Immediate Boolean	5-8
<b>Boolean Instructions</b>	<b>5-9</b>
Store (STR)	5-9
Store Not (STRN)	5-9
Or (OR)	5-10
Or Not (ORN)	5-10
And (AND)	5-11
And Not (ANDN)	5-11
And Store (AND STR)	5-12
Or Store (OR STR)	5-12
Out (OUT)	5-13
Or Out (OR OUT)	5-13
Not (NOT)	5-14
Positive Differential (PD)	5-14
Store Positive Differential (STRPD)	5-15
Store Negative Differential (STRND)	5-15
Or Positive Differential (ORPD)	5-16
Or Negative Differential (ORND)	5-16
And Positive Differential (ANDPD)	5-17
And Negative Differential (ANDND)	5-17
Set (SET)	5-18
Reset (RST)	5-18
Pause (PAUSE)	5-19
<b>Comparative Boolean</b>	<b>5-20</b>
Store If Equal (STRE)	5-20
Store If Not Equal (STRNE)	5-20
Or If Equal (ORE)	5-21
Or If Not Equal (ORNE)	5-21
And If Equal (ANDE)	5-22
And If Not Equal (ANDNE)	5-22
Store (STR)	5-23
Store Not (STRN)	5-23
Or (OR)	5-24
Or Not (ORN)	5-24
And (AND)	5-25
And Not (ANDN)	5-25

---

<b>Immediate Instructions</b> .....	<b>5-26</b>
Store Immediate (STRI) .....	5-26
Store Not Immediate (STRNI) .....	5-26
Or Immediate (ORI) .....	5-26
Or Not Immediate (ORNI) .....	5-26
OR Immediate Instructions Cont'd .....	5-27
And Immediate (ANDI) .....	5-27
And Not Immediate (ANDNI) .....	5-27
Out Immediate (OUTI) .....	5-28
Or Out Immediate (OROUTI) .....	5-28
Set Immediate (SETI) .....	5-29
Reset Immediate (RSTI) .....	5-29
<b>Timer, Counter and Shift Register Instructions</b> .....	<b>5-30</b>
Using Timers .....	5-30
Timer (TMR) and Timer Fast (TMRF) .....	5-31
Timer Example Using Discrete Status Bits .....	5-32
Timer Example Using Comparative Contacts .....	5-32
Accumulating Timer (TMRA) Accumulating Fast Timer (TMRAF) .....	5-33
Accumulating Timer Example using Discrete Status Bits .....	5-34
Accumulator Timer Example Using Comparative Contacts .....	5-34
Using Counters .....	5-35
Counter (CNT) .....	5-36
Counter Example Using Discrete Status Bits .....	5-37
Counter Example Using Comparative Contacts .....	5-37
Stage Counter (SGCNT) .....	5-38
Stage Counter Example Using Discrete Status Bits .....	5-39
Stage Counter Example Using Comparative Contacts .....	5-39
Up Down Counter (UDC) .....	5-40
Up / Down Counter Example Using Discrete Status Bits .....	5-41
Up / Down Counter Example Using Comparative Contacts .....	5-41
Shift Register (SR) .....	5-42
<b>Accumulator / Stack Load and Output Data Instructions</b> .....	<b>5-43</b>
Using the Accumulator .....	5-43
Copying Data to the Accumulator .....	5-43
Changing the Accumulator Data .....	5-44
Using the Accumulator Stack .....	5-45
Using Pointers .....	5-46
Load (LD) .....	5-48
Load Double (LDD) .....	5-49
Load Formatted (LDF) .....	5-50
Load Address (LDA) .....	5-51
Out (OUT) .....	5-52
Out Double (OUTD) .....	5-52
Out Formatted (OUTF) .....	5-53
Pop (POP) .....	5-53
Pop Instruction Continued .....	5-54
<b>Logical Instructions (Accumulator)</b> .....	<b>5-55</b>
And (AND) .....	5-55

---

And Double (ANDD) .....	5-56
Or (OR) .....	5-57
Or Double (ORD) .....	5-58
Exclusive Or (XOR) .....	5-59
Exclusive Or Double (XORD) .....	5-60
Compare (CMP) .....	5-61
Compare Double (CMPD) .....	5-62
<b>Math Instructions .....</b>	<b>5-63</b>
Add (ADD) .....	5-63
Add Double (ADDD) .....	5-64
Subtract (SUB) .....	5-65
Subtract Double (SUBD) .....	5-66
Multiply (MUL) .....	5-67
Multiply Double (MULD) .....	5-68
Divide (DIV) .....	5-69
Divide Double (DIVD) .....	5-70
Increment (INC) .....	5-71
Decrement (DEC) .....	5-71
Increment Binary (INCB) .....	5-72
Decrement Binary (DECB) .....	5-72
Add Binary (ADDB) .....	5-73
Subtract Binary (SUBB) .....	5-74
Multiply Binary (MULB) .....	5-75
Divide Binary (DIVB) .....	5-76
<b>Bit Operation Instructions .....</b>	<b>5-77</b>
Sum (SUM) .....	5-77
Shift Left (SHFL) .....	5-77
Shift Right (SHFR) .....	5-79
Encode (ENCO) .....	5-80
Decode (DECO) .....	5-81
<b>Number Conversion Instructions (Accumulator) .....</b>	<b>5-82</b>
Binary (BIN) .....	5-82
Binary Coded Decimal (BCD) .....	5-83
Invert (INV) .....	5-84
ASCII to HEX (ATH) .....	5-85
HEX to ASCII (HTA) .....	5-86
Gray Code (GRAY) .....	5-88
Shuffle Digits (SFLDGT) .....	5-89
Shuffle Digits Block Diagram .....	5-89
<b>Table Instructions .....</b>	<b>5-91</b>
Move (MOV) .....	5-91
Move Memory Cartridge / Load Label (MOVMC), (LDLBL) .....	5-92
Copy Data From a Data Label Area to V Memory .....	5-93
<b>CPU Control Instructions .....</b>	<b>5-94</b>
No Operation (NOP) .....	5-94
End (END) .....	5-94
Stop (STOP) .....	5-94

Reset Watch Dog Timer (RSTWT) .....	5-95
<b>Program Control Instructions .....</b>	<b>5-96</b>
For / Next (FOR) (NEXT) .....	5-96
Goto Subroutine (GTS) (SBR) .....	5-98
Subroutine Return (RT) .....	5-98
Subroutine Return Conditional (RTC) .....	5-98
Master Line Set (MLS) .....	5-101
Master Line Reset (MLR) .....	5-101
Understanding Master Control Relays .....	5-101
MLS/MLR Example .....	5-102
<b>Interrupt Instructions .....</b>	<b>5-103</b>
Interrupt (INT) .....	5-103
Interrupt Return (IRT) .....	5-103
Interrupt Return Conditional (IRTC) .....	5-103
Enable Interrupts (ENI) .....	5-103
Disable Interrupts (DISI) .....	5-104
External Interrupt Program Example .....	5-104
Timed Interrupt Program Example .....	5-105
Independent Timed Interrupt .....	5-105
<b>Message Instructions .....</b>	<b>5-106</b>
Fault (FAULT) .....	5-106
Fault Example .....	5-106
Data Label (DLBL) .....	5-107
ASCII Constant (ACON) .....	5-107
Numerical Constant (NCON) .....	5-107
Data Label Example .....	5-108
Print Message (PRINT) .....	5-109
<b>Network Instructions .....</b>	<b>5-113</b>
Read from Network (RX) .....	5-113
Write to Network (WX) .....	5-115

## Chapter 6: Drum Instruction Programming

<b>Introduction .....</b>	<b>6-2</b>
Purpose .....	6-2
Drum Terminology .....	6-2
Drum Chart Representation .....	6-3
Output Sequences .....	6-3
<b>Step Transitions .....</b>	<b>6-4</b>
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
<b>Overview of Drum Operation .....</b>	<b>6-8</b>

Drum Instruction Block Diagram .....	6-8
Powerup State of Drum Registers .....	6-9
<b>Drum Control Techniques .....</b>	<b>6-10</b>
Drum Control Inputs .....	6-10
Self-Resetting Drum .....	6-11
Initializing Drum Outputs .....	6-11
Using Complex Event Step Transitions .....	6-11
<b>Drum Instruction .....</b>	<b>6-12</b>
Timed Drum with Discrete Outputs (DRUM) .....	6-12
Event Drum (EDRUM) .....	6-14
Handheld Programmer Drum Mnemonics .....	6-16

## Chapter 7: RLLPLUS Stage Programming

<b>Introduction to Stage Programming .....</b>	<b>7-2</b>
Overcoming “Stage Fright” .....	7-2
<b>Learning to Draw State Transition Diagrams .....</b>	<b>7-3</b>
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
<b>Using the Stage Jump Instruction for State Transitions .....</b>	<b>7-7</b>
Stage Jump, Set, and Reset Instructions .....	7-7
<b>Stage Program Example: Toggle On/Off Lamp Controller .....</b>	<b>7-8</b>
A 4-State Process .....	7-8
<b>Four Steps to Writing a Stage Program .....</b>	<b>7-9</b>
<b>Stage Program Example: A Garage Door Opener .....</b>	<b>7-10</b>
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
<b>Stage Program Design Considerations .....</b>	<b>7-15</b>
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 DirectSOFT .....	7-18
<b>Parallel Processing Concepts .....</b>	<b>7-19</b>
Parallel Processes .....	7-19
Converging Processes .....	7-19
Convergence Stages (CV) .....	7-19
Convergence Jump (CVJMP) .....	7-20
Convergence Stage Guidelines .....	7-20
<b>RLLPLUS (Stage) Instructions .....</b>	<b>7-21</b>
Staget (SG) .....	7-21
Initial Staget (ISG) .....	7-22
JUMP (JMP) .....	7-22
Not Jump (NJMP) .....	7-22
Converge Stage (CV) and Converge Jump (CVJMP) .....	7-23
<b>Questions and Answers about Stage Programming .....</b>	<b>7-25</b>

## Chapter 8: PID Loop Operation

<b>DL05 PID Loop Features .....</b>	<b>8-2</b>
Main Features .....	8-2
The Basics of PID Loops .....	8-4
<b>Loop Setup Parameters .....</b>	<b>8-6</b>
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
<b>Loop Sample Rate and Scheduling .....</b>	<b>8-13</b>
Loop Sample Rates Addr + 07 .....	8-13
Choosing the Best Sample Rate .....	8-13
Programming the Sample Rate .....	8-14
PID Loop Effect on CPU Scan Time .....	8-15
<b>Ten Steps to Successful Process Control .....</b>	<b>8-17</b>
Step 1: Know the Recipe .....	8-17
Step 2: Plan Loop Control Strategy .....	8-17
Step 3: Size and Scale Loop Components .....	8-17
Step 4: Select I/O Modules .....	8-17
Step 5: Wiring and Installation .....	8-18
Step 6: Loop Parameters .....	8-18
Step 7: Check Open Loop Performance .....	8-18
Step 8: Loop Tuning .....	8-18



Step 9: Run Process Cycle .....	8-18
Step 10: Save Parameters .....	8-18
<b>Basic Loop Operation .....</b>	<b>8-19</b>
Data Locations .....	8-19
Data Sources .....	8-19
Direct Access to Analog I/O .....	8-20
Loop Modes .....	8-21
CPU Modes and Loop Modes .....	8-22
How to Change Loop Modes .....	8-23
Operator Panel Control of PID Modes .....	8-24
PLC Modes' Effect on Loop Modes .....	8-24
Loop Mode Override .....	8-24
Bumpless Transfers .....	8-25
<b>PID Loop Data Configuration .....</b>	<b>8-26</b>
Loop Parameter Data Formats .....	8-26
Choosing Unipolar or Bipolar Format .....	8-26
Handling Data Offsets .....	8-27
Setpoint (SP) Limits .....	8-27
Remote Setpoint (SP) Location .....	8-28
Process Variable (PV) Configuration .....	8-28
Control Output Configuration .....	8-29
Error Term Configuration .....	8-30
<b>PID Algorithms .....</b>	<b>8-31</b>
Position Algorithm .....	8-31
Velocity Algorithm .....	8-32
Direct-Acting and Reverse-Acting Loops .....	8-33
P-I-D Loop Terms .....	8-34
Using a Subset of PID Control .....	8-35
Derivative Gain Limiting .....	8-36
Bias Term .....	8-36
Bias Freeze .....	8-37
<b>Loop Tuning Procedure .....</b>	<b>8-38</b>
Open-Loop Test .....	8-38
Manual Tuning Procedure .....	8-39
Auto Tuning Procedure .....	8-40
Tuning Cascaded Loops .....	8-44
<b>PV Analog Filter .....</b>	<b>8-45</b>
The DL05 Built-in Analog Filter .....	8-45
Creating an Analog Filter in Ladder Logic .....	8-46
<b>Feedforward Control .....</b>	<b>8-47</b>
Feedforward Example .....	8-48
<b>Time-Proportioning Control .....</b>	<b>8-49</b>
On/Off Control Program Example .....	8-50
<b>Cascade Control .....</b>	<b>8-51</b>
Introduction .....	8-51
Cascaded Loops in the DL05 CPU .....	8-52

<b>Process Alarms</b> .....	<b>8-53</b>
PV Absolute Value Alarms .....	8-54
PV Deviation Alarms .....	8-54
PV Rate-of-Change Alarm .....	8-55
PV Alarm Hysteresis .....	8-56
Alarm Programing Error .....	8-56
<b>Ramp/Soak Generator</b> .....	<b>8-57</b>
Introduction .....	8-57
Ramp/Soak Table .....	8-58
Ramp/Soak Table Flags .....	8-60
Ramp/Soak Generator Enable .....	8-60
Ramp/Soak Controls .....	8-60
Ramp/Soak Profile Monitoring .....	8-61
Ramp/Soak Programming Errors .....	8-61
Testing Your Ramp/Soak Profile .....	8-61
<b>Troubleshooting Tips</b> .....	<b>8-62</b>
<b>Bibliography</b> .....	<b>8-63</b>
<b>Glossary of PID Loop Terminology</b> .....	<b>8-64</b>

## Chapter 9: Maintenance and Troubleshooting

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

## Appendix A: Auxiliary Functions

<b>Introduction</b> .....	<b>A-2</b>
Purpose of Auxiliary Functions .....	A-2
Accessing AUX Functions via DirectSOFT .....	A-3
Accessing AUX Functions via the Handheld Programmer .....	A-3
<b>AUX 2* — RLL Operations</b> .....	<b>A-4</b>
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
<b>AUX 3* — V-memory Operations</b> .....	<b>A-4</b>
AUX 31 Clear V Memory .....	A-4
<b>AUX 4* — I/O Configuration</b> .....	<b>A-4</b>

AUX 41 Show I/O Configuration .....	A-4
<b>AUX 5* — CPU Configuration .....</b>	<b>A-5</b>
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
<b>AUX 6* — Handheld Programmer Configuration .....</b>	<b>A-8</b>
AUX 61 Show Revision Numbers .....	A-8
AUX 62 Beeper On/Off .....	A-8
AUX 65 Run Self Diagnostics .....	A-8
<b>AUX 7* — EEPROM Operations .....</b>	<b>A-8</b>
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
<b>AUX 8* — Password Operations .....</b>	<b>A-9</b>
AUX 81 Modify Password .....	A-9
AUX 82 Unlock CPU .....	A-10
AUX 83 Lock CPU .....	A-10

## Appendix B: DL05 Error Codes

## Appendix C: Instruction Execution Times

<b>Introduction .....</b>	<b>C-2</b>
V-Memory Data Registers .....	C-2
V-Memory Bit Registers .....	C-2
How to Read the Tables .....	C-2
<b>Instruction Execution Times .....</b>	<b>C-3</b>
Boolean Instructions .....	C-3
Comparative Boolean Instructions .....	C-4
Immediate Instructions .....	C-10
Timer, Counter, and Shift Register .....	C-10
Accumulator Data Instructions .....	C-11
Logical Instructions .....	C-12
Math Instructions .....	C-12

Bit Instructions .....	C-14
Number Conversion Instructions .....	C-14
Table Instructions .....	C-14
CPU Control Instructions .....	C-15
Program Control Instructions .....	C-15
Interrupt Instructions .....	C-15
Network Instructions .....	C-15
Message Instructions .....	C-16
RLLPLUS Instructions .....	C-16
Drum Instructions .....	C-16

## Appendix D: Special Relays

<b>DL05 PLC Special Relays .....</b>	<b>D-2</b>
Startup and Real-Time Relays .....	D-2
CPU Status Relays .....	D-2
System Monitoring .....	D-3
Accumulator Status .....	D-3
HSIO Pulse Output Relay .....	D-4
Communication Monitoring Relays .....	D-4
Equal Relays for HSIO Mode 10 Counter Presets .....	D-4

## Appendix E: DL05 Product Weights

<b>Product Weight Table .....</b>	<b>E-2</b>
-----------------------------------	------------

## Appendix F: European Union Directives (CE)

<b>European Union (EU) Directives .....</b>	<b>F-2</b>
Member Countries .....	F-2
Special Installation Manual .....	F-3
Other Sources of Information .....	F-4
<b>Basic EMC Installation Guidelines .....</b>	<b>F-4</b>
Enclosures .....	F-4
Suppression and Fusing .....	F-5
Internal Enclosure Grounding .....	F-5
Equip-potential Grounding .....	F-6
Communications and Shielded Cables .....	F-6
Analog and RS232 Cables .....	F-7
Multidrop Cables .....	F-7
Shielded Cables .....	F-7
within Enclosures .....	F-7
Network Isolation .....	F-7
DC Powered Versions .....	F-8
Items Specific to the DL 05 .....	F-9

# Getting Started

---

## In This Chapter. . . .

- Introduction
  - Conventions Used
  - DL05 Micro PLC Components
  - Programming Methods
  - I/O Selection Quick Chart
  - Quick Start for PLC Checkout and Programming
  - Steps to Designing a Successful System
  - Questions and Answers about DL05 Micro PLCs
-

## Introduction

### The Purpose of this Manual

Thank you for purchasing a DL05 Micro PLC. This manual shows you how to install, program, and maintain all the Micro PLCs in the DL05 family. It also helps you understand how to interface them to other devices in a control system. This manual contains important information for personnel who will install DL05 PLCs, and for the PLC programmer. If you understand PLC systems our manuals will provide all the information you need to get and keep your system up and running.

### Where to Begin

If you already understand the DL05 Micro PLC please read Chapter 2, "Installation, Wiring, and Specifications", and proceed on to other chapters as needed. Be sure to keep this manual handy for reference when you run into questions. If you are a new DL05 customer, we suggest you read this manual completely so you can understand the wide variety of features in the DL05 family of products. We believe you will be pleasantly surprised with how much you can accomplish with our products.

### Supplemental Manuals

The ***D0-OPTIONS-M*** manual will be most helpful to select and use any of the optional modules that are available for the DL05 PLC which includes the analog I/O modules. If you have purchased operator interfaces or ***DirectSOFT™***, you will need to supplement this manual with the manuals that are written for these products.

### Technical Support

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

- **Table of Contents** – chapter and section listing of contents, in the front of this manual
- **Appendices** – reference material for key topics, near the end of this manual

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

- **Internet** – the address of our website is:  
**In Brazil:** <http://www.soliton.com.br>

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

## Conventions Used



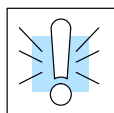
When you see the “light bulb” icon in the left-hand margin, the paragraph to its immediate right will give you a **special tip**.

The word **TIP:** in boldface will mark the beginning of the text.



When you see the “notepad” icon in the left-hand margin, the paragraph to its immediate right will be a **special note**.

The word **NOTE:** in boldface will mark the beginning of the text.

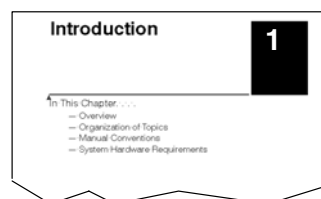


When you see the “exclamation mark” icon in the left-hand margin, the paragraph to its immediate right will be a **warning**. This information could prevent injury, loss of property, or even death (in extreme cases).

The word **WARNING:** in boldface will mark the beginning of the text.

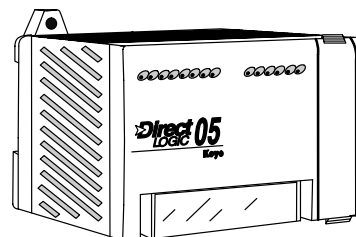
### Key Topics for Each Chapter

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



## DL05 Micro PLC Components

The DL05 Micro PLC family is a versatile product line that provides a wide variety of features in a very compact footprint. The PLCs are small, yet offer many features usually found in larger, more expensive systems. These include a removeable connector, and two RS-232C communication ports.



### The DL05 Micro PLC Family

The DL05 Micro PLC family includes eight different versions. All have the same appearance and CPU performance. The CPU offers the same instruction set as our popular DL240 CPU, plus several more instructions specifically designed for machine control applications. All DL05 PLCs have two RS-232C communications ports. Units with DC inputs have selectable high-speed input features on three input points. Units with DC outputs offer selectable pulse output capability on the first and second output points. All DL05 Micro PLCs offer a large amount of program memory, a substantial instruction set and advanced diagnostics. Details of these features and more are covered in Chapter 4, CPU Specifications and Operation. The eight types of DL05 Micro PLCs provide a variety of Input/Output choices, listed in the following table.

DL05 Part Number	Discrete Input Type	Discrete Output Type	External Power	High-Speed Input	Pulse Output
D0-05AR	AC	Relay	95-240 VAC	No	No
D0-05DR	DC	Relay	95-240 VAC	Yes	No
D0-05AD	AC	DC	95-240 VAC	No	Yes
D0-05DD	DC	DC	95-240 VAC	Yes	Yes
D0-05AA	AC	AC	95-240 VAC	No	No
D0-05DA	DC	AC	95-240 VAC	Yes	No
D0-05DR-D	DC	Relay	12-24 VDC	Yes	No
D0-05DD-D	DC	DC	12-24 VDC	Yes	Yes

## Programming Methods

### DirectSOFT Programming for Windows™

Two programming methods are available: RLL (Relay Ladder Logic) and RLL<sup>PLUS</sup>. RLL<sup>PLUS</sup> combines the added feature of flow chart programming (Stage™) to the standard RLL language. Both the **DirectSOFT™** programming package and the handheld programmer support RLL<sup>PLUS</sup> as well as standard RLL instructions.

The DL05 Micro PLC can be programmed with one of the most advanced programming packages in the industry — **DirectSOFT**, a Windows-based software package that supports familiar features such as cut-and-paste between applications, point-and-click editing, viewing and editing multiple application programs at the same time, etc.



**Direct**SOFT universally supports the **Direct**LOGIC CPU families. This means you can use the full version of **Direct**SOFT to program DL05, DL105, DL205, DL305, DL405 or any new CPUs we may add to our product line. (Upgrade software may be required for new CPUs as they become available.). A separate manual discusses **Direct**SOFT programming software. **Direct**SOFT version 2.4 or later is needed to program the DL05.

### Handheld Programmer

All DL05 Micro PLCs have built-in programming ports for use with the handheld programmer (D2–HPP), the same programmer used with the DL105 and DL205 families. The handheld programmer can be used to create, modify and debug your application program. A separate manual discusses the Handheld Programmer. Only D2–HPPs with firmware version 1.09 or later will program the DL05.

## I/O Selection Quick Chart

The eight versions of the DL05 have Input/Output circuits which can interface to a wide variety of field devices. In several instances a particular Input or Output circuit can interface to either DC or AC voltages, or both sinking and sourcing circuit arrangements. Check this chart carefully to find the proper DL05 Micro PLC to interface to the field devices in your application.

DL05 Part Number	INPUTS			OUTPUTS		
	I/O type / commons	Sink / Source	Voltage Ranges	I/O type / commons	Sink / Source	Voltage / Current Ratings
D0–05AR	AC / 2	–	90 – 120 VAC	Relay / 2	Sink or Source	6 – 27 VDC, 2A * 6 – 240 VAC, 2A *
D0–05DR	DC / 2	Sink or Source	12 – 24 VDC	Relay / 2	Sink or Source	6 – 27 VDC, 2A * 6 – 240 VAC, 2A *
D0–05AD	AC / 2	–	90 – 120 VAC	DC / 1	Sink	6 – 27 VDC, 0.5A (Y0–Y2) 6 – 27 VDC, 1.0A (Y3–Y5)
D0–05DD	DC / 2	Sink or Source	12 – 24 VDC	DC / 1	Sink	6 – 27 VDC, 0.5A (Y0–Y2) 6 – 27 VDC, 1.0A (Y3–Y5)
D0–05AA	AC / 2	–	90 – 120 VAC	AC / 2	–	17 – 240 VAC, 47 – 63 Hz 0.5A *
D0–05DA	DC / 2	Sink or Source	12 – 24 VDC	AC / 2	–	17 – 240 VAC, 47 – 63 Hz 0.5A *
D0–05DR–D	DC / 2	Sink or Source	12 – 24 VDC	Relay / 2	Sink or Source	6 – 27 VDC, 2A * 6 – 240 VAC, 2A *
D0–05DD–D	DC / 2	Sink or Source	12 – 24 VDC	DC / 1	Sink	6 – 27 VDC, 0.5A (Y0–Y2) 6 – 27 VDC, 1.0A (Y3–Y5)

\* See Chapter 2 Specifications for your particular DL05 version.

## Quick Start for PLC Checkout and Programming

If you have experience with PLCs, or if you just want to setup a quick example, this example is for you! This example is not intended to tell you everything you need to start-up your system, warnings and helpful tips are in the rest of the manual. It is only intended to give you a general picture of what you will need to do to get your system powered-up.

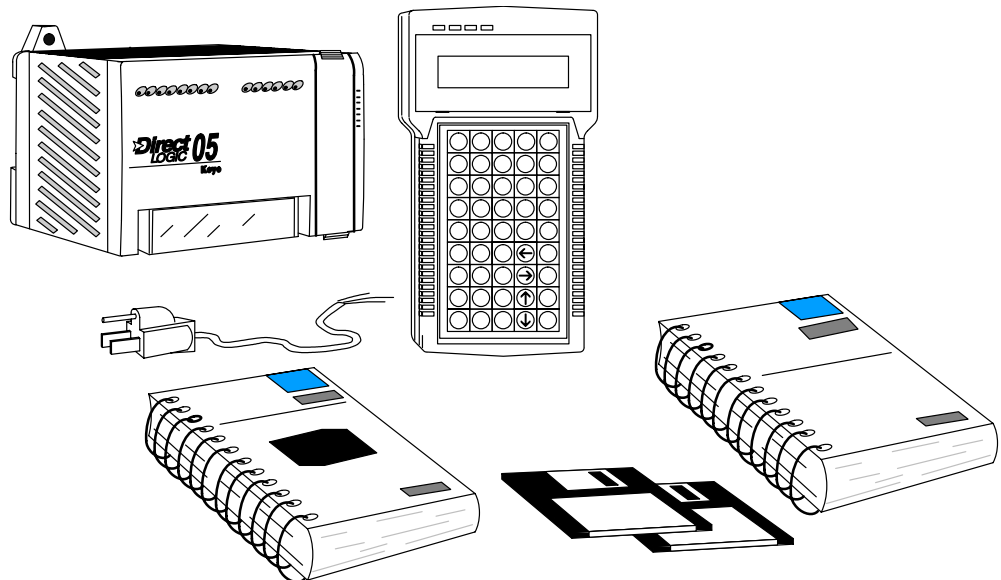
### Step 1: Unpack the DL05 Equipment

Unpack the DL05 and gather the parts necessary to build this demonstration system. The recommended components are:

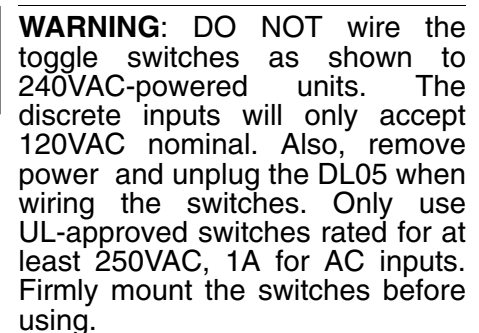
- DL05 Micro PLC
- AC power cord or DC power supply
- Toggle switches (see Step 2 on next page).
- Hook-up wire, 16-22 AWG
- DL05 User Manual (this manual)
- A small screwdriver, 5/8" flat or #1 Philips type

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

- **DirectSOFT** Programming Software, **DirectSOFT** Manual, and a programming cable (connects the DL05 to a personal computer), or
- D2-HPP Handheld Programmer (comes with programming cable), and the Handheld Programmer Manual



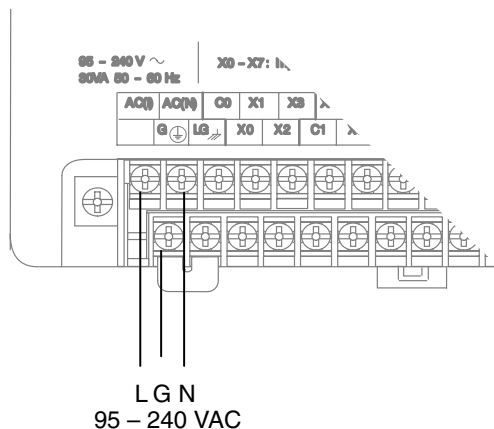
**D0-05DR, D0-05DD, D0-05DA**  
**D0-05DR-D, D0-05DD-D**  
 (DC input versions, 12-24VDC)



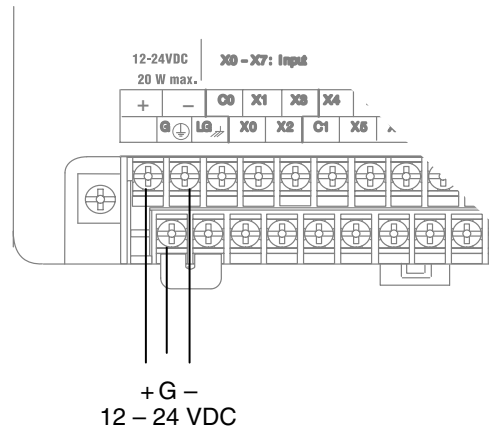
## Step 3: Connect the Power Wiring

Connect the power input wiring for the DL05. Observe all precautions stated earlier in this manual. For more details on wiring, see Chapter 2 on Installation, Wiring, and Specifications. When the wiring is complete, close the connector covers. Do not apply power at this time.

110/220 VAC Power Input

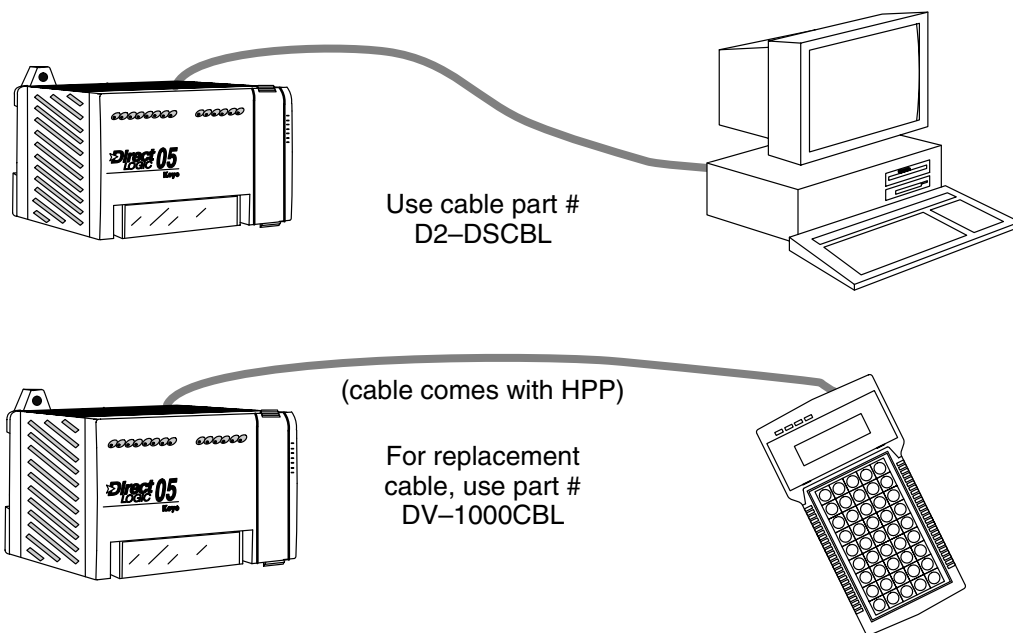


12/24 VDC Power Input



## Step 4: Connect the Programming Device

Most programmers will use **DirectSOFT** programming software, installed on a personal computer. Or, you may need the portability of the Handheld Programmer. Both devices will connect to COM port 1 of the DL05 via the appropriate cable.



## Step 5: Switch on the System Power

Apply power to the system and ensure the PWR indicator on the DL05 is on. If not, remove power from the system and check all wiring and refer to the troubleshooting section in Chapter 9 for assistance.

## Step 6: Initialize Scratchpad Memory

It's a good precaution to always clear the system memory (scratchpad memory) on a new DL05. There are two ways to clear the system memory:

- In **DirectSOFT**, select the **PLC** menu, then **Setup**, then **Initialize Scratchpad**. For additional information, see the **DirectSOFT** Manual.
- For the Handheld Programmer, use the AUX key and execute AUX 54. See the Handheld Programmer Manual for additional information.

## Step 7: Enter a Ladder Program

At this point, **DirectSOFT** programmers need to refer to the Quick Start Tutorial in the DirectSOFT Manual. There you will learn how to establish a communications link with the DL05 PLC, change CPU modes to Run or Program, and enter a program.

If you are learning how to program with the Handheld Programmer, make sure the CPU is in Program Mode (the RUN LED on the front of the DL05 should be off). If the RUN LED is on, use the MODE key on the Handheld Programmer to put the PLC in Program Mode. Enter the following keystrokes on the Handheld Programmer.

Equivalent <b>DirectSOFT</b> display	CLR	CLR	Clear the Program				
	C 2	E 4	AUX	ENT	ENT	CLR	
	NEXT	\$ STR	→	A 0	ENT		Move to the first address and enter X0 contact
	GX OUT	→	A 0	ENT			Enter output Y0
	SHFT	E 4	N TMR	D 3	ENT		Enter the END statement

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

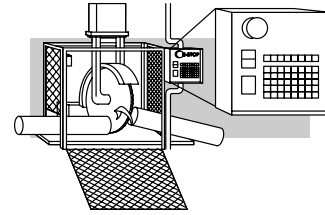
The RUN indicator on the PLC will illuminate indicating the CPU has entered the Run mode. If not, repeat this step, ensuring the program is entered properly or refer to the troubleshooting guide in chapter 8.

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

## Steps to Designing a Successful System

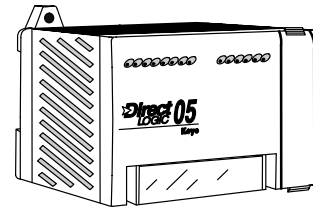
### Step 1: Review the Installation Guidelines

Always make safety the first priority in any system design. Chapter 2 provides several guidelines that will help you design a safer, more reliable system. This chapter also includes wiring guidelines for the various versions of the DL05 PLC.



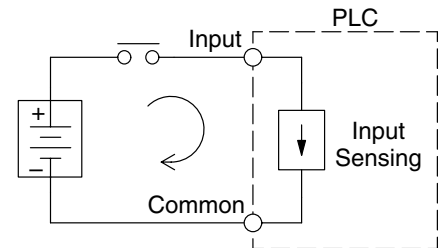
### Step 2: Understand the PLC Setup Procedures

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



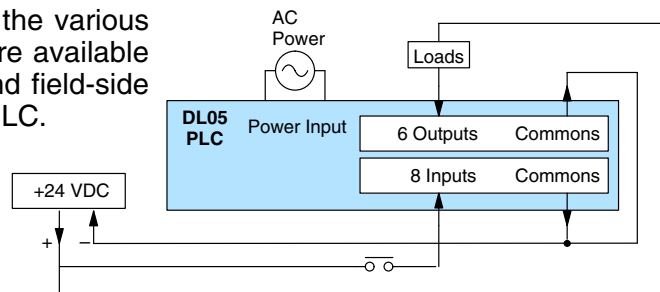
### Step 3: Review the I/O Selection Criteria

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



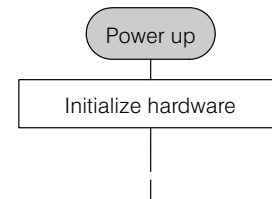
### Step 4: Choose a System Wiring Strategy

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



### Step 5: Understand the System Operation

Before you begin to enter a program, it is very helpful to understand how the DL05 system processes information. This involves not only program execution steps, but also involves the various modes of operation and memory layout characteristics.

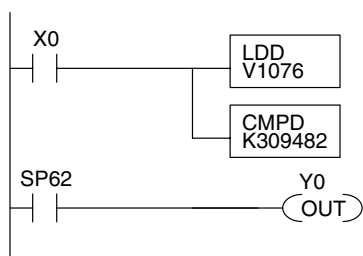


### Step 6: Review the Programming Concepts

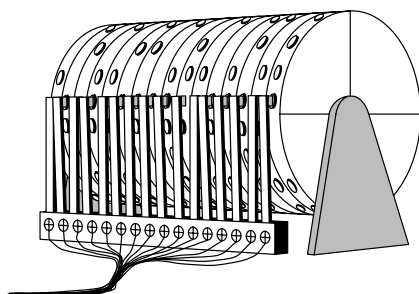
The DL05 PLC instruction set provides for three main approaches to solving the application program, depicted in the figure below.

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

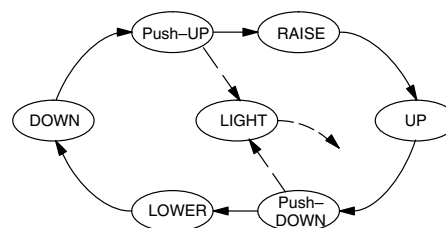
**Standard RLL Programming**  
(see Chapter 5)



**Timer/Event Drum Sequencer**  
(see Chapter 6)



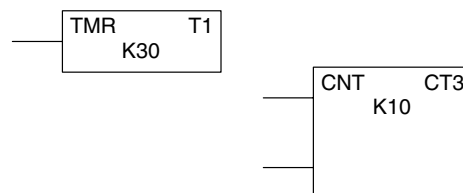
**Stage Programming**  
(see Chapter 7)



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

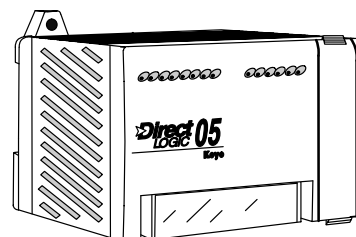
### Step 7: Choose the Instructions

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



### Step 8: Understand the Maintenance and Troubleshooting Procedures

Sometimes equipment failures occur when we least expect it. Switches fail, loads short and need to be replaced, etc. In most cases, the majority of the troubleshooting and maintenance time is spent trying to locate the problem. The DL05 Micro PLC has many built-in features such as error codes that can help you quickly identify problems.



## Questions and Answers about DL05 Micro PLCs

**Q. What is the instruction set like?**

**A.** The instruction set is very close to our popular DL240 CPU. However, there are significant additions, such as the drum instruction, networking, and High-Speed I/O capabilities.

**Q. Do I have to buy the full *DirectSOFT* programming package to program the DL05?**

**A.** No. We offer a DL05-specific version of *DirectSOFT* that's very affordable.

**Q. Is the DL05 expandable?**

**A.** No, the DL05 series are stand-alone PLCs. However, our DL205 system is expandable, yet very compact and affordable.

**Q. Does the DL05 have motion control capability?**

**A.** Yes. The High-Speed I/O features offer either encoder inputs with high-speed counting and presets with interrupt, or a pulse/direction output for stepper control. Three types of motion profiles are available, which are explained in Chapter 3.

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

**A.** The DL05 contains a non-removable FLASH memory for program storage, which may be written and erased thousands of times. You may transfer programs to/from *DirectSOFT* on a PC, or the HPP (which does support a removable EEPROM).

**Q. Does the DL05 contain fuses for its outputs?**

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

**Q. Is the DL05 Micro PLC U.L.<sup>®</sup> approved?**

**A.** The Micro PLC has met the requirements of UL (Underwriters' Laboratories, Inc.), and CUL (Canadian Underwriters' Laboratories, Inc.).

**Q. Does the DL05 Micro PLC comply with European Union (EU) Directives?**

**A.** The Micro PLC has met the requirements of the European Union Directives (CE).



**Q. Which devices can I connect to the communication ports of the DL05?**

**A. Port 1:** The port is RS-232C, fixed at 9600 baud, and uses the proprietary K-sequence protocol. The DL05 can also connect to MODBUS and DirectNET networks as a slave device through port 1. The port communicates with the following devices:

- DV-1000 Data Access Unit or Optimization Operator interface panels
- **DirectSOFT** (running on a personal computer)
- D2-HPP handheld programmer
- Other devices which communicate via K-sequence protocol should work with the DL05 Micro PLC. Contact the vendor for details.

**A. Port 2:** The port is RS-232C, with selective baud rates (300-38,400bps), address and parity. It also supports the proprietary K-sequence protocol as well as DirectNet and Modbus and non-sequence/print protocols.

**Q. Can the DL05 accept 5VDC inputs?**

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

# Installation, Wiring, and Specifications

---

## In This Chapter. . . .

- Safety Guidelines
  - Orientation to DL05 Front Panel
  - Mounting Guidelines
  - Wiring Guidelines
  - System Wiring Strategies
  - Glossary of Specification Terms
  - Wiring Diagrams and Specifications
  - D0-10ND3 DC Input
  - D0-16ND3 DC Input
  - D0-10TD1 DC Output
  - D0-16TD1 DC Output
  - D0-10TD2 DC Output
  - D0-16TD2 DC Output
  - D0-07CDR DC Input and Output
  - D0-08TR Relay Output
  - D0-08CDD1 DC Input and Output
-

## Safety Guidelines



**NOTE: Products with CE marks** perform their required functions safely and adhere to relevant standards as specified by CE directives provided they are used according to their intended purpose and that the instructions in this manual are adhered to. The protection provided by the equipment may be impaired if this equipment is used in a manner not specified in this manual.



**WARNING:** Providing a safe operating environment for personnel and equipment is your responsibility and should be your primary goal during system planning and installation. Automation systems can fail and may result in situations that can cause serious injury to personnel or damage to equipment. Do not rely on the automation system alone to provide a safe operating environment. You should use external electromechanical devices, such as relays or limit switches, that are independent of the PLC application to provide protection for any part of the system that may cause personal injury or damage.

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

### Plan for Safety

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

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

### Three Levels of Protection

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

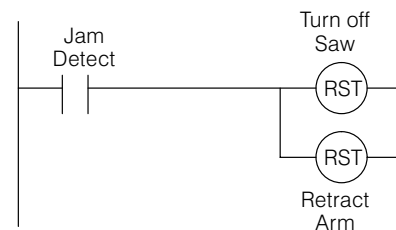
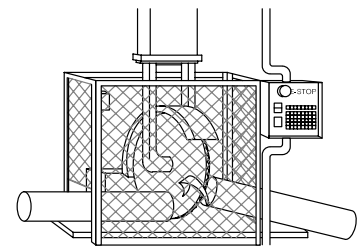
- Orderly system shutdown sequence in the PLC control program
- Mechanical disconnect for output module power
- Emergency stop switch for disconnecting system power

## Orderly System Shutdown

The first level of fault detection is ideally the PLC control program, which can identify machine problems. You must analyze your application and identify any shutdown sequences that must be performed. These types of problems are usually things such as jammed parts, etc. that do not pose a risk of personal injury or equipment damage.



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



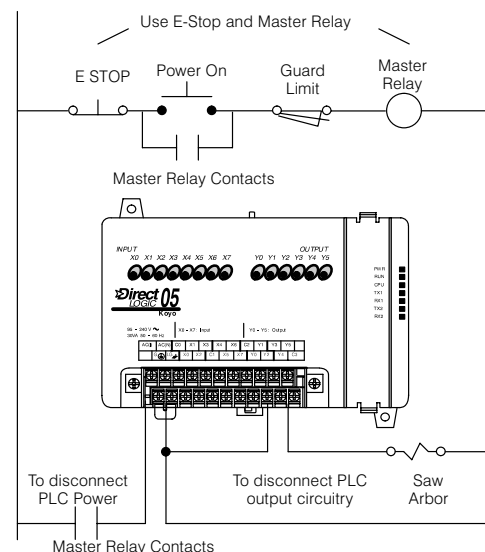
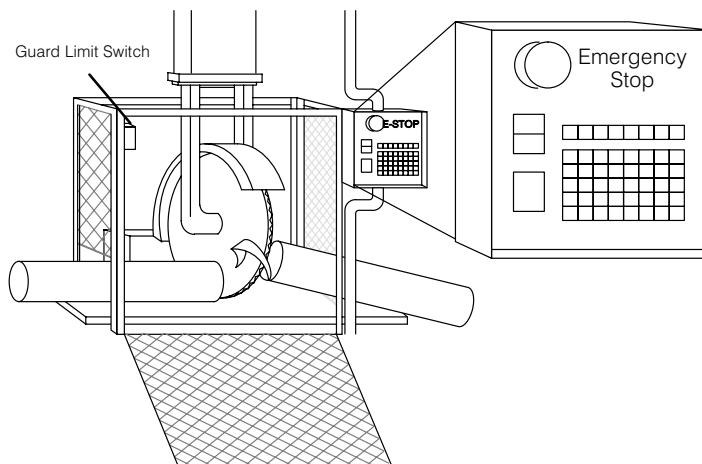
## System Power Disconnect

You should also use electromechanical devices, such as master control relays and/or limit switches, to prevent accidental equipment startup at an unexpected time. These devices should be installed in such a manner to prevent *any* machine operations from occurring.

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

## Emergency Stop

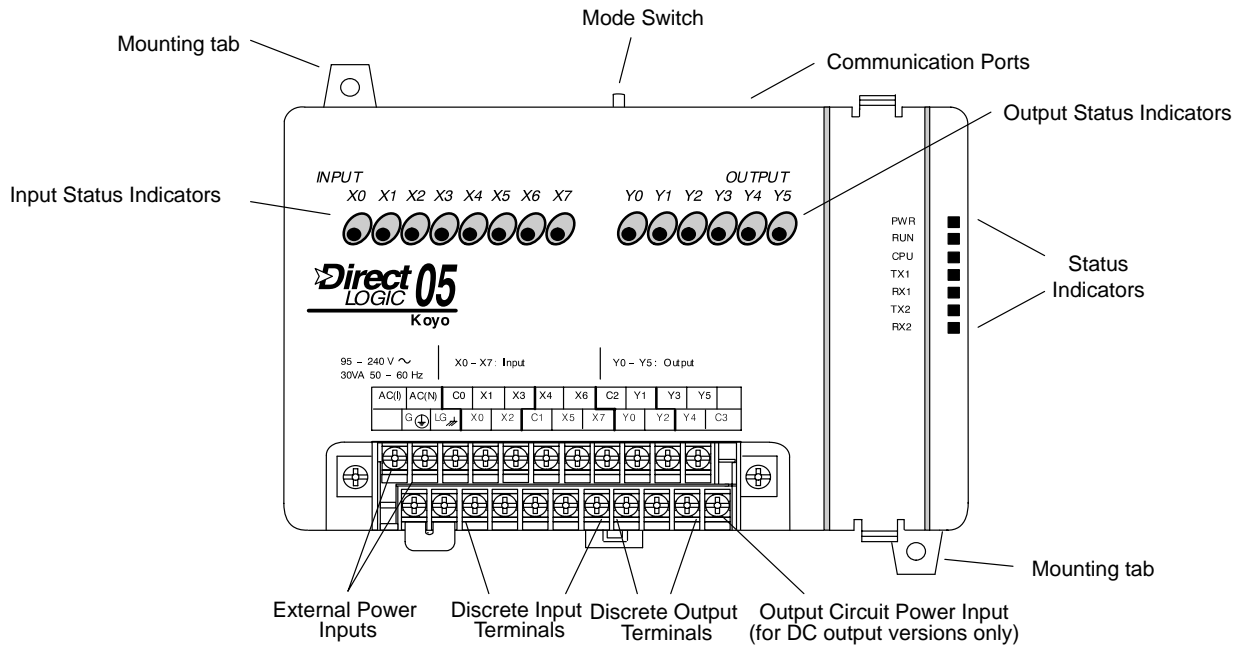
The machinery must provide a quick *manual* method of disconnecting *all* system power. The disconnect device or switch must be clearly labeled “**Emergency Stop**”.



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

## Orientation to DL05 Front Panel

Most connections, indicators, and labels on the DL05 Micro PLCs are located on its front panel. The communication ports are located on the top side of the PLC. Please refer to the drawing below.



The upper section of the connector accepts external power connections on the two left-most terminals. From left to right, the next five terminals are one of the input commons (C0) and input connections X1, X3, X4, and X6. The remaining four connections are an output common (C2) and output terminals Y1, Y3, and Y5.

The lower section of the connector has the chassis ground (G) and the logic ground (LG) on the two left-most terminals. The next two terminals are for the inputs X0 and X2. Next is the other input common (C1) followed by inputs X5 and X7. The last four terminals are for outputs Y0, Y2, Y4, and the second output common (C3). On DC output units, the end terminal on the right accepts power for the output stage.



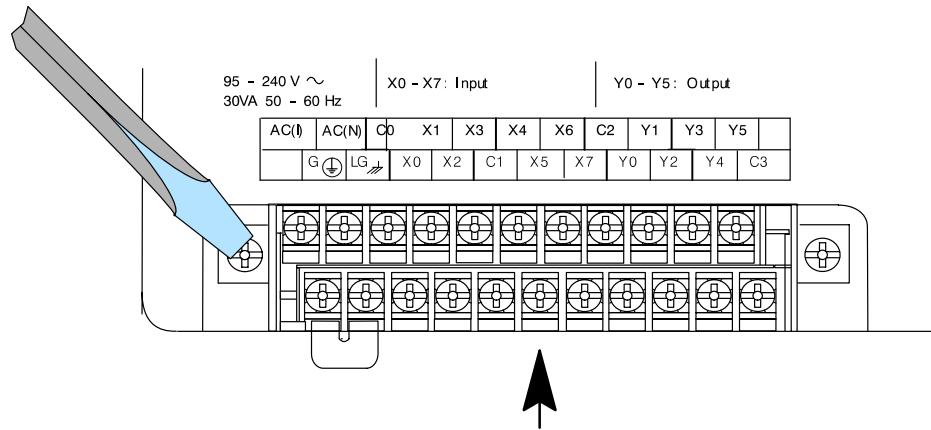
**WARNING:** For some applications, field device power may still be present on the terminal block even though the Micro PLC is turned off. To minimize the risk of electrical shock, check all field device power *before* you expose or remove either connector

## Connector Removal

All of the terminals for the DL05 are contained on one connector block. In some instances, it may be desirable to remove the connector block for easy wiring. The connector is designed for easy removal with just a small screwdriver. The drawing below shows the procedure for removal at one end.

### Connector Removal

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



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

The terminal block connector on DL05 PLCs have regular screw terminals, which will accept either standard or #1 Philips screwdriver tips. You can insert one 16 AWG wire under a terminal, or two 18 AWG wires (one on each side of the screw). Be careful not to overtighten; maximum torque is 6 inch/ounces.

Spare terminal block connectors and connector covers may be ordered by individual part numbers:

Part Number	Qty Per Package	Description
D0-IOCON	2	DL05 I/O Terminal Block
D0-IOCVR	2	DL05 I/O Terminal Cover

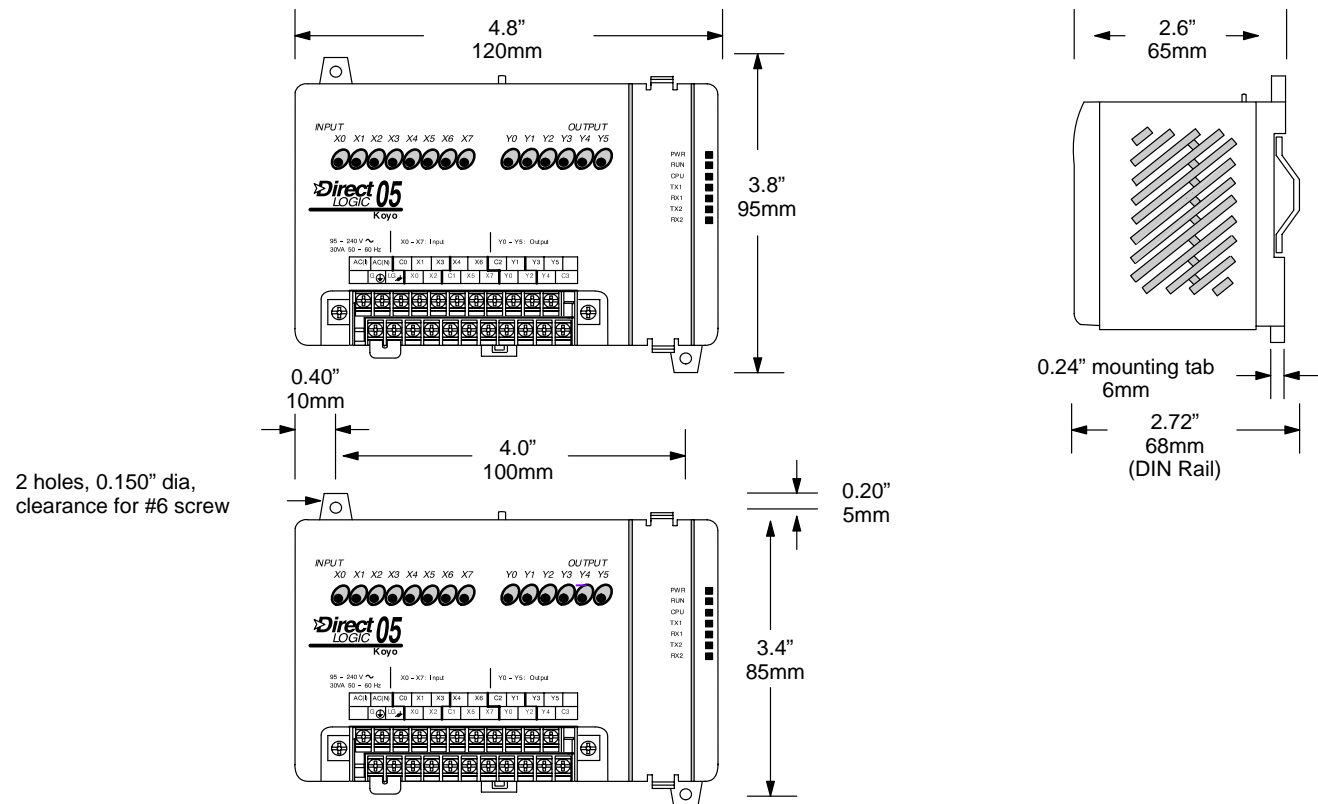
## Mounting Guidelines

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

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

### Unit Dimensions

The following diagram shows the outside dimensions and mounting hole locations for all versions of the DL05. Make sure you follow the installation guidelines to allow proper spacing from other components.



### Enclosures

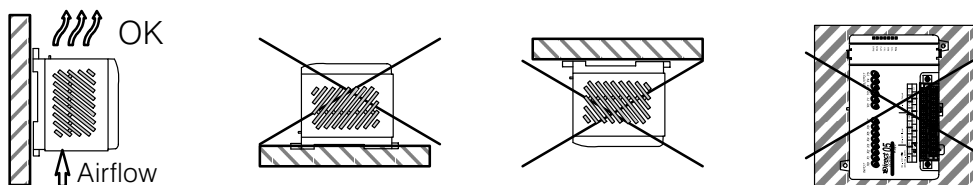
Your selection of a proper enclosure is important to ensure safe and proper operation of your DL05 system. Applications of DL05 systems vary and may require additional features. The minimum considerations for enclosures include:

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

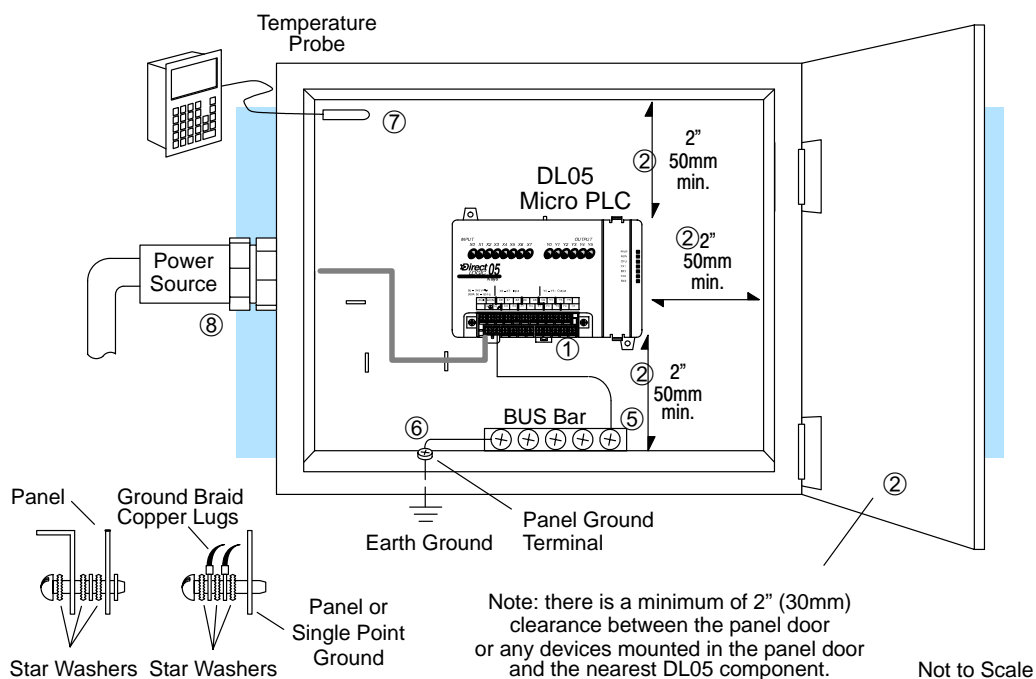
## Panel Layout & Clearances

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

1. Mount the PLCs horizontally as shown below to provide proper ventilation. You **cannot** mount the DL05 units vertically, upside down, or on a flat horizontal surface. If you place more than one unit in a cabinet, there must be a minimum of 7.2" (183mm) between the units.



2. Provide a minimum clearance of 2" (50mm) between the unit and all sides of the cabinet. *Note, remember to allow for any operator panels or other items mounted in the door.*
3. There should also be at least 3" (78mm) of clearance between the unit and any wiring ducts that run parallel to the terminals.



4. The ground terminal on the DL05 base must be connected to a single point ground. Use copper stranded wire to achieve a low impedance. Copper eye lugs should be crimped and soldered to the ends of the stranded wire to ensure good surface contact.
5. There must be a single point ground (i.e. copper bus bar) for all devices in the panel requiring an earth ground return. The single point of ground must be connected to the panel ground termination. The panel ground termination must be connected to earth ground. Minimum wire sizes, color coding, and general safety practices should comply with appropriate electrical codes and standards for your area.



6. A good common ground reference (Earth ground) is essential for proper operation of the DL05. One side of all control and power circuits and the ground lead on flexible shielded cable must be properly connected to Earth ground. There are several methods of providing an adequate common ground reference, including:
  - a) Installing a ground rod as close to the panel as possible.
  - b) Connection to incoming power system ground.
7. Evaluate any installations where the ambient temperature may approach the lower or upper limits of the specifications. If you suspect the ambient temperature will not be within the operating specification for the DL05 system, measures such as installing a cooling/heating source must be taken to get the ambient temperature within the range of specifications.
8. The DL05 systems are designed to be powered by 95-240 VAC or 12-24 VDC normally available throughout an industrial environment. Electrical power in some areas where the PLCs are installed is not always stable and storms can cause power surges. Due to this, powerline filters are recommended for protecting the DL05 PLCs from power surges and EMI/RFI noise. The Automation Powerline Filter, for use with 120 VAC and 240 VAC, 1-5 Amps, is an excellent choice. However, you can use a filter of your choice. These units install easily between the power source and the PLC.



### Using Mounting Rails

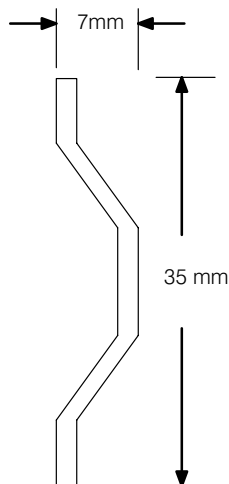
**NOTE:** If you are using other components in your system, make sure you refer to the appropriate manual to determine how those units can affect mounting dimensions.

DL05 Micro PLCs can be secured to a panel by using mounting rails. We recommend rails that conform to DIN EN standard 50 022. They are approximately 35mm high, with a depth of 7mm. If you mount the Micro PLC on a rail, do consider using end brackets on each side of the PLC. The end bracket helps keep the PLC from sliding horizontally along the rail, reducing the possibility of accidentally pulling the wiring loose.

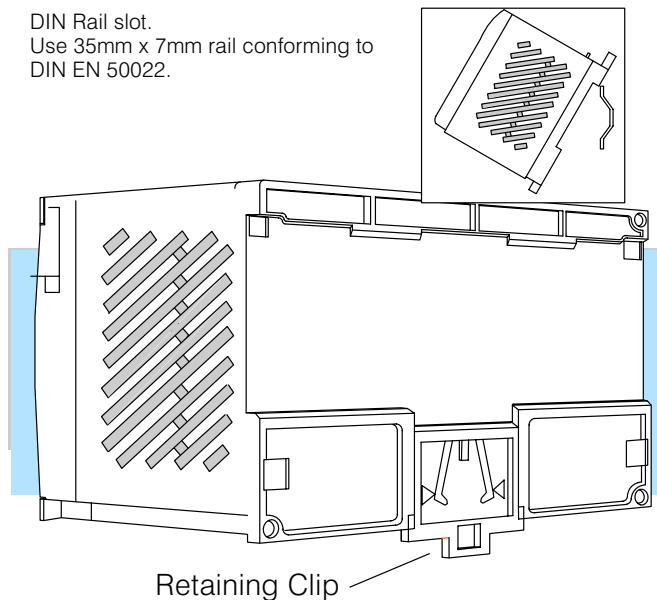
On the bottom of the PLC is a small retaining clip. To secure the PLC to a DIN rail, place it onto the rail and gently push up on the clip to lock it onto the rail.

To remove the PLC, pull down on the retaining clip, lift up on the PLC slightly, then pulling it away from the rail.

## DIN Rail Dimensions



DIN Rail slot.  
Use 35mm x 7mm rail conforming to  
DIN EN 50022.



Retaining Clip

**NOTE:** Refer to our catalog for a complete listing of **DINector** connection systems.



### Environmental Specifications

The following table lists the environmental specifications that generally apply to DL05 Micro PLCs. The ranges that vary for the Handheld Programmer are noted at the bottom of this chart. Certain output circuit types may have derating curves, depending on the ambient temperature and the number of outputs ON. Please refer to the appropriate section in this chapter pertaining to your particular DL05 PLC.

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

\* Operating temperature for the Handheld Programmer and the DV–1000 is 32° to 122° F (0° to 50° C)  
Storage temperature for the Handheld Programmer and the DV–1000 is −4° to 158° F (−20° to 70° C).

\*\*Equipment will operate down to 5% relative humidity. However, static electricity problems occur much more frequently at low humidity levels (below 30%). Make sure you take adequate precautions when you touch the equipment. Consider using ground straps, anti-static floor coverings, etc. if you use the equipment in low-humidity environments.

### Agency Approvals

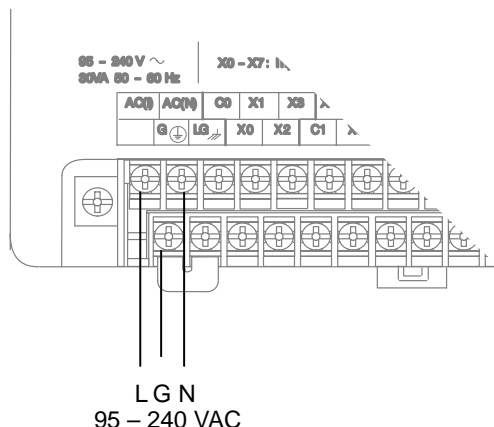
Some applications require agency approvals for particular components. The DL05 Micro PLC agency approvals are listed below:

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

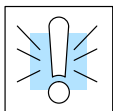
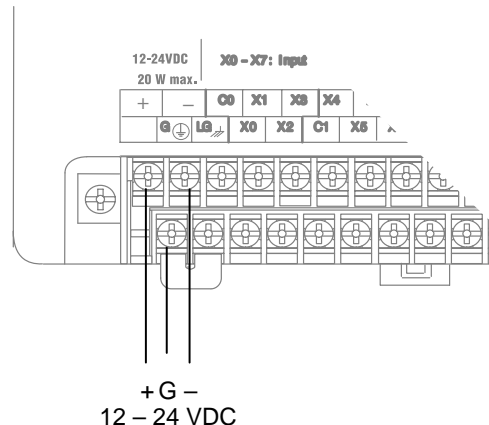
## Wiring Guidelines

Connect the power input wiring for the DL05. Observe all precautions stated earlier in this manual. For more details on wiring, see Chapter 2 on Installation, Wiring, and Specifications. When the wiring is complete, close the connector covers. Do not apply power at this time.

**110/220 VAC Power Input**



**12/24 VDC Power Input**



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

### Fuse Protection for Input Power

There are no internal fuses for the input power circuits, so external circuit protection is needed to ensure the safety of service personnel and the safe operation of the equipment itself. To meet UL/CUL specifications, the input power must be fused. Depending on the type of input power being used, follow these fuse protection recommendations:

#### 208/240 VAC Operation

When operating the unit from 208/240 VAC, whether the voltage source is a step-down transformer or from two phases, fuse both the line (L) and neutral (N) leads. The recommended fuse size is 0.375A.

#### 110/125 VAC Operation

When operating the unit from 110/125 VAC, it is only necessary to fuse the line (L) lead; it is not necessary to fuse the neutral (N) lead. The recommended fuse size is 0.5A.

### 12/24 VDC Operation

When operating at these lower DC voltages, wire gauge size is just as important as proper fusing techniques. Using large conductors minimizes the voltage drop in the conductor. Each DL05 input power terminal can accommodate one 16 AWG wire or two 18 AWG wires. A DC failure can maintain an arc for much longer time and distance than AC failures. Typically, the main bus is fused at a higher level than the branch device, which in this case is the DL05. The recommended fuse size for the branch circuit to the DL05 is 1A (for example, a Littlefuse 312.001 or equivalent).

### External Power Source

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

Item	DL05 VAC Powered Units	DL05 VDC Powered Units
Input Voltage Range	110/220 VAC (95–240 VAC)	12–24 VDC (10.8–26.4 VDC)
Maximum Inrush Current	13 A, 1ms (95–240 VAC) 15 A, 1ms (240–264 VAC)	10A
Maximum Power	30 VA	20 W
Voltage Withstand (dielectric)	1 minute @ 1500 VAC between primary, secondary, field ground	
Insulation Resistance	> 10 M $\Omega$ at 500 VDC	

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



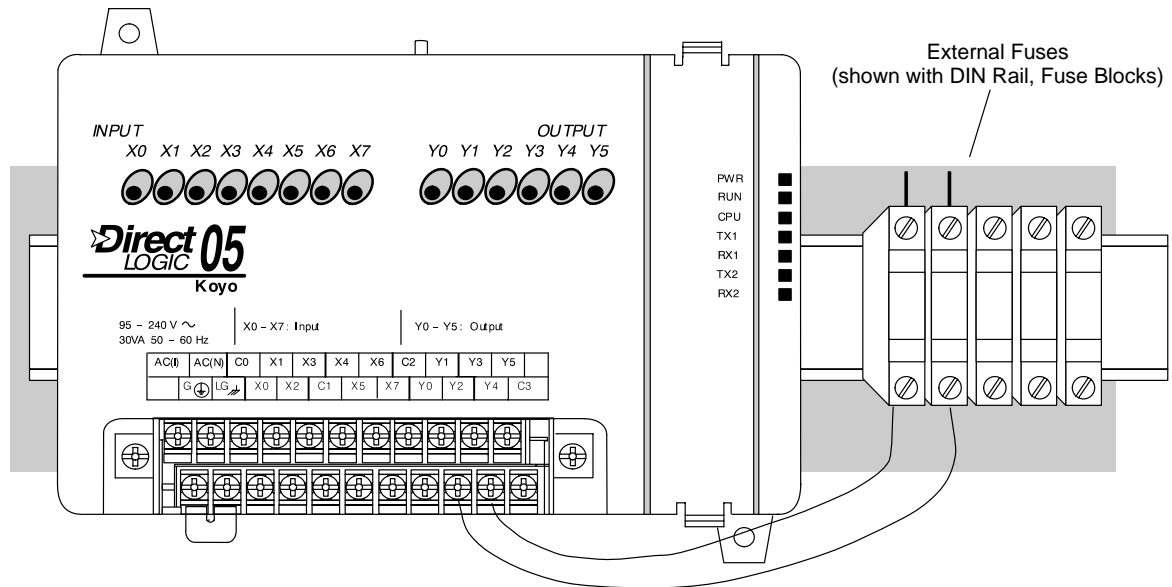
### Planning the Wiring Routes

The following guidelines provide general information on how to wire the I/O connections to DL05 Micro PLCs. For specific information on wiring a particular PLC refer to the corresponding specification sheet further in this chapter.

1. Each terminal connection of the DL05 PLC can accept one 16 AWG wire or two 18 AWG size wires. Do not exceed this recommended capacity.
2. Always use a continuous length of wire. Do not splice wires to attain a needed length.
3. Use the shortest possible wire length.
4. Use wire trays for routing where possible.
5. Avoid running wires near high energy wiring.
6. Avoid running input wiring close to output wiring where possible.
7. To minimize voltage drops when wires must run a long distance, consider using multiple wires for the return line.
8. Avoid running DC wiring in close proximity to AC wiring where possible.
9. Avoid creating sharp bends in the wires.
10. Install the recommended powerline filter to reduce power surges and EMI/RFI noise.

### Fuse Protection for Input and Output Circuits

Input and Output circuits on DL05 Micro PLCs do not have internal fuses. In order to protect your Micro PLC, we suggest you add external fuses to your I/O wiring. A fast-blow fuse, with a lower current rating than the I/O bank's common current rating can be wired to each common. Or, a fuse with a rating of slightly less than the maximum current per output point can be added to each output. Refer to the Micro PLC specification sheets further in this chapter to find the maximum current per output point or per output common. Adding the external fuse does not guarantee the prevention of Micro PLC damage, but it will provide added protection.



### I/O Point Numbering

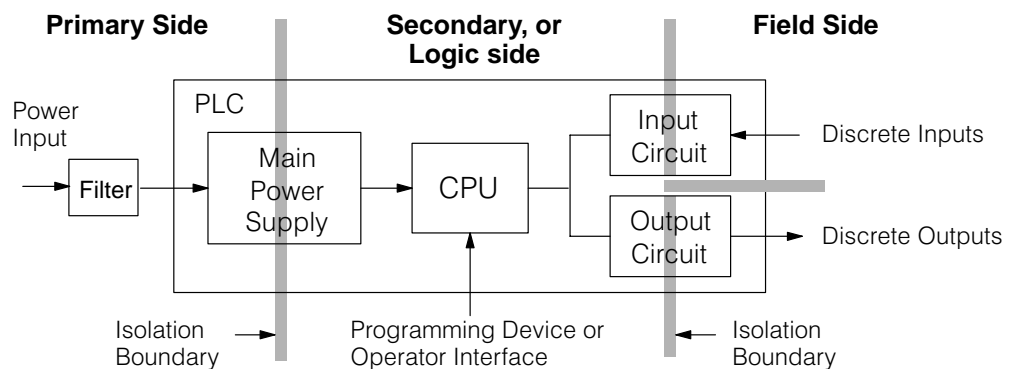
All DL05 Micro PLCs have a fixed I/O configuration. It follows the same octal numbering system used on other **DirectLogic** family PLCs, starting at X0 and Y0. The letter X is always used to indicate inputs and the letter Y is always used for outputs. The I/O numbering always starts at zero and does not include the digits 8 or 9. The addresses are typically assigned in groups of 8 or 16, depending on the number of points in an I/O group. For the DL05 the eight inputs use reference numbers X0 – X7. The six output points use references Y0 – Y5.

## System Wiring Strategies

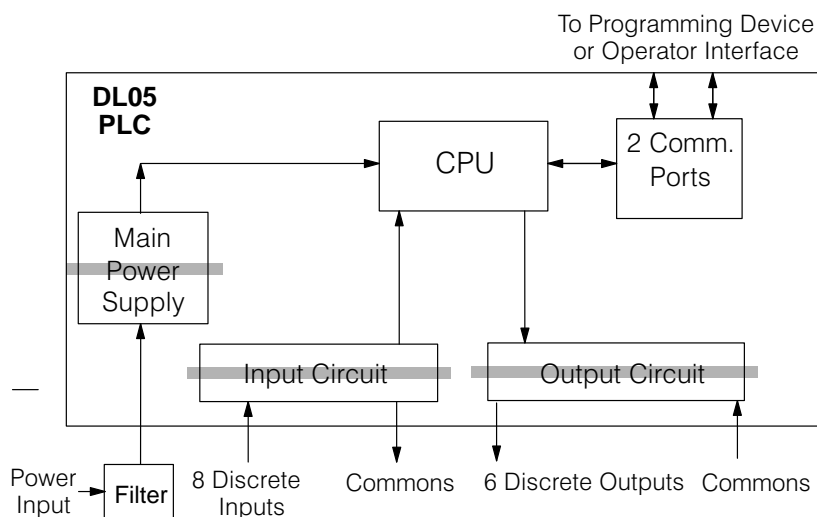
### PLC Isolation Boundaries

The DL05 Micro PLC is very flexible and will work in many different wiring configurations. By studying this section before actual installation, you can probably find the best wiring strategy for your application. This will help to lower system cost, wiring errors, and avoid safety problems.

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



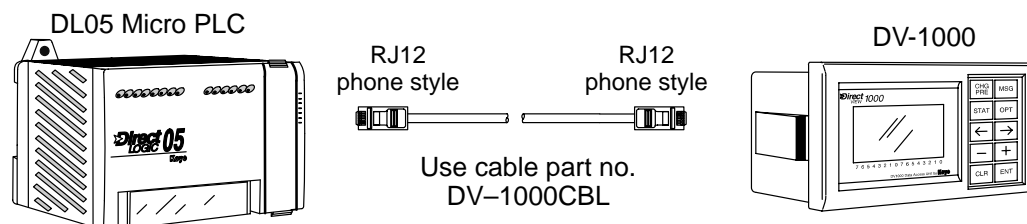
The next figure shows the internal layout of DL05 PLCs, as viewed from the front panel.



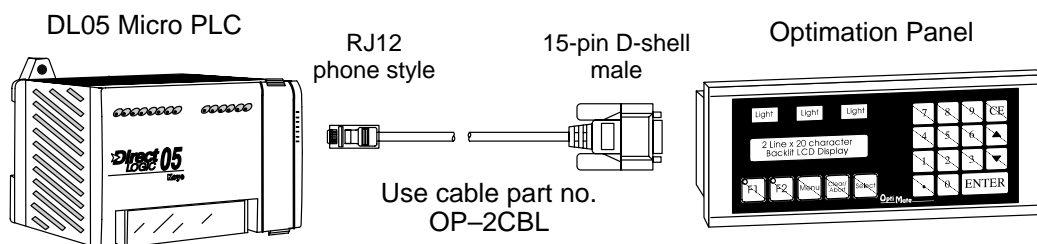
### Connecting Operator Interface Devices

Operator interfaces require data and power connections. Operator interfaces with a large CRT usually require separate AC power. However, small operator interface devices like the popular DV-1000 Data Access Unit and the Optimization panels may be powered directly from the DL05 Micro PLC.

Connect the DV-1000 to either communication port on the DL05 Micro PLC using the cable shown below. A single cable contains transmit/receive data wires and +5V power.

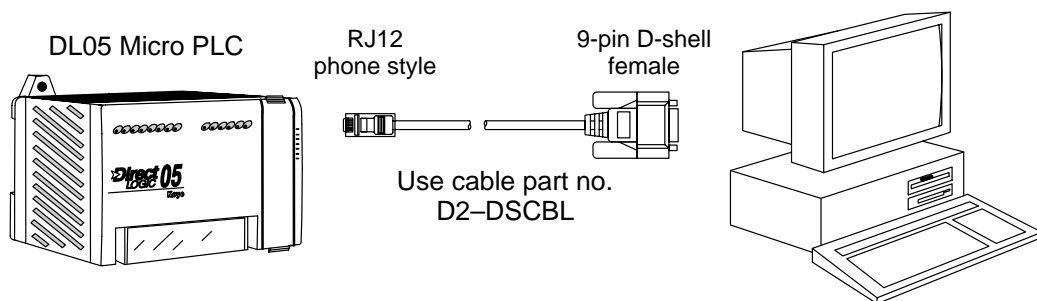


Optimization operator interface panels require separate power and communications connections. Connect the DL05 to the proper D-shell connector on the rear of the Optimization panel using the cable shown below. Optimization panels require 8–30VDC power.

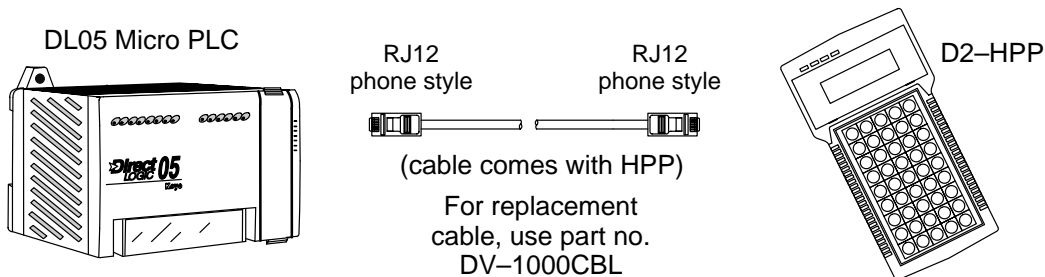


### Connecting Programming Devices

DL05 Micro PLCs can be programmed with either a handheld programmer or with *DirectSOFT* on a PC. Connect the DL05 to a PC using the cable shown below.



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



## Sinking / Sourcing Concepts

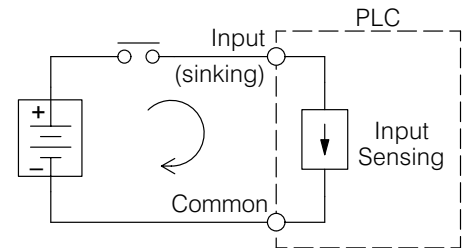
Before going further in our study of wiring strategies, we must have a solid understanding of “sinking” and “sourcing” concepts. Use of these terms occurs frequently in input or output circuit discussions. It is the goal of this section to make these concepts easy to understand, further ensuring your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

**Sinking = Path to supply ground (–)**

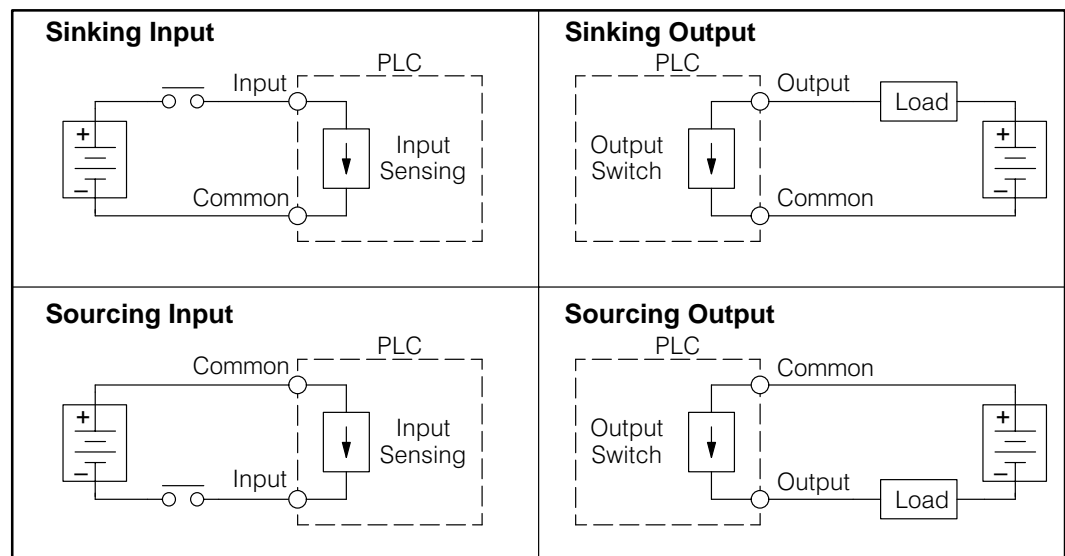
**Sourcing = Path to supply source (+)**

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

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



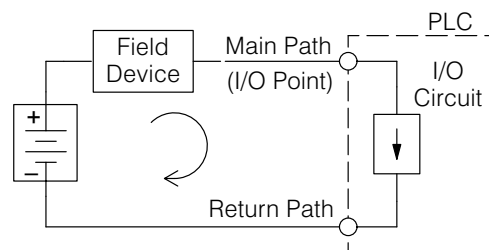
By applying the circuit principle above to the four possible combinations of input/output sinking/sourcing types, we have the four circuits as shown below. DL05 Micro PLCs provide all except the sourcing output I/O circuit types.



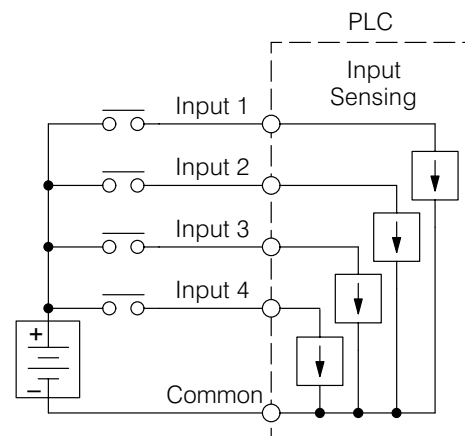


## I/O “Common” Terminal Concepts

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

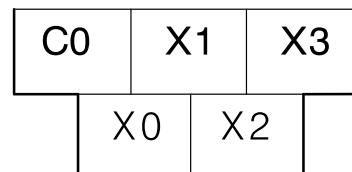


If we had unlimited space and budget for I/O terminals, then every I/O point could have two dedicated terminals just as the figure above shows. However, providing this level of flexibility is not practical or even necessary for most applications. So, most Input or Output point groups on PLCs share the return path among two or more I/O points. The figure to the right shows a group (or *bank*) of 4 input points which share a common return path. In this way, the four inputs require only five terminals instead of eight.



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

Most DL05 input and output circuits are grouped into banks that share a common return path. The best indication of I/O common grouping is on the wiring label. The I/O common grouping bar, labeled at the right, occurs in the section of wiring label below it. It indicates X0, X1, X2, and X3 share the common terminal located to the left of X1.



The following complete label shows two banks of four inputs and two banks of three outputs. One common is provided for each bank.

AC(I)	AC(N)	C0	X1	X3	X4	X6	C2	Y1	Y3	Y5	
	G $\oplus$	LG $\neq$	X0	X2	C1	X5	X7	Y0	Y2	Y4	C3

The following label is for DC output versions. One common is provided for all of the outputs and the terminal on the bottom right accepts power for the output stage.

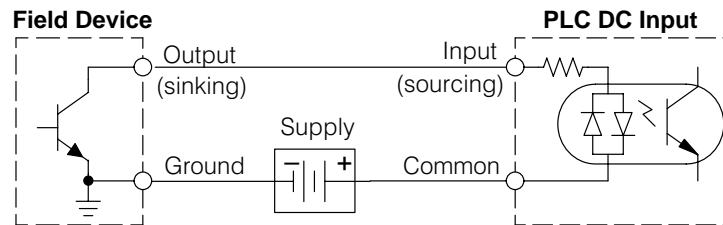
AC(I)	AC(N)	C0	X1	X3	X4	X6	C2	Y1	Y3	Y5	
	G $\oplus$	LG $\neq$	X0	X2	C1	X5	X7	Y0	Y2	Y4	+V

## Connecting DC I/O to “Solid State” Field Devices

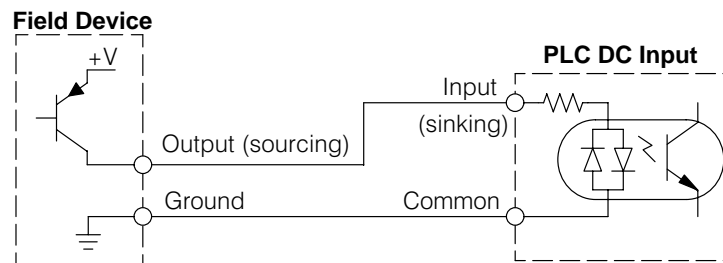
### Solid State Input Sensors

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

The DL05's DC inputs are flexible in that they detect current flow in either direction, so they can be wired as either sourcing or sinking. In the following circuit, a field device has an open-collector NPN transistor output. It sinks current from the PLC input point, which sources current. The power supply can be the FA-24PS +24 VDC power supply or another supply (+12 VDC or +24VDC), as long as the input specifications are met.



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

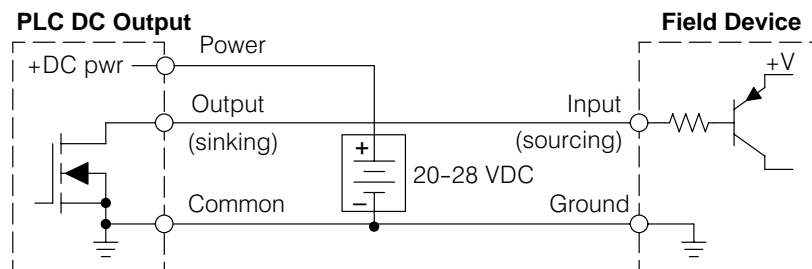


### Solid State Output Loads

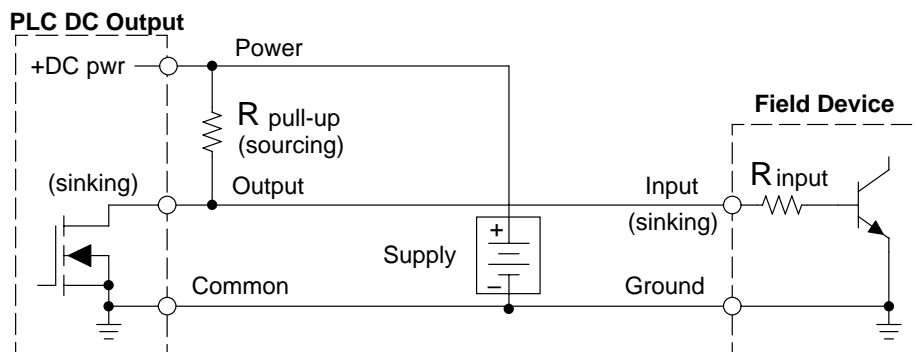
Sometimes an application requires connecting a PLC output point to a solid state input on a device. This type of connection is usually made to carry a low-level signal, not to send DC power to an actuator.

The DL05's DC outputs are sinking-only. This means that each DC output provides a path to ground when it is energized. Also, remember that all six outputs have the same electrical common, even though there are two common terminal screws. Finally, recall that the DC output circuit requires power (20 – 28 VDC) from an external power source.

In the following circuit, the PLC output point sinks current to the output common when energized. It is connected to a sourcing input of a field device input.



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



**NOTE:** DO NOT attempt to drive a heavy load ( $>25$  mA) with this pull-up method.  
**NOTE 2:** Using the pull-up resistor to implement a sourcing output has the effect of inverting the output point logic. In other words, the field device input is energized when the PLC output is OFF, from a ladder logic point-of-view. Your ladder program must comprehend this and generate an inverted output. Or, you may choose to cancel the effect of the inversion elsewhere, such as in the field device.

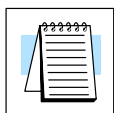
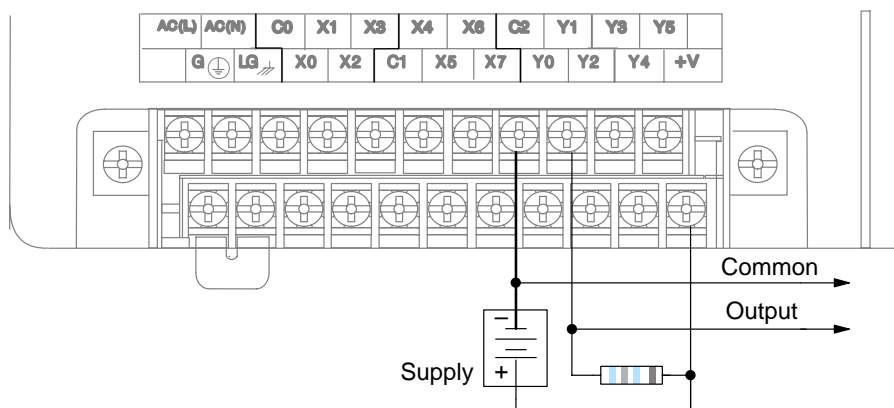
It is important to choose the correct value of  $R_{\text{pull-up}}$ . In order to do so, we need to know the nominal input current to the field device ( $I_{\text{input}}$ ) when the input is energized. If this value is not known, it can be calculated as shown (a typical value is 15 mA). Then use  $I_{\text{input}}$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

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

$$R_{\text{pull-up}} = \frac{V_{\text{supply}} - 0.7}{I_{\text{input}}} - R_{\text{input}}$$

$$P_{\text{pull-up}} = \frac{V_{\text{supply}}^2}{R_{\text{pull-up}}}$$

The drawing below shows the actual wiring of the DL05 Micro PLC to the supply and pull-up resistor.



## Relay Output Wiring Methods

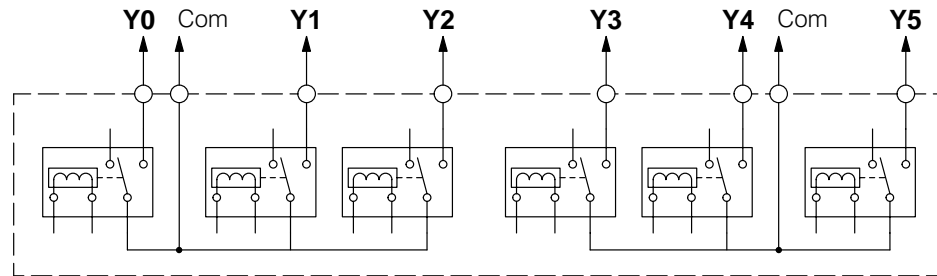
The D0-05AR and the D0-05DR models feature relay outputs. Relays are best for the following applications:

- Loads that require higher currents than the solid-state DL05 outputs can deliver
- Cost-sensitive applications
- Some output channels need isolation from other outputs (such as when some loads require AC while others require DC)

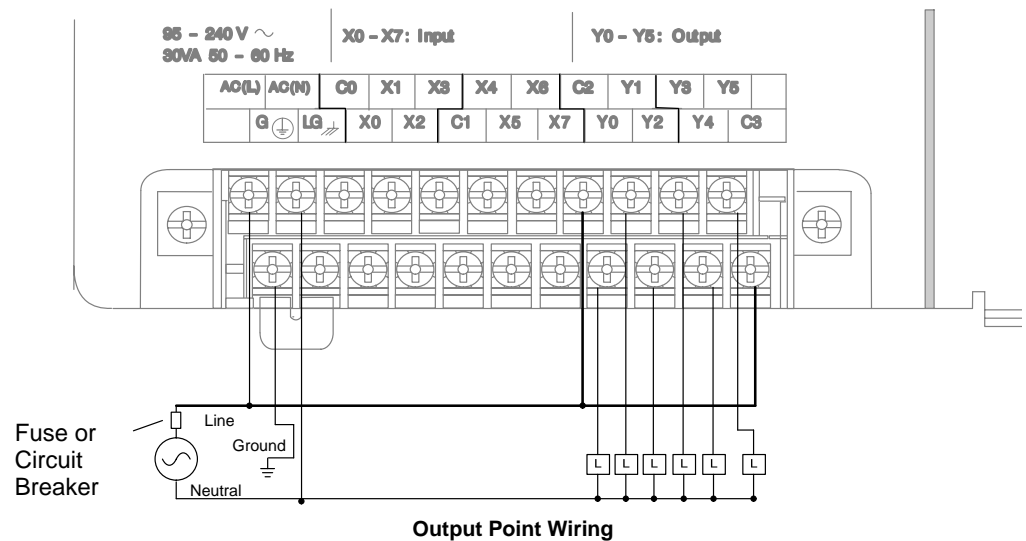
Some applications in which NOT to use relays:

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

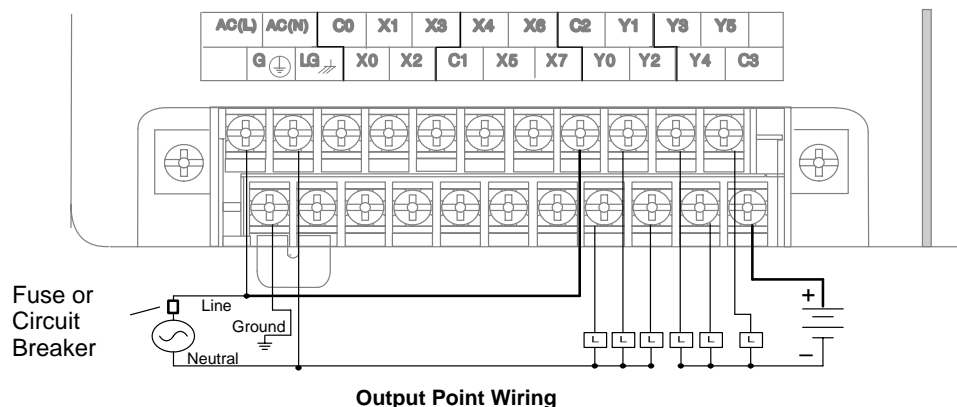
Assuming relays are right for your application, we're now ready to explore various ways to wire relay outputs to the loads. Note that there are six normally-open SPST relays available. They are organized with three relays per common. The figure below shows the relays and the internal wiring of the PLC. Note that each group is isolated from the other group of outputs.



In the circuit below, all loads use the same AC power supply which powers the DL05 PLC. In this example, all commons are connected together.



In the circuit on the following page, loads for Y0 - Y2 use the same AC power supply which powers the DL05 PLC. Loads for Y3 - Y5 use a separate DC supply. In this example, the commons are separated according to which supply powers the associated load.



### Surge Suppression For Inductive Loads

Inductive load devices (devices with a coil) generate transient voltages when de-energized with a relay contact. When a relay contact is closed it “bounces”, which energizes and de-energizes the coil until the “bouncing” stops. The transient voltages generated are much larger in amplitude than the supply voltage, especially with a DC supply voltage.

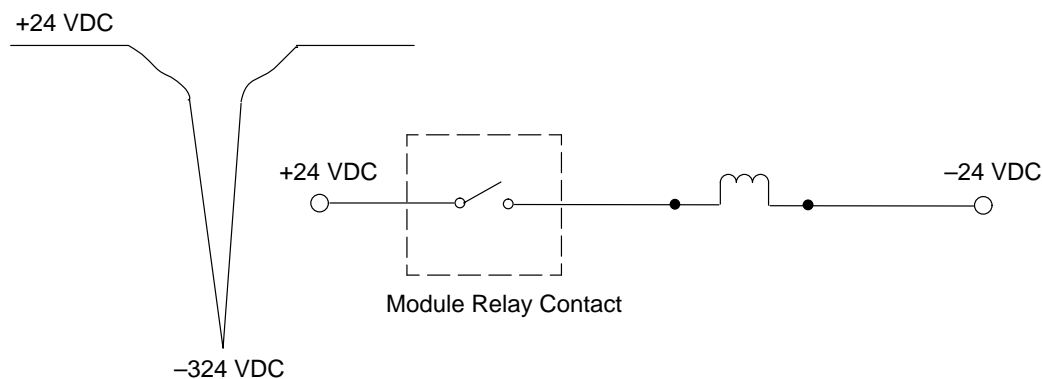
When switching a DC-supplied inductive load the full supply voltage is always present when the relay contact opens (or “bounces”). When switching an AC-supplied inductive load there is one chance in 60 (60 Hz) or 50 (50 Hz) that the relay contact will open (or “bounce”) when the AC sine wave is zero crossing. If the voltage is not zero when the relay contact opens there is energy stored in the inductor that is released when the voltage to the inductor is suddenly removed. This release of energy is the cause of the transient voltages.

When inductive load devices (motors, motor starters, interposing relays, solenoids, valves, etc.) are controlled with relay contacts, it is recommended that a surge suppression device be connected directly across the coil of the field device. If the inductive device has plug-type connectors, the suppression device can be installed on the terminal block of the relay output.

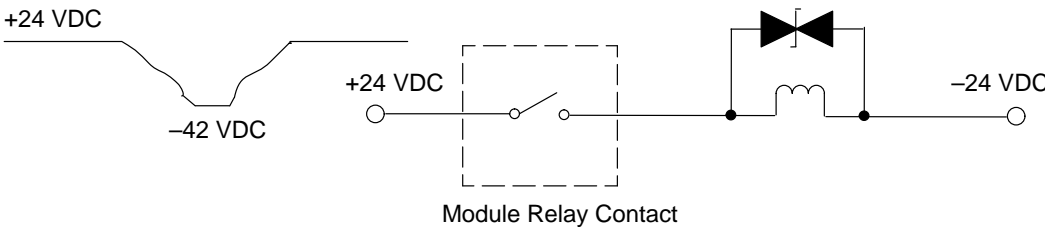
**Transient Voltage Suppressors (TVS or transorb)** provide the best surge and transient suppression of AC and DC powered coils, providing the fastest response with the smallest overshoot.

**Metal Oxide Varistors (MOV)** provide the next best surge and transient suppression of AC and DC powered coils.

For example, the waveform in the figure below shows the energy released when opening a contact switching a 24 VDC solenoid. Notice the large voltage spike.



This figure shows the same circuit with a transorb (TVS) across the coil. Notice that the voltage spike is significantly reduced.



Use the following table to help select a TVS or MOV suppressor for your application based on the inductive load voltage.

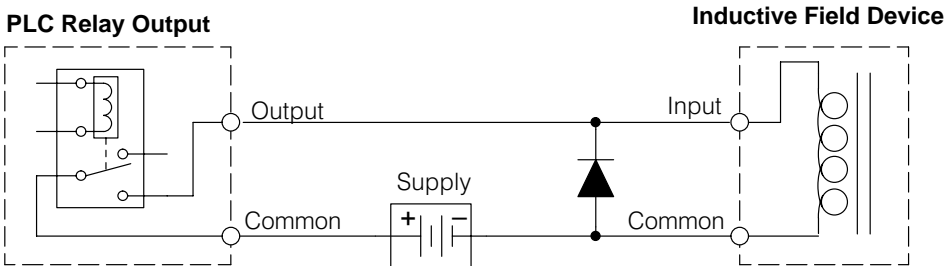
Vendor / Catalog	Type (TVS, MOV, Diode)	Inductive Load Voltage	Part Number
General Instrument Transient Voltage Suppressors, LiteOn Diodes; from DigiKey Catalog; Phone: 1-800-344-4539	TVS	110/120 VAC	P6KE180CAGICT-ND
	TVS	220/240 VAC	P6KE350CA
	TVS	12/24 VDC or VAC	P6K30CAGICT-ND
	Diode	12/24 VDC or VAC	1N4004CT-ND
Harris Metal Oxide Varistors; from Newark Catalog; Phone: 1-800-463-9275	MOV	110/120 VAC	V150LA20C
	MOV	220/240 VAC	V250LA20C

Prolonging Relay Contact Life

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

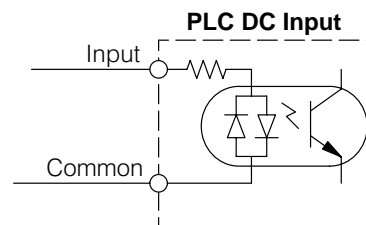
For inductive loads in DC circuits we recommend using a suppression diode as shown in the following diagram (DO NOT use this circuit with an AC power supply). When the load is energized the diode is reverse-biased (high impedance). When the load is turned off, energy stored in its coil is released in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

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

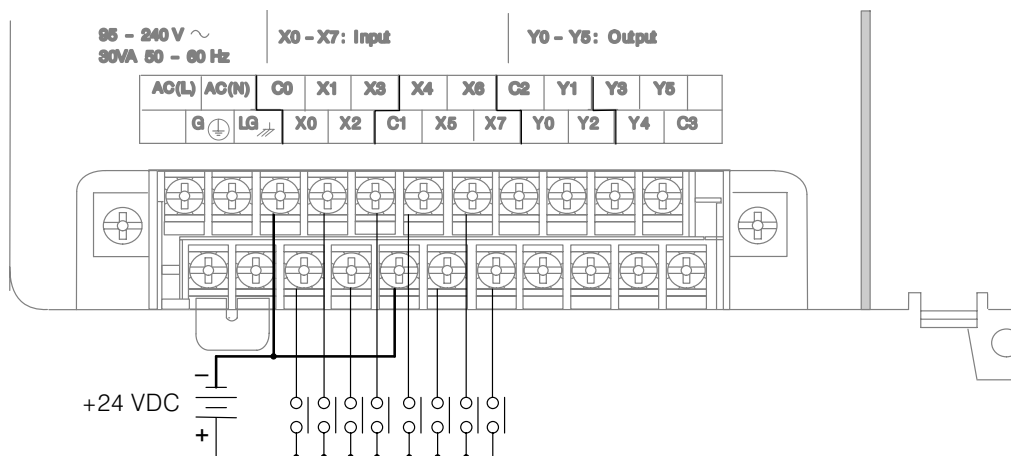


## DC Input Wiring Methods

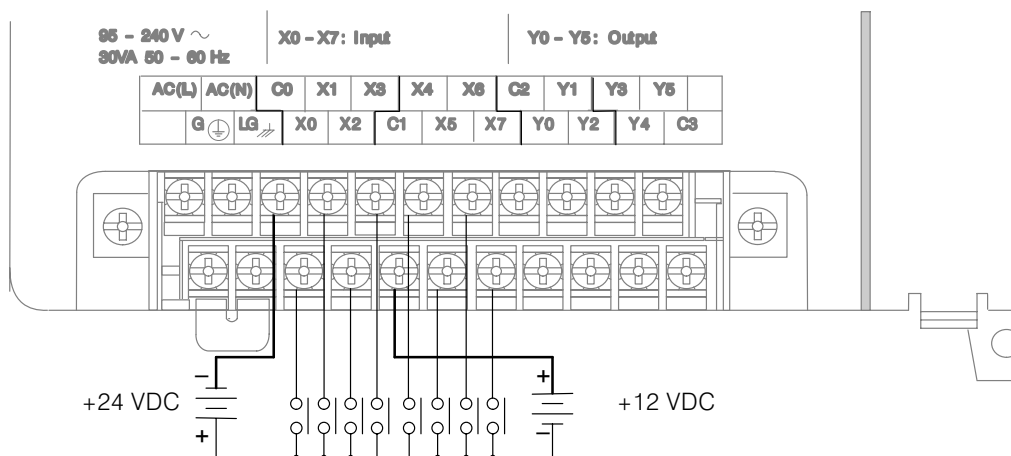
DL05 Micro PLCs with DC inputs are particularly flexible because they can be either sinking or sourcing. The dual diodes (shown to the right) allow current to flow in either direction. The inputs accept 10.8 – 26.4 VDC. The target applications are +12 VDC and +24 VDC. You can actually wire half of the inputs as DC sinking and the other half as DC sourcing. Inputs grouped by a common must be all sinking or all sourcing.



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



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

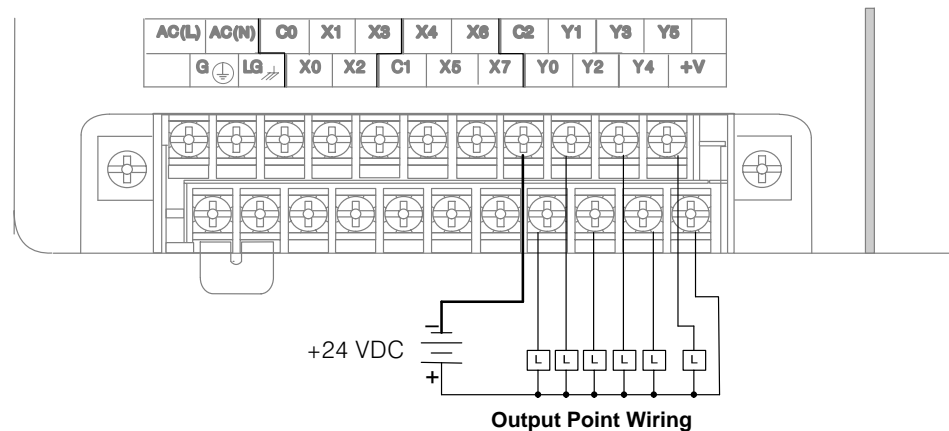


## DC Output Wiring Methods

DL05 DC output circuits are high-performance transistor switches with low on-resistance and fast switching times. Please note the following characteristics which are unique to the DC output type:

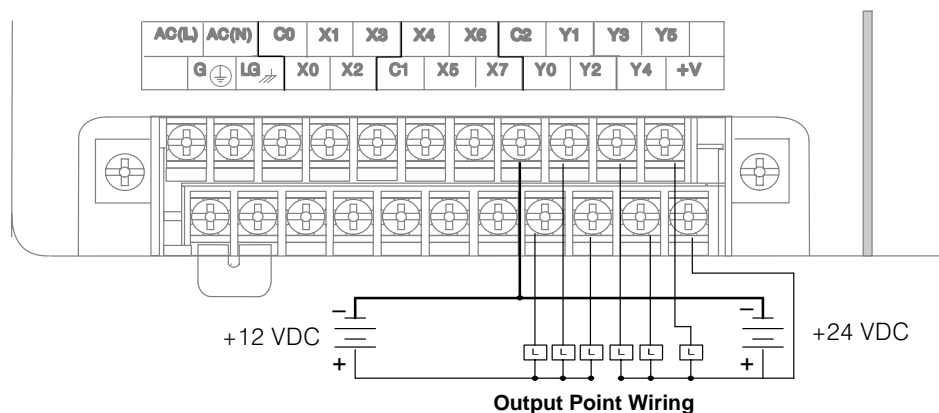
- There is only one electrical common for all six outputs. All six outputs belong to one bank.
- The output switches are current-sinking only. However, you can still use different DC voltages from one load to another.
- The output circuit inside the PLC requires external power. The supply (–) must be connected to a common terminal, and the supply (+) connects to the right-most terminal on the upper connector.

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



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

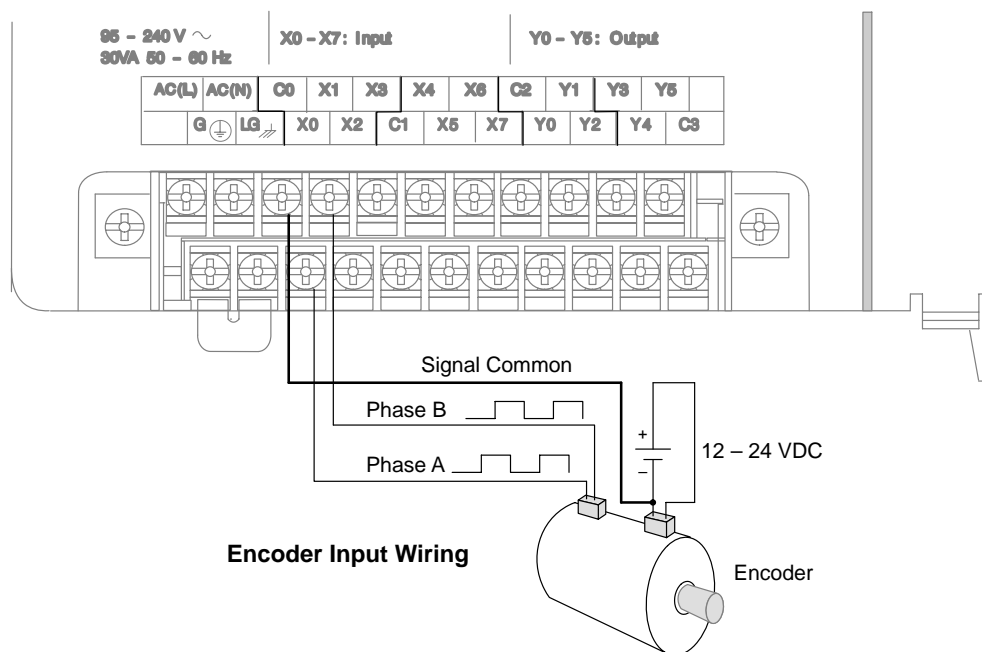
- all supply voltages are within the specified range
- all output points are wired as sinking
- all source (–) terminals are connected together



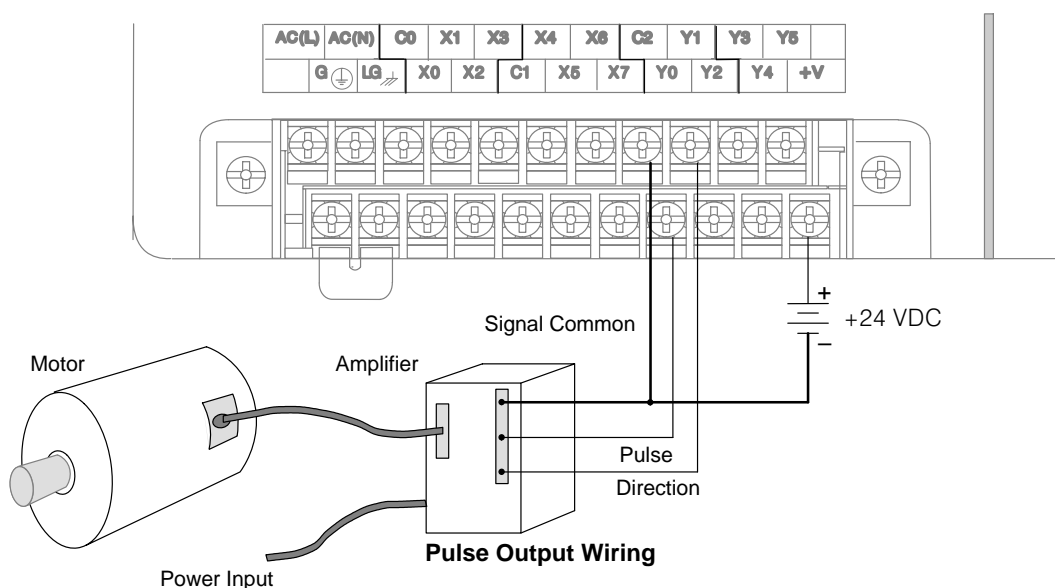


## High-Speed I/O Wiring Methods

DL05 versions with DC type input or output points contain a dedicated High-Speed I/O circuit (HSIO). The circuit configuration is programmable, and it processes select I/O points independently from the CPU scan. Chapter 3 discusses the programming options for HSIO. While the HSIO circuit has six modes, we show wiring diagrams for two of the most popular modes in this chapter. The high-speed input interfaces to points X0 – X2. Properly configured, the DL05 can count quadrature pulses at up to 5 kHz from an incremental encoder as shown below.



DL05 versions with DC type output points can use the High Speed I/O Pulse Output feature. It can generate high-speed pulses for specialized control such as stepper motor / intelligent drive systems. Output Y0 and Y1 can generate pulse and direction signals, or it can generate CCW and CW pulse signals respectively. See Chapter 3 on high-speed input and pulse output options.



## Glossary of Specification Terms

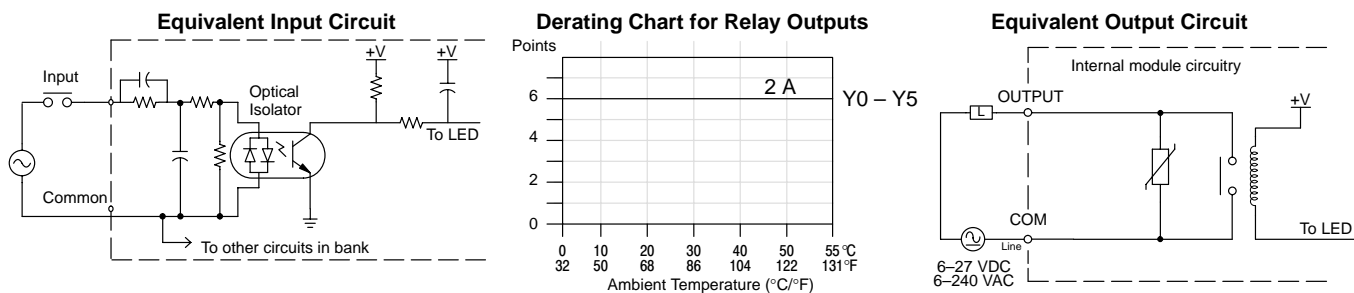
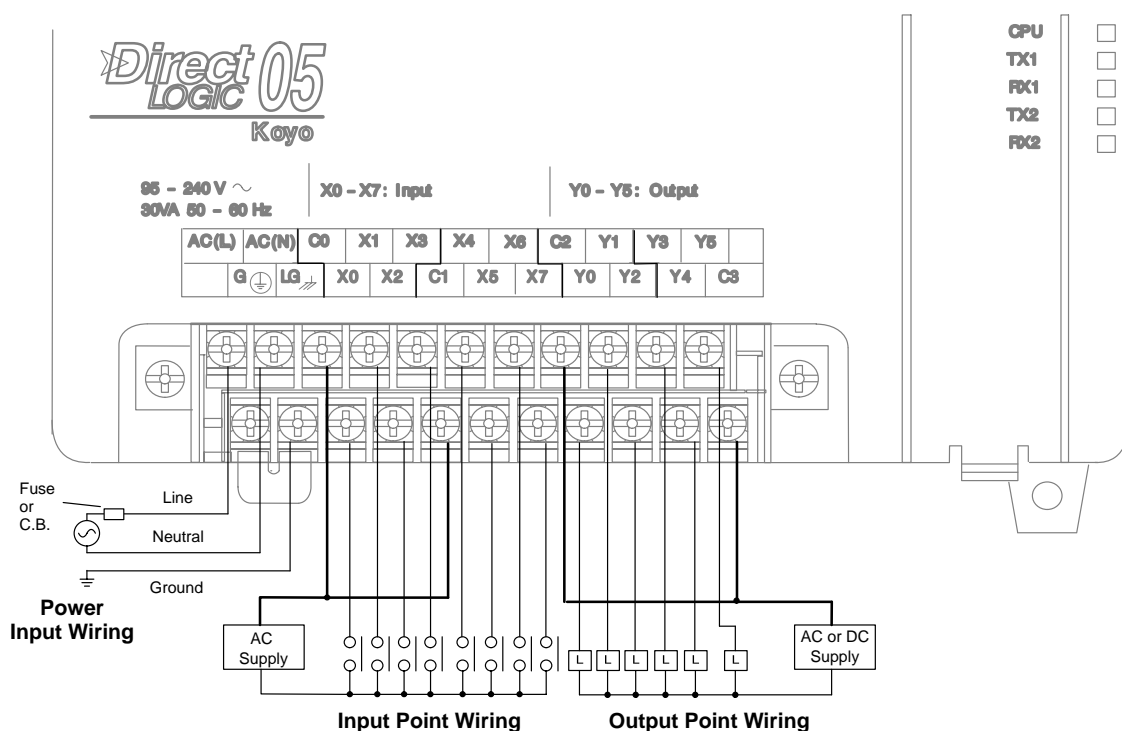
<b>Discrete Input</b>	One of eight input connections to the PLC which converts an electrical signal from a field device to a binary status (off or on), which is read by the internal CPU each PLC scan.
<b>Discrete Output</b>	One of six output connections from the PLC which converts an internal ladder program result (0 or 1) to turn On or Off an output switching device. This enables the program to turn on and off large field loads.
<b>I/O Common</b>	A connection in the input or output terminals which is shared by multiple I/O circuits. It usually is in the return path to the power supply of the I/O circuit.
<b>Input Voltage Range</b>	The operating voltage range of the input circuit.
<b>Maximum Voltage</b>	Maximum voltage allowed for the input circuit.
<b>ON Voltage Level</b>	The minimum voltage level at which the input point will turn ON.
<b>OFF Voltage Level</b>	The maximum voltage level at which the input point will turn OFF
<b>Input Impedance</b>	Input impedance can be used to calculate input current for a particular operating voltage.
<b>Input Current</b>	Typical operating current for an active (ON) input.
<b>Minimum ON Current</b>	The minimum current for the input circuit to operate reliably in the ON state.
<b>Maximum OFF Current</b>	The maximum current for the input circuit to operate reliably in the OFF state.
<b>OFF to ON Response</b>	The time the module requires to process an OFF to ON state transition.
<b>ON to OFF Response</b>	The time the module requires to process an ON to OFF state transition.
<b>Status Indicators</b>	The LEDs that indicate the ON/OFF status of an input or output point. All LEDs on DL05 Micro PLCs are electrically located on the logic side of the input or output circuit.

## Wiring Diagrams and Specifications

The remainder of this chapter dedicates two pages to each of the eight versions of DL05 Micro PLCs. Each section contains a basic wiring diagram, equivalent I/O circuits, and specification tables. Please refer to the section which describes the particular DL05 version used in your application.

### D0-05AR I/O Wiring Diagram

The D0-05AR Micro PLC features eight AC inputs and six relay contact outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals at the left as shown.



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

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

### D0-05AR General Specifications

External Power Requirements	95 – 240 VAC, 30 VA maximum,
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Slave) MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Master/Slave) MODBUS (Master/Slave) Non-sequence / print
Programming cable type	D2-DSCBL
Operating Temperature	32 to 131° F (0 to 55° C)
Storage Temperature	-4 to 158° F (-20 to 70° C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3-304
Terminal Type	Removable
Wire Gauge	One AWG16 or two AWG18, AWG24 minimum

### AC Input Specifications X0 – X7

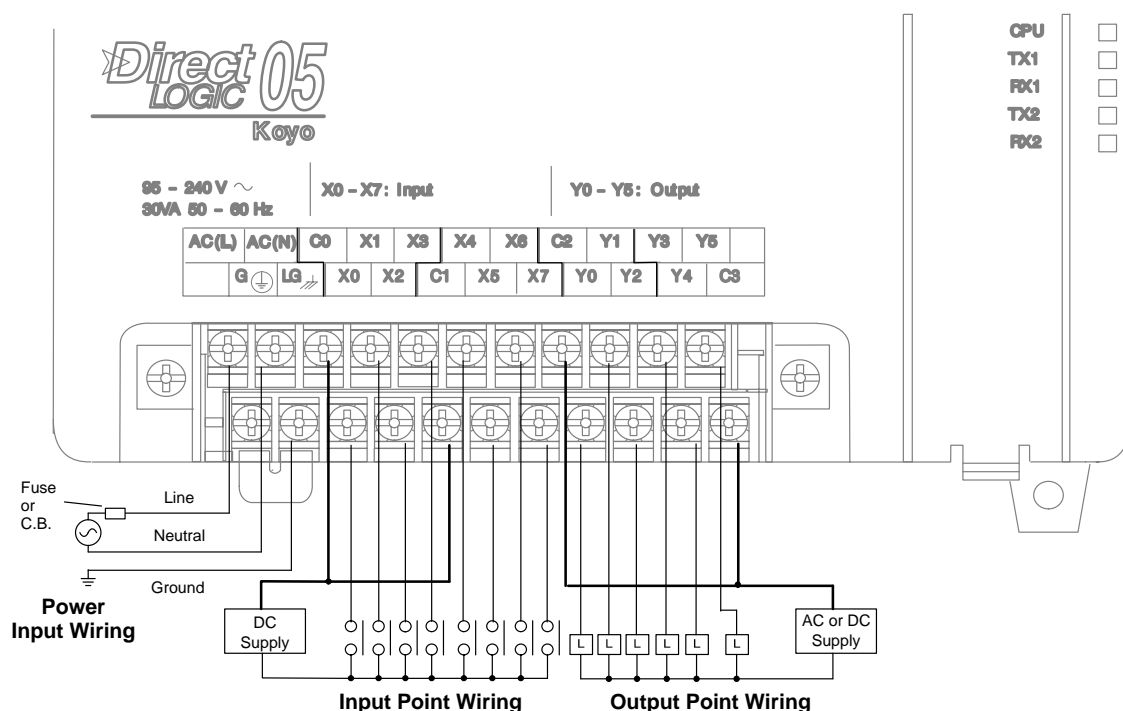
Input Voltage Range (Min. - Max.)	80 – 132 VAC, 47 - 63 Hz
Operating Voltage Range	90 – 120 VAC, 47 -63 Hz
Input Current	8 mA @ 100 VAC at 50 Hz 10 mA @ 100 VAC at 60 Hz
Max. Input Current	12 mA @ 132 VAC at 50 Hz 15 mA @ 132 VAC at 60 Hz
Input Impedance	14KΩ @50 Hz, 12KΩ @60 Hz
ON Current/Voltage	>6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

### Relay Output Specifications Y0 – Y5

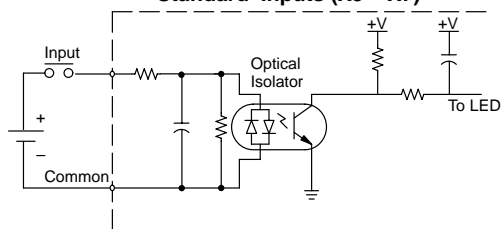
Output Voltage Range (Min. – Max.)	5 – 264 VAC (47 -63 Hz), 5 – 30 VDC
Operating Voltage Range	6 – 240 VAC (47 -63 Hz), 6 – 27 VDC
Output Current	2A / point, 6A / common
Max. leakage current	0.1 mA @264VAC
Smallest Recommended Load	5 mA @5 VDC
OFF to ON Response	< 15 mS
ON to OFF Response	< 10 mS
Status Indicators	Logic Side
Commons	3 channels / common x 2 banks
Fuses	None (external recommended)

## D0-05DR I/O Wiring Diagram

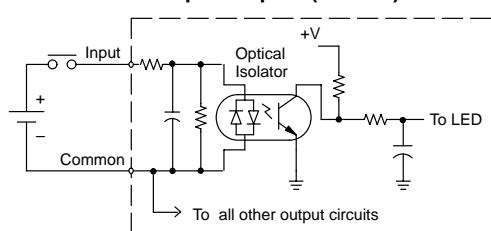
These micro PLCs feature eight DC inputs and six relay contact outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals at the left as shown.



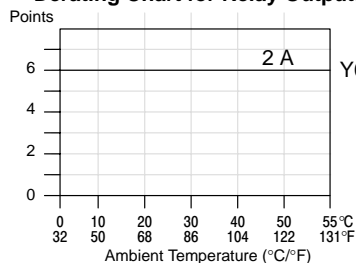
**Equivalent Circuit, Standard Inputs (X3 - X7)**



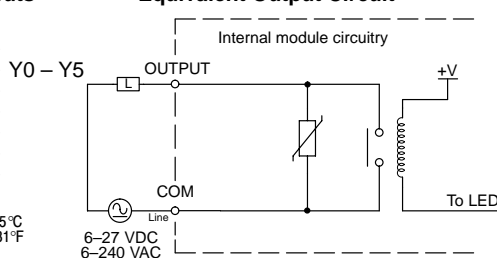
**Equivalent Circuit, High-Speed Inputs (X0 - X2)**



**Derating Chart for Relay Outputs**



**Equivalent Output Circuit**



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

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

## D0-05DR General Specifications

External Power Requirements	95 – 240 VAC, 30 VA maximum,
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Slave) MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Master/Slave) MODBUS (Master/Slave) Non-sequence / print
Programming cable type	D2-DSCBL
Operating Temperature	32 to 131° F (0 to 55° C)
Storage Temperature	-4 to 158° F (-20 to 70° C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3-304
Terminal Type	Removable
Wire Gauge	One AWG16 or two AWG18, AWG24 minimum

## DC Input Specifications

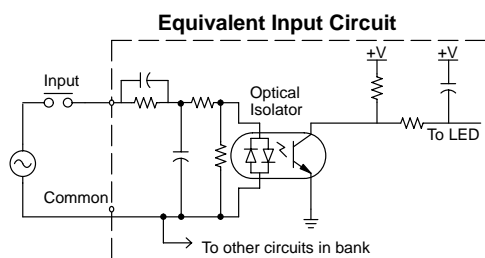
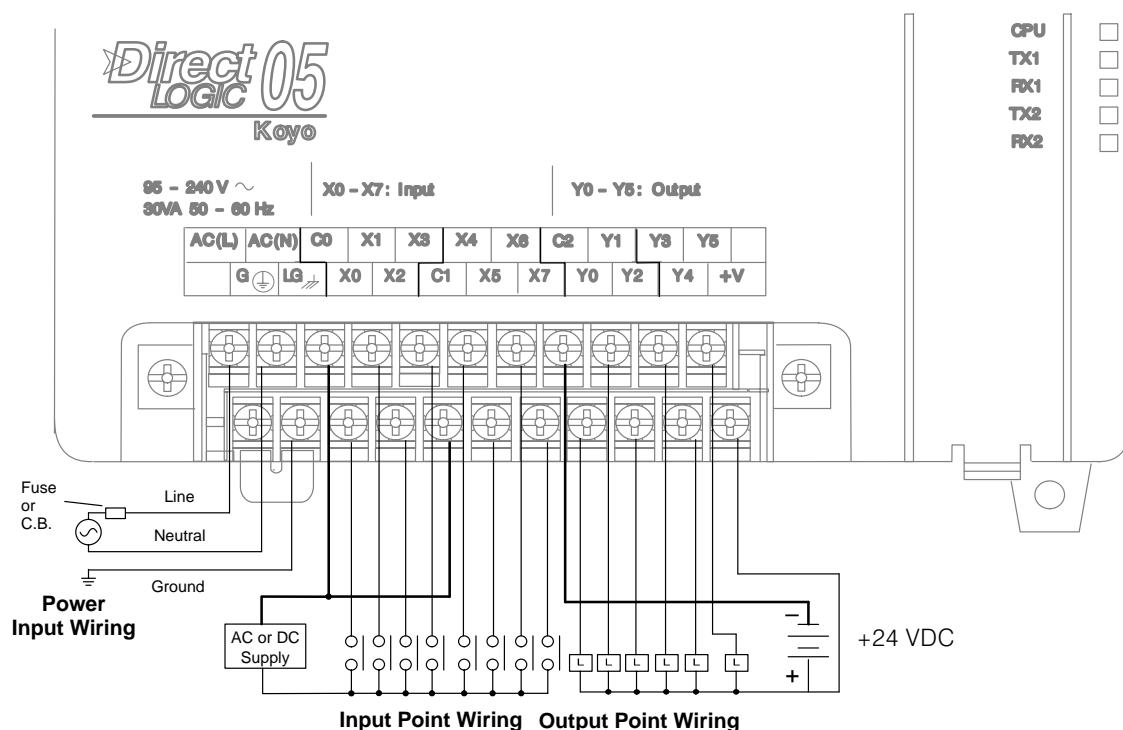
Parameter	High-Speed Inputs, X0 – X2	Standard DC Inputs X3 – X7
Min. - Max. Voltage Range	10.8 – 26.4 VDC	10.8 – 26.4 VDC
Operating Voltage Range	12 -24 VDC	12 -24 VDC
Peak Voltage	30 VDC (5 kHz maximum frequency)	30 VDC
Minimum Pulse Width	100 µs	N/A
ON Voltage Level	> 10 VDC	> 10 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Input Impedance	1.8 kΩ @ 12 – 24 VDC	2.8 kΩ @ 12 – 24 VDC
Max. Input Current	6mA @12VDC 13mA @24VDC	4mA @12VDC 8.5mA @24VDC
Minimum ON Current	>5 mA	>4 mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<100 µs	2 – 8 mS, 4 mS typical
ON to OFF Response	< 100 µs	2 – 8 mS, 4 mS typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 2 bank	

## Relay Output Specifications

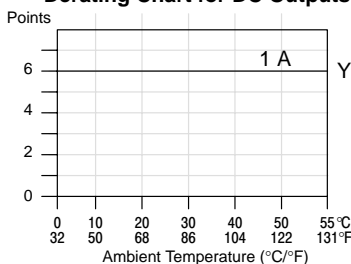
Output Voltage Range (Min. - Max.)	5 -264 VAC (47 -63 Hz), 5 - 30 VDC
Operating Voltage	6 -240 VAC (47 -63 Hz), 6 - 27 VDC
Output Current	2A / point 6A / common
Maximum Voltage	264 VAC, 30 VDC
Max leakage current	0.1 mA @264 VAC
Smallest Recommended Load	5 mA
OFF to ON Response	< 15 mS
ON to OFF Response	< 10 mS
Status Indicators	Logic Side
Commons	3 channels / common x 2 banks
Fuses	None (external recommended)

## D0-05AD I/O Wiring Diagram

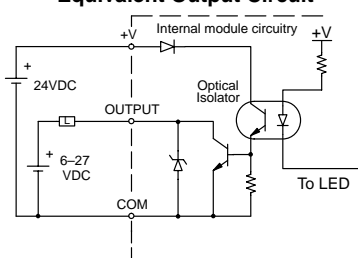
The D0-05AD Micro PLC features eight AC inputs and six DC outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals at the left as shown.



Derating Chart for DC Outputs



Equivalent Output Circuit



The eight AC input channels use terminals in the middle of the connector. Inputs are organized into two banks of four. Each bank has an isolated common terminal. The wiring example above shows all commons connected together, but separate supplies and common circuits may be used. The equivalent input circuit shows one channel of a typical bank.

The six current sinking DC output channels use terminals on the right side of the connector. All outputs actually share the same electrical common. Note the requirement for external power on the end (right-most) terminal. The equivalent output circuit shows one channel of the bank of six.

### D0-05AD General Specifications

External Power Requirements	95 – 240 VAC, 30 VA maximum,
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Slave) MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Master/Slave) MODBUS (Master/Slave) Non-sequence / print
Programming cable type	D2-DSCBL
Operating Temperature	32 to 131° F (0 to 55° C)
Storage Temperature	-4 to 158° F (-20 to 70° C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3-304
Terminal Type	Removable
Wire Gauge	One AWG16 or two AWG18, AWG24 minimum

### AC Input Specifications

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

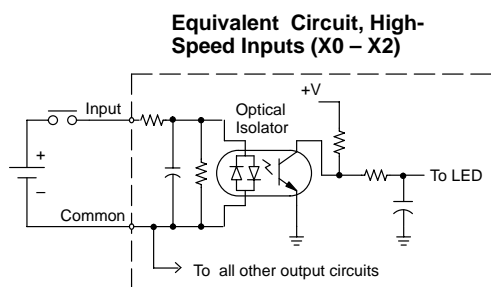
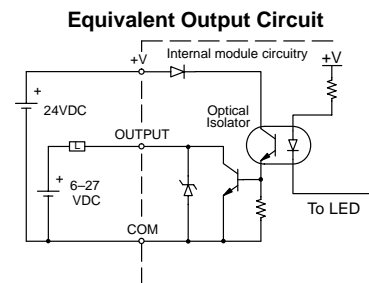
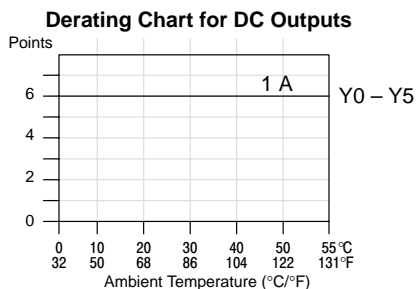
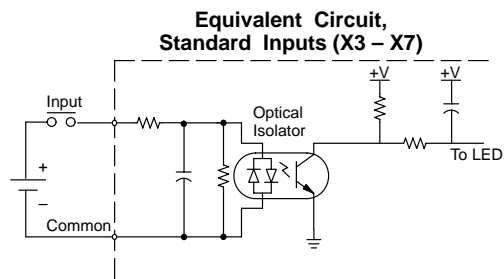
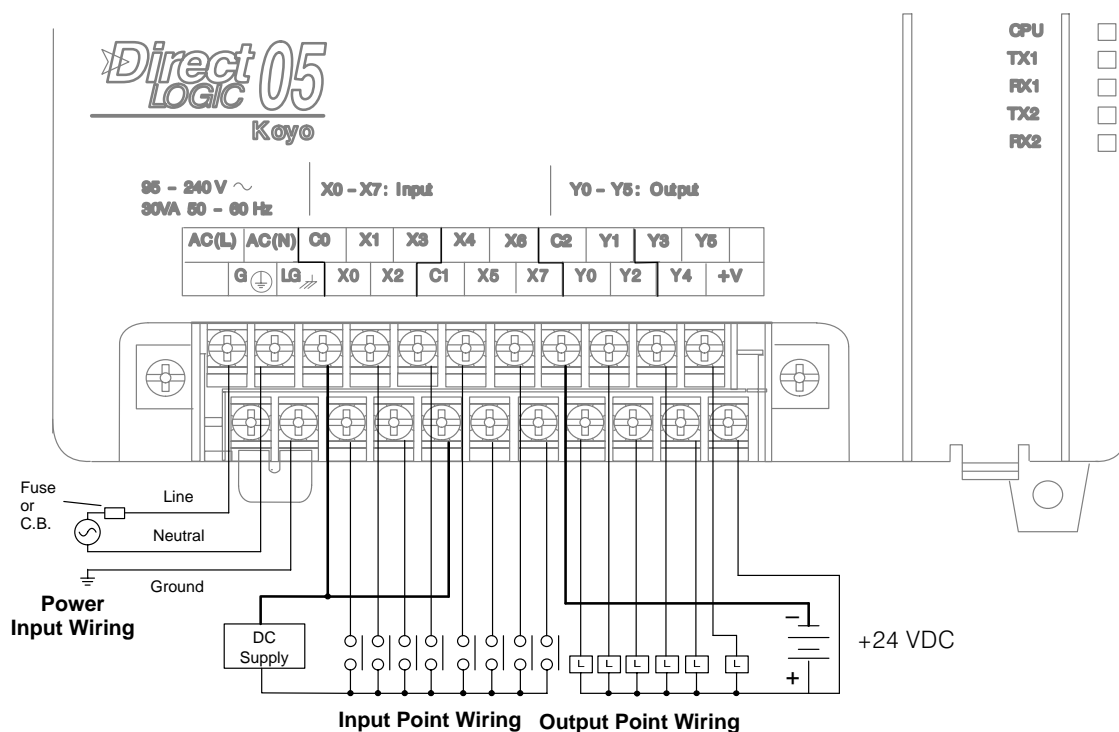
### DC Output Specifications

Parameter	Pulse Outputs, Y0 – Y1	Standard Outputs, Y2 – Y5
Min. - Max. Voltage Range	5 – 30 VDC	5 – 30 VDC
Operating Voltage	6 – 27 VDC	6 – 27 VDC
Peak Voltage	< 50 VDC (7 kHz max. frequency)	< 50 VDC
On Voltage Drop	0.3 VDC @ 1A	0.3 VDC @ 1A
Max Current (resistive)	0.5 A / pt. (1A / point for standard pt.)	1.0 A / point
Max leakage current	15 $\mu$ A @ 30 VDC	15 $\mu$ A @ 30 VDC
Max inrush current	2 A for 100 mS	2 A for 100 mS
External DC power required	20 - 28 VDC max 150mA	20 - 28 VDC max 150mA
OFF to ON Response	<10 $\mu$ S	< 10 $\mu$ S
ON to OFF Response	<30 $\mu$ S	< 60 $\mu$ S
Status Indicators	Logic Side	Logic Side
Commons	6 channels / common x 1 banks	
Fuses	None	None



## D0-05DD I/O Wiring Diagram

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



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

The six current sinking DC output channels use terminals on the right side of the connector. All outputs actually share the same electrical common. Note the requirement for external power on the end (right-most) terminal. The equivalent output circuit shows one channel of the bank of six.

## D0-05DD General Specifications

External Power Requirements	95 – 240 VAC, 30 VA maximum,
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Slave) MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Master/Slave) MODBUS (Master/Slave) Non-sequence / print
Programming cable type	D2-DSCBL
Operating Temperature	32 to 131° F (0 to 55° C)
Storage Temperature	-4 to 158° F (-20 to 70° C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3-304
Terminal Type	Removable
Wire Gauge	One AWG16 or two AWG18, AWG24 minimum

## DC Input Specifications

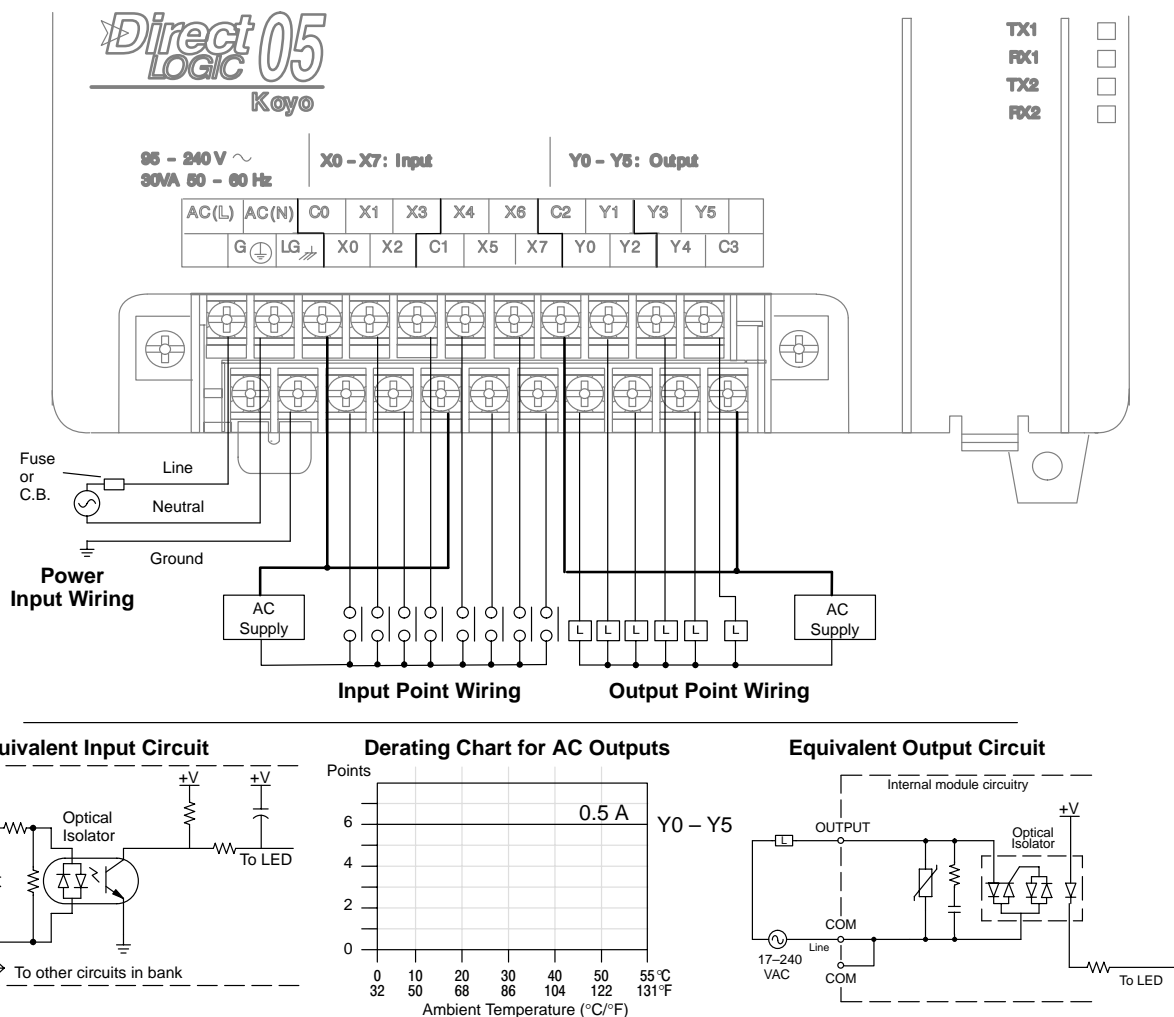
Parameter	High-Speed Inputs, X0 – X2	Standard DC Inputs X3 – X7
Min. - Max. Voltage Range	10.8 – 26.4 VDC	10.8 – 26.4 VDC
Operating Voltage Range	12 – 24 VDC	12 – 24 VDC
Peak Voltage	30 VDC (5 kHz maximum frequency)	30 VDC
Minimum Pulse Width	100 $\mu$ s	N/A
ON Voltage Level	> 9.0 VDC	> 9.0 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Max. Input Current	6mA @12VDC, 13mA @24VDC	4mA @12VDC, 8.5mA @24VDC
Input Impedance	1.8 k $\Omega$ @ 12 – 24 VDC	2.8 k $\Omega$ @ 12 – 24 VDC
Minimum ON Current	>5 mA	>4 mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<100 $\mu$ S	2 – 8 mS, 4 mS typical
ON to OFF Response	< 100 $\mu$ S	2 – 8 mS, 4 mS typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 2 banks	

## DC Output Specifications

Parameter	Pulse Outputs, Y0 – Y1	Standard Outputs, Y3 – Y5
Min. - Max. Voltage Range	5 – 30 VDC	5 – 30 VDC
Operating Voltage	6 – 27 VDC	6 – 27 VDC
Peak Voltage	< 50 VDC (7 kHz max. frequency)	< 50 VDC
On Voltage Drop	0.3 VDC @ 1 A	0.3 VDC @ 1 A
Max Current (resistive)	0.5 A / pt., 1A / pt. as standard pt.	1.0 A / point
Max leakage current	15 $\mu$ A @ 30 VDC	15 $\mu$ A @ 30 VDC
Max inrush current	2 A for 100 mS	2 A for 100 mS
External DC power required	20 - 28 VDC      Max 150mA	20 - 28 VDC      Max 150mA
OFF to ON Response	< 10 $\mu$ s	< 10 $\mu$ s
ON to OFF Response	< 30 $\mu$ s	< 60 $\mu$ s
Status Indicators	Logic Side	Logic Side
Commons	6 channels / common x 1 banks	
Fuses	None (external recommended)	

## D0-05AA I/O Wiring Diagram

The D0-05AA Micro PLC features eight AC inputs and six AC outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals at the left as shown.



The eight AC input channels use terminals in the middle of the connector. Inputs are organized into two banks of four. Each bank has an isolated common terminal. The wiring example above shows all commons connected together, but separate supplies and common circuits may be used. The equivalent input circuit shows one channel of a typical bank.

The six output channels use terminals on the right side of the connector. Outputs are organized into two banks of three triac switches. Each bank has a common terminal. The wiring example above shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank.

### D0-05AA General Specifications

External Power Requirements	95 – 240 VAC, 30 VA maximum,
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Slave) MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Master/Slave) MODBUS (Master/Slave) Non-sequence / print
Programming cable type	D2-DSCBL
Operating Temperature	32 to 131° F (0 to 55° C)
Storage Temperature	–4 to 158° F (–20 to 70° C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3-304
Terminal Type	Removable
Wire Gauge	One AWG16 or two AWG18, AWG24 minimum

### AC Input Specifications

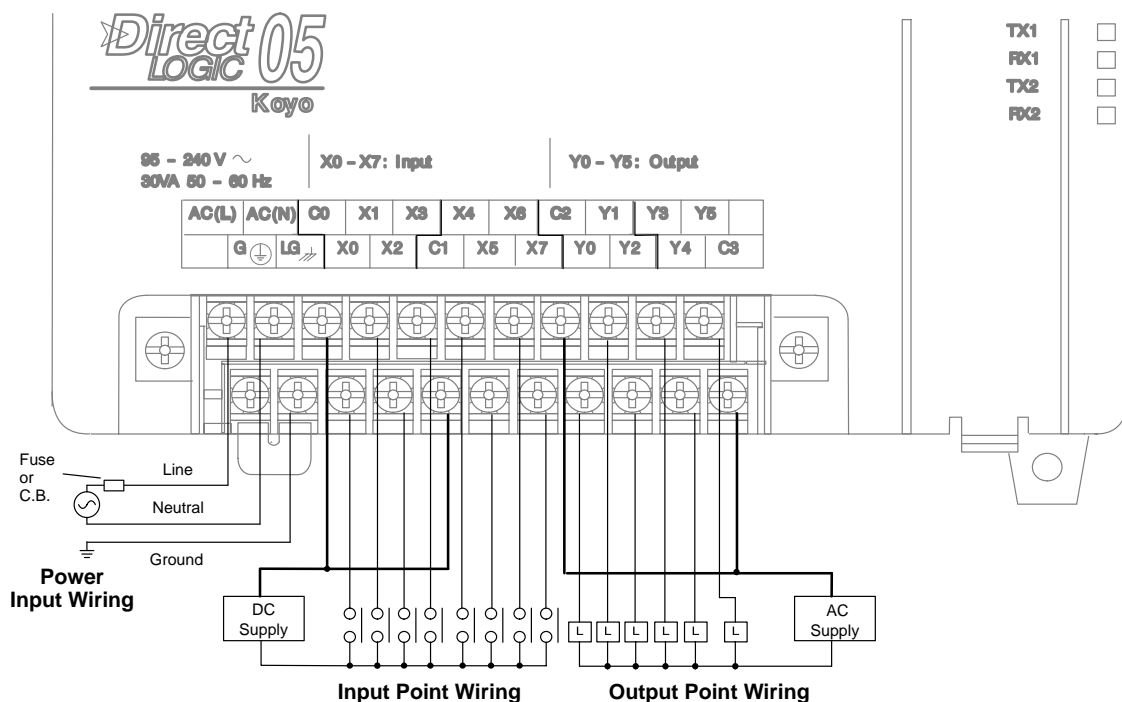
Input Voltage Range (Min. - Max.)	80 – 132 VAC, 47 - 63 Hz
Operating Voltage Range	90 – 120 VAC, 47 - 63 Hz
Input Current	8 mA @100 VAC at 50 Hz 10 mA @100 VAC at 60 Hz
Max. Input Current	12 mA @132 VAC at 50 Hz 15 mA @132 VAC at 60 Hz
Input Impedance	14K $\Omega$ @50 Hz, 12K $\Omega$ @60Hz
ON Current/Voltage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

### AC Output Specifications

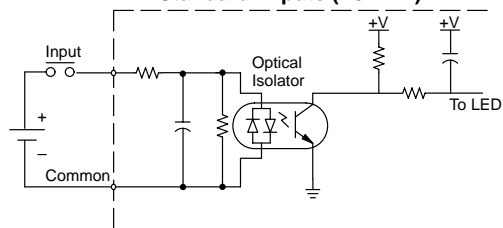
Output Voltage Range (Min. - Max.)	15 – 264 VAC, 47 – 63 Hz
Operating Voltage	17 – 240 VAC, 47 – 63 Hz
On Voltage Drop	1.5 VAC (>50mA) 4.0 VAC (<50mA)
Max Current	0.5 A / point, 1.5 A / common
Max leakage current	<4 mA @ 264 VAC
Max inrush current	10 A for 10 mS
Minimum Load	10 mA
OFF to ON Response	1 mS
ON to OFF Response	1 mS +1/2 cycle
Status Indicators	Logic Side
Commons	3 channels / common x 2 banks
Fuses	None (external recommended)

## D0-05DA I/O Wiring Diagram

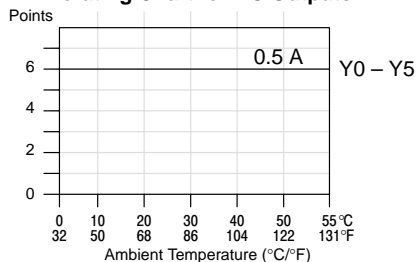
The D0-05DA Micro PLC features eight DC inputs and six AC outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals at the left as shown.



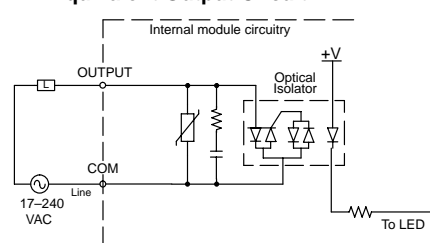
**Equivalent Circuit, Standard Inputs (X3 - X7)**



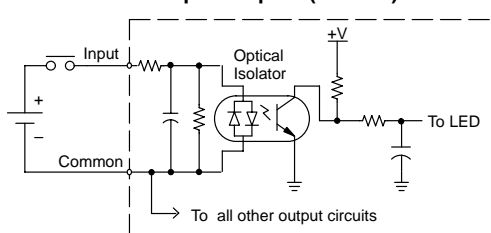
**Derating Chart for AC Outputs**



**Equivalent Output Circuit**



**Equivalent Circuit, High-Speed Inputs (X0 - X2)**



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

The six output channels use terminals on the right side of the connector. Outputs are organized into two banks of three triac switches. Each bank has a common terminal. The wiring example above shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank.

## D0-05DA General Specifications

External Power Requirements	95 – 240 VAC, 30 VA maximum,
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Slave) MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Master/Slave) MODBUS (Master/Slave) Non-sequence/print
Programming cable type	D2-DSCBL
Operating Temperature	32 to 131° F (0 to 55° C)
Storage Temperature	-4 to 158° F (-20 to 70° C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3-304
Terminal Type	Removable
Wire Gauge	One AWG16 or two AWG18, AWG24 minimum

## DC Input Specifications

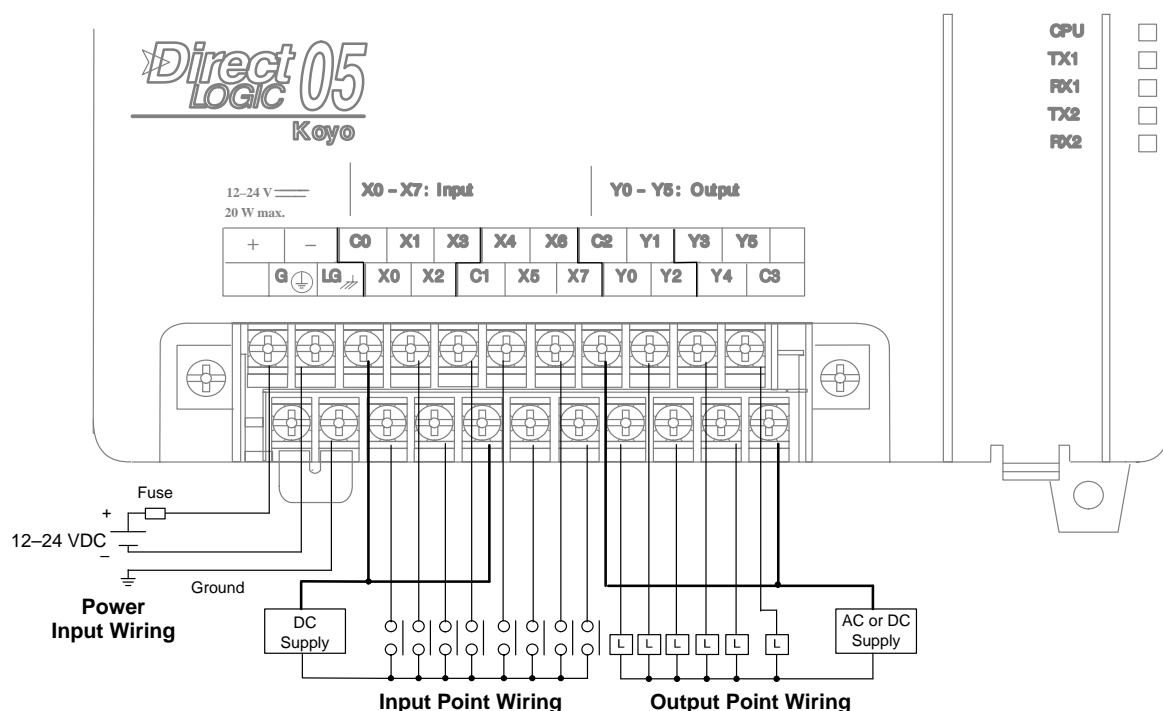
Parameter	High-Speed Inputs, X0 – X2	Standard DC Inputs X3 – X7
Input Voltage Range	10.8 – 26.4 VDC	10.8 – 26.4 VDC
Operating Voltage Range	12 – 24 VDC	12 – 24 VDC
Maximum Voltage	30 VDC (5 kHz maximum frequency)	30 VDC
Minimum Pulse Width	100 $\mu$ S	N/A
ON Voltage Level	> 10 VDC	> 10 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Input Impedance	1.8 k $\Omega$ @ 12 – 24 VDC	2.8 k $\Omega$ @ 12 – 24 VDC
Minimum ON Current	>5 mA	>4 mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<100 $\mu$ S	2 – 8 mS, 4 mS typical
ON to OFF Response	< 100 $\mu$ S	2 – 8 mS, 4 mS typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 2 bank	

## AC Output Specifications

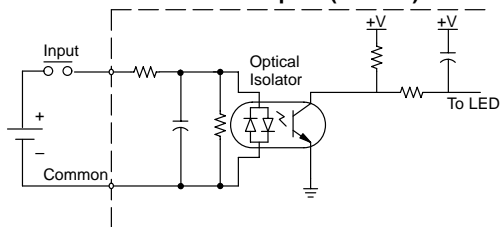
Output Voltage Range (Min. - Max.)	15 – 264 VAC, 47 – 63 Hz
Operating Voltage	17 – 240 VAC, 47 – 63 Hz
On Voltage Drop	1.5 VAC @ > 50mA, 4 VAC @ < 50mA
Max Current	0.5 A / point, 1.5 A / common
Max leakage current	< 4 mA @ 264 VAC, 60Hz
Max inrush current	10 A for 10 mS
Minimum Load	10 mA
OFF to ON Response	1 mS
ON to OFF Response	1 mS +1/2 cycle
Status Indicators	Logic Side
Commons	3 channels / common x 2 banks
Fuses	None (external recommended)

## D0-05DR-D I/O Wiring Diagram

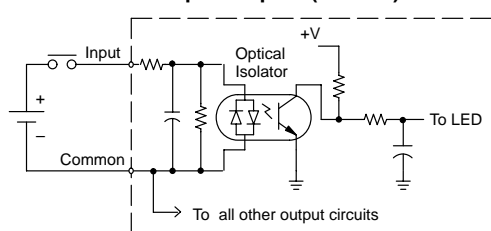
These micro PLCs feature eight DC inputs and six relay contact outputs. The following diagram shows a typical field wiring example. The DC external power connection uses three terminals at the left as shown.



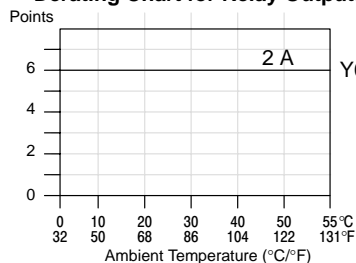
**Equivalent Circuit, Standard Inputs (X3 - X7)**



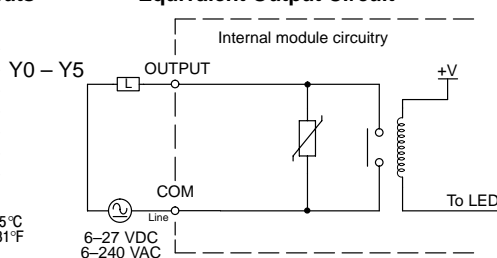
**Equivalent Circuit, High-Speed Inputs (X0 - X2)**



**Derating Chart for Relay Outputs**



**Equivalent Output Circuit**



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

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

## D0-05DR-D General Specifications

External Power Requirements	12 – 24 VDC, 20 W maximum,
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Slave) MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Master/Slave) MODBUS (Master/Slave) Non-sequence / print
Programming cable type	D2-DSCBL
Operating Temperature	32 to 131° F (0 to 55° C)
Storage Temperature	–4 to 158° F (–20 to 70° C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3-304
Terminal Type	Removable
Wire Gauge	One AWG16 or two AWG18, AWG24 minimum

## DC Input Specifications

Parameter	High-Speed Inputs, X0 – X2	Standard DC Inputs X3 – X7
Min. - Max. Voltage Range	10.8 – 26.4 VDC	10.8 – 26.4 VDC
Operating Voltage Range	12 -24 VDC	12 -24 VDC
Peak Voltage	30 VDC (5 kHz maximum frequency)	30 VDC
Minimum Pulse Width	100 µs	N/A
ON Voltage Level	> 10 VDC	> 10 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Input Impedance	1.8 kΩ @ 12 – 24 VDC	2.8 kΩ @ 12 – 24 VDC
Max. Input Current	6mA @12VDC 13mA @24VDC	4mA @12VDC 8.5mA @24VDC
Minimum ON Current	>5 mA	>4 mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<100 µs	2 – 8 mS, 4 mS typical
ON to OFF Response	< 100 µs	2 – 8 mS, 4 mS typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 2 bank	

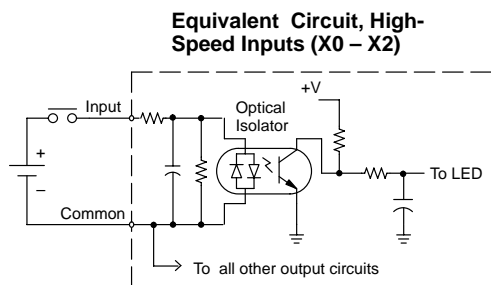
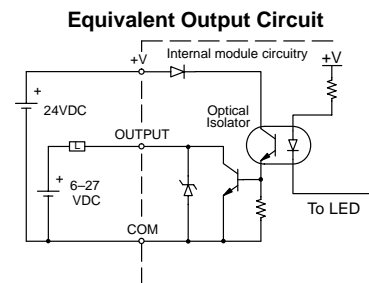
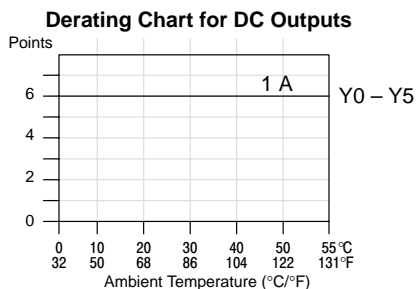
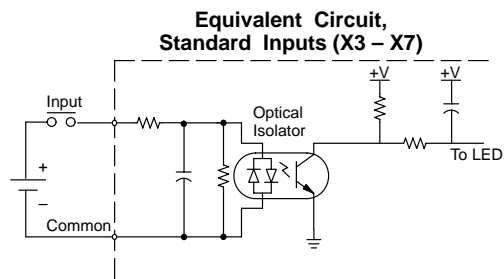
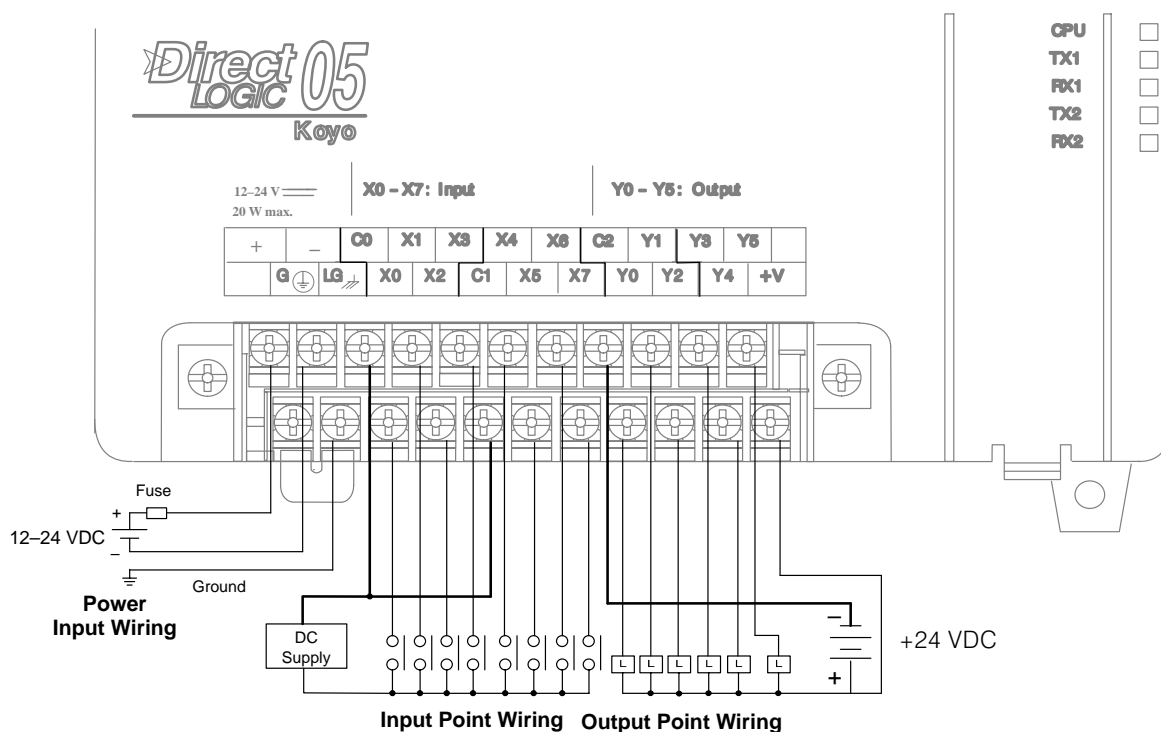
## Relay Output Specifications

Output Voltage Range (Min. - Max.)	5 -264 VAC (47 -63 Hz), 5 - 30 VDC
Operating Voltage	6 -240 VAC (47 -63 Hz), 6 - 27 VDC
Output Current	2A / point 6A / common
Maximum Voltage	264 VAC, 30 VDC
Max leakage current	0.1 mA @264 VAC
Smallest Recommended Load	5 mA
OFF to ON Response	< 15 mS
ON to OFF Response	< 10 mS
Status Indicators	Logic Side
Commons	3 channels / common x 2 banks
Fuses	None (external recommended)



## D0-05DD-D I/O Wiring Diagram

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



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

The six current sinking DC output channels use terminals on the right side of the connector. All outputs actually share the same electrical common. Note the requirement for external power on the end (right-most) terminal. The equivalent output circuit shows one channel of the bank of six.

## D0-05DD-D General Specifications

External Power Requirements	12 – 24 VDC, 20 W maximum,
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Slave) MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K-Sequence (Slave) DirectNET (Master/Slave) MODBUS (Master/Slave) Non-sequence / print
Programming cable type	D2-DSCBL
Operating Temperature	32 to 131° F (0 to 55° C)
Storage Temperature	–4 to 158° F (–20 to 70° C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One AWG16 or two AWG18, AWG24 minimum

## DC Input Specifications

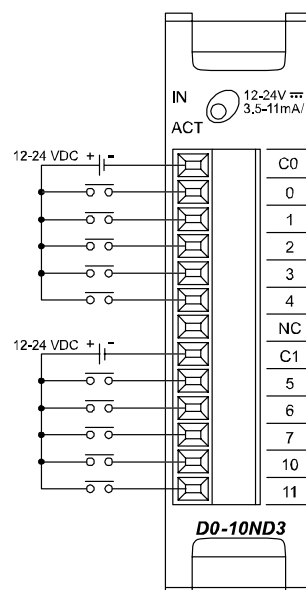
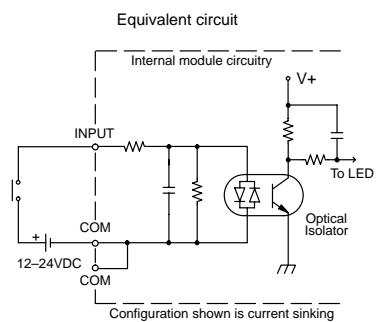
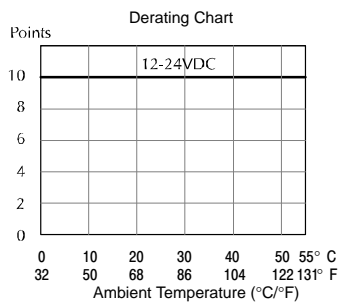
Parameter	High-Speed Inputs, X0 – X2	Standard DC Inputs X3 – X7
Min. - Max. Voltage Range	10.8 – 26.4 VDC	10.8 – 26.4 VDC
Operating Voltage Range	12 – 24 VDC	12 – 24 VDC
Peak Voltage	30 VDC (5 kHz maximum frequency)	30 VDC
Minimum Pulse Width	100 $\mu$ s	N/A
ON Voltage Level	> 9.0 VDC	> 9.0 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Max. Input Current	6mA @12VDC, 13mA @24VDC	4mA @12VDC, 8.5mA @24VDC
Input Impedance	1.8 k $\Omega$ @ 12 – 24 VDC	2.8 k $\Omega$ @ 12 – 24 VDC
Minimum ON Current	>5 mA	>4 mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<100 $\mu$ S	2 – 8 mS, 4 mS typical
ON to OFF Response	< 100 $\mu$ S	2 – 8 mS, 4 mS typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 2 banks	

## DC Output Specifications

Parameter	Pulse Outputs, Y0 – Y1	Standard Outputs, Y3 – Y5
Min. - Max. Voltage Range	5 – 30 VDC	5 – 30 VDC
Operating Voltage	6 – 27 VDC	6 – 27 VDC
Peak Voltage	< 50 VDC (7 kHz max. frequency)	< 50 VDC
On Voltage Drop	0.3 VDC @ 1 A	0.3 VDC @ 1 A
Max Current (resistive)	0.5 A / pt., 1A / pt. as standard pt.	1.0 A / point
Max leakage current	15 $\mu$ A @ 30 VDC	15 $\mu$ A @ 30 VDC
Max inrush current	2 A for 100 mS	2 A for 100 mS
External DC power required	20 - 28 VDC      Max 150mA	20 - 28 VDC      Max 150mA
OFF to ON Response	< 10 $\mu$ s	< 10 $\mu$ s
ON to OFF Response	< 30 $\mu$ s	< 60 $\mu$ s
Status Indicators	Logic Side	Logic Side
Commons	6 channels / common x 1 banks	
Fuses	None (external recommended)	

## D0-10ND3 DC Input

Inputs per module	10 (sink/source)
Input voltage range	10.8–26.4 VDC
Operating voltage range	12–24 VDC
Peak voltage	30.0 VDC
Input current	Typical: 4.0 mA @ 12 VDC 8.5 mA @ 24 VDC
Maximum input current	11 mA @ 26.4 VDC
Input impedance	2.8k $\Omega$ @ 12–24 VDC
ON voltage level	> 10.0 VDC
OFF voltage level	< 2.0 VDC
Minimum ON current	3.5 mA
Maximum OFF current	0.5 mA
OFF to ON response	2–8 ms, typical 4 ms
ON to OFF response	2–8 ms, typical 4 ms
Status indicators	Module activity: one green LED
Commons per module	2 non-isolated
Fuse	N/A
Base power required (5V)	Typical 35 mA (all pts. ON)

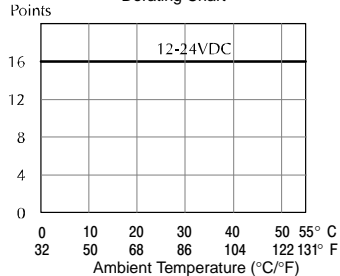


**Note:** The DL05 must have firmware version V4.10 (or later) for this module to function properly.

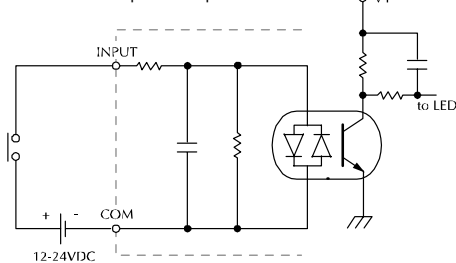
## D0-16ND3 DC Input

Inputs per module	16 (sink/source)
Input voltage range	20–28 VDC
Operating voltage range	24 VDC
Peak voltage	30.0 VDC
Input current	Typical: 4.0 mA @ 24 VDC
Maximum input current	6 mA @ 28 VDC
Input impedance	4.7k $\Omega$ @ 24 VDC
ON voltage level	> 19.0 VDC
OFF voltage level	< 7.0 VDC
Minimum ON current	3.5 mA
Maximum OFF current	1.5 mA
OFF to ON response	2–8 ms, typical 4 ms
ON to OFF response	2–8 ms, typical 4 ms
Status indicators	Module activity: one green LED
Commons per module	4 non-isolated
Fuse	N/A
Base power required	Typical 35 mA (all pts. ON)

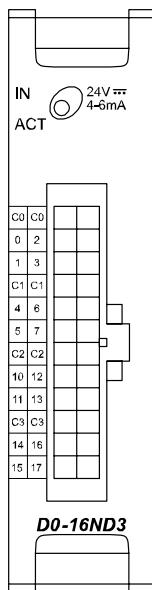
Derating Chart



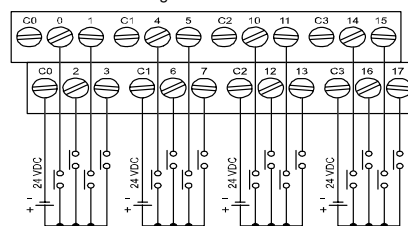
Equivalent input circuit



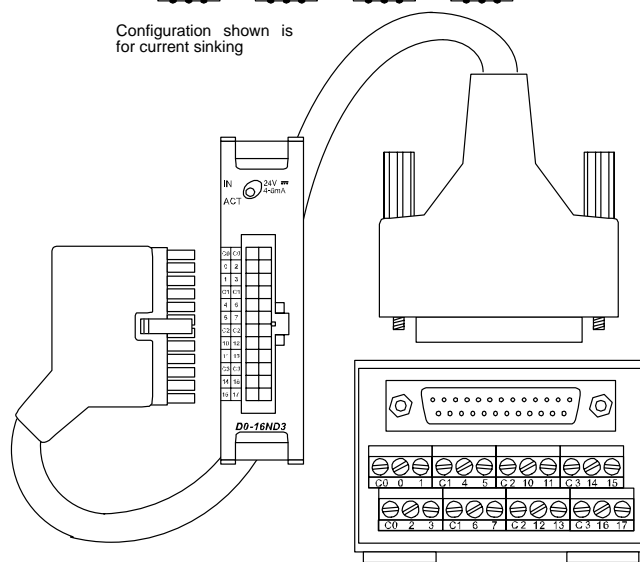
Configuration shown is for current sinking



Wiring for ZL-CM056



Configuration shown is for current sinking

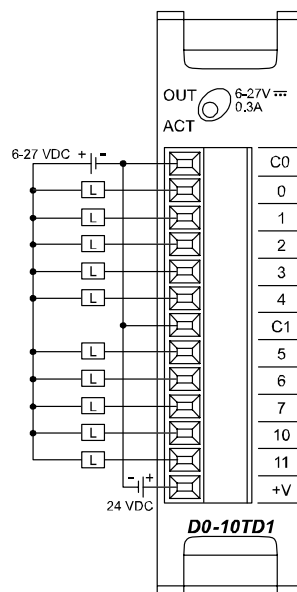
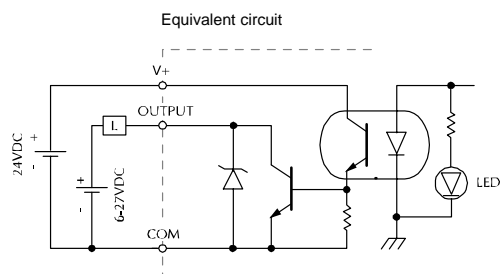
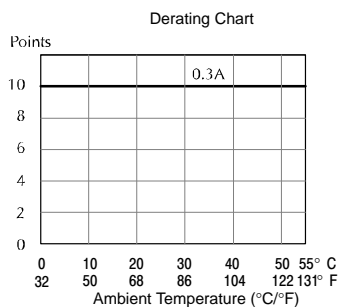


**Note:** The DL05 must have firmware version V4.10 (or later) for this module to function properly.

Use Ziplink ZL-CBL056 cable and ZL-CM056 connector module or build your own cables using 24-pin Molex Micro Fit 3.0 receptacle, part number 43025, or compatible.

## D0-10TD1 DC Output

Outputs per module	10 (sinking)
Operating voltage range	6–27 VDC
Output voltage range	5–30 VDC
Peak voltage	50.0 VDC
Maximum output current	0.3 A/point, 1.5 A/common
Minimum output current	0.5 mA
Maximum leakage current	15 $\mu$ A @ 30.0 VDC
ON voltage drop	0.5 VDC @ 0.3 A
Maximum inrush current	1 A for 10 ms
OFF to ON response	< 10 $\mu$ s
ON to OFF response	< 60 $\mu$ s
Status indicators	Module activity: one green LED
Commons per module	2 non-isolated (5 points/common)
Fuse	N/A
External DC power required	20–28 VDC max. 200 mA (all pts. on)
Base power required (5V)	Max. 150 mA (all pts. ON)

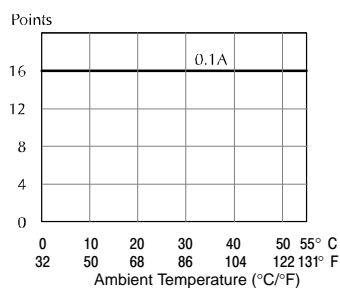


**Note:** The DL05 must have firmware version V4.10 (or later) for this module to function properly.

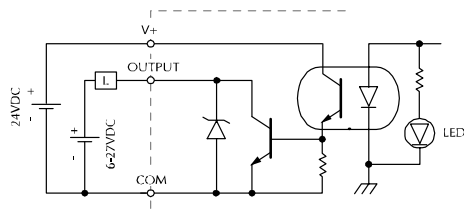
## D0-16TD1 DC Output

Outputs per module	16 (sinking)
Operating voltage range	6–27 VDC
Output voltage range	5–30 VDC
Peak Voltage	50.0 VDC
Maximum output current	0.1 A/point, 0.8 A/common
Minimum output current	0.5 mA
Maximum leakage current	15 $\mu$ A @ 30.0 VDC
On voltage drop	0.5 VDC @ 0.1 A
Maximum inrush current	1 A for 10 ms
OFF to ON response	< 0.5 ms
ON to OFF response	< 0.5 ms
Status indicators	Module activity: one green LED
Commons per module	2 isolated (8 points/common)
Fuse	N/A
External DC power required	20–28 VDC max. 70 mA (all pts. on)
Base power required (5V)	Max. 200 mA (all pts. ON)

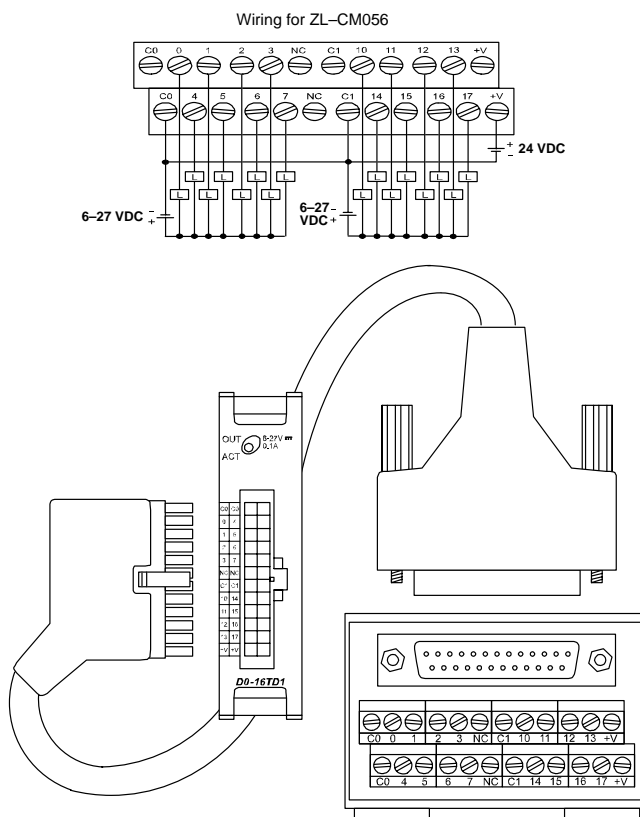
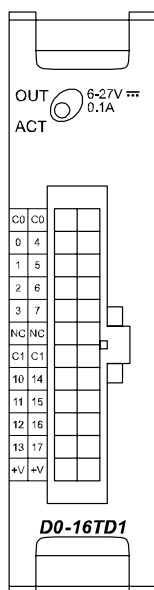
Derating Chart



Equivalent input circuit



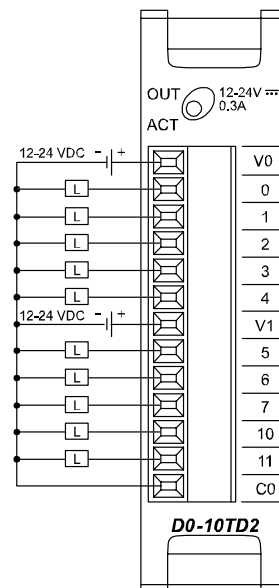
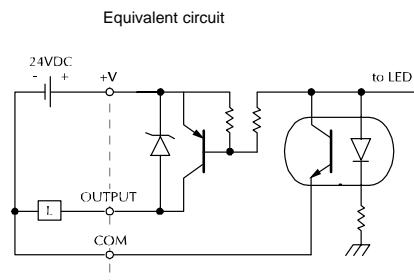
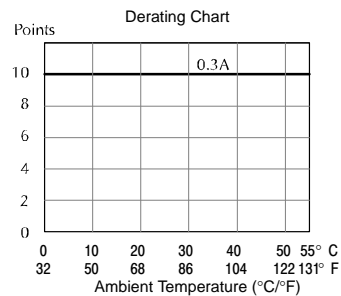
**Note:** The DL05 must have firmware version V4.10 (or later) for this module to function properly.



Use Ziplugin ZL-CBL056 cable and ZL-CM056 connector module or build your own cables using 24-pin Molex Micro Fit 3.0 receptacle, part number 43025, or compatible.

## D0-10TD2 DC Output

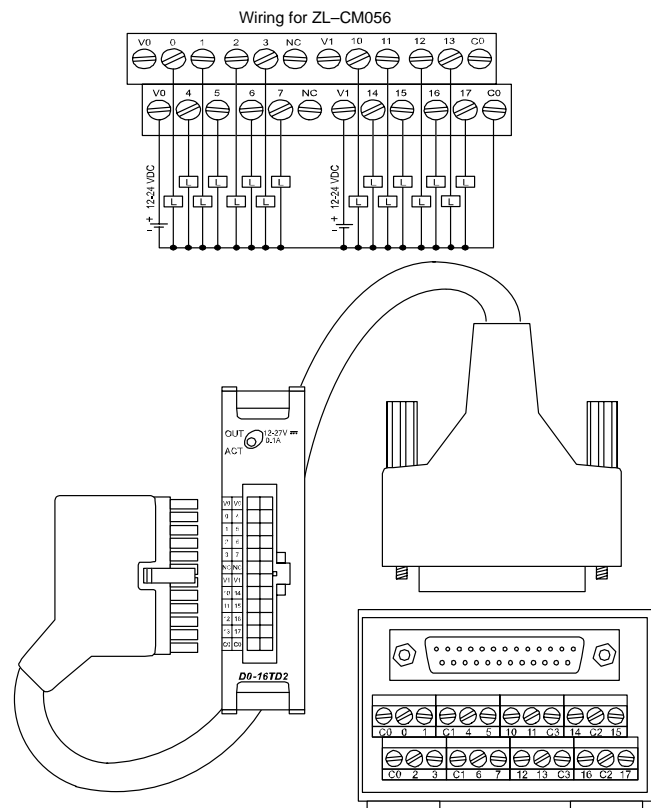
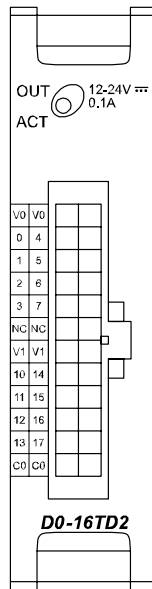
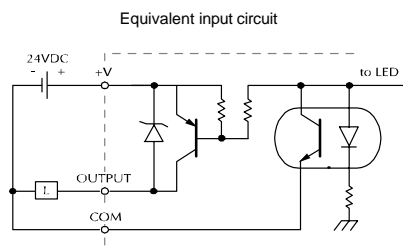
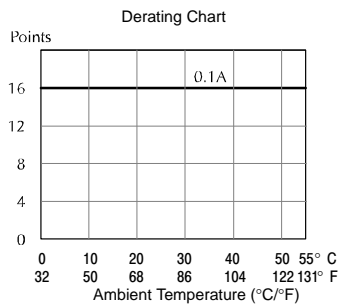
Outputs per module	10 (sourcing)
Operating voltage range	12–24 VDC
Output voltage range	10.8–26.4 VDC
Peak voltage	50.0 VDC
Maximum output current	0.3 A/point, 1.5 A/common
Minimum output current	0.5 mA
Maximum leakage current	1.5 $\mu$ A @ 30.0 VDC
ON voltage drop	1.0 VDC @ 0.3 A
Maximum inrush current	1 A for 10 ms
OFF to ON response	< 10 $\mu$ s
ON to OFF response	< 60 $\mu$ s
Status indicators	Module activity: one green LED
Commons per module	2 non-isolated (5 points/common)
Fuse	N/A
Base power required (5V)	Max. 150 mA (all pts. ON)



**Note:** The DL05 must have firmware version V4.10 (or later) for this module to function properly.

## D0-16TD2 DC Output

Outputs per module	16 (sourcing)
Operating voltage range	12–24 VDC
Output voltage range	10.8–26.4 VDC
Peak Voltage	50.0 VDC
Maximum output current	0.1 A/point, 0.8 A/common
Minimum output current	0.5 mA
Maximum leakage current	1.5 $\mu$ A @ 26.4 VDC
ON voltage drop	1.0 VDC @ 0.3 A
Maximum inrush current	1 A for 10 ms
OFF to ON response	< 0.5 ms
ON to OFF response	< 0.5 ms
Status indicators	Module activity: one green LED
Commons per module	2 non-isolated (8 points/common)
Fuse	N/A
Base power required (5V)	Max. 200 mA (all pts. ON)



**Note:** The DL05 must have firmware version V4.10 (or later) for this module to function properly.

Use Ziplugin ZL-CBL056 cable and ZL-CM056 connector module or build your own cables using 24-pin Molex Micro Fit 3.0 receptacle, part number 43025, or compatible.

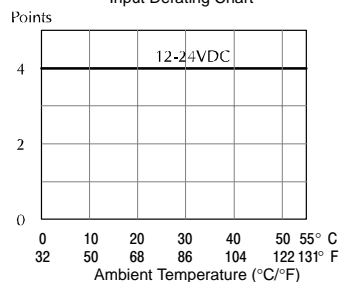


## D0-07CDR DC Input and Output

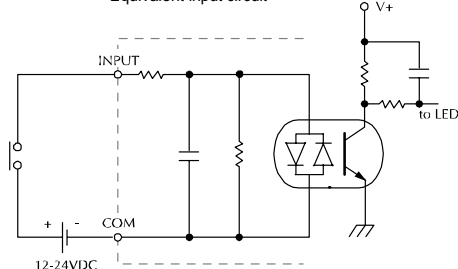
Input Specification	
Inputs per module	4 (sink/source)
Operating voltage range	12–24 VDC
Input voltage range	10.8–26.4 VDC
Peak voltage	30.0 VDC
Maximum input current	11 mA @ 26.4 VDC
Input current	Typical: 4 mA @ 12 VDC 8.5 mA @ 24 VDC
Input impedance	2.8k $\Omega$ @ 12–24 VDC
ON voltage level	> 10.0 VDC
OFF voltage level	< 2.0 VDC
Minimum ON current	3.5 mA
Maximum OFF current	0.5 mA
ON to OFF response	2–8 ms, typical 4 ms
OFF to ON response	2–8 ms, typical 4 ms
Commons	1 (4 points / common)

Output Specification	
Outputs per module	3
Operating voltage range	6–27 VDC / 6–240 VAC
Output type	Relay, form A, SPST
Peak voltage	30.0 VDC/264 VAC
Maximum current (resistive)	1 A/point, 4 A/common
Minimum load current	5 mA @ 5 VDC
Maximum leakage current	0.1 mA @ 264 VAC
ON voltage drop	N/A
Maximum inrush current	Output: 3 A for 10 ms Common: 10 A for 10 ms
OFF to ON response	< 15 ms
ON to OFF response	< 10 ms
Status indicators	Module activity: one green LED
Commons	1 (3 points/common)
Fuse	N/A
Base power required (5 V)	Max. 200 mA (all points ON)

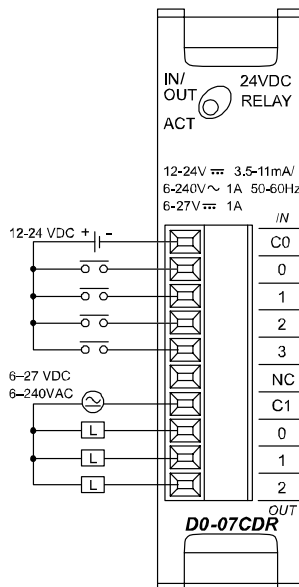
Input Derating Chart



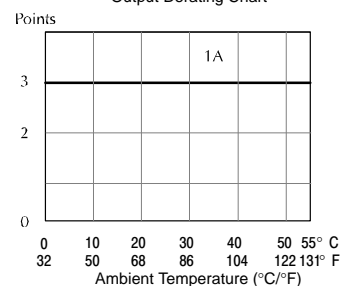
Equivalent input circuit



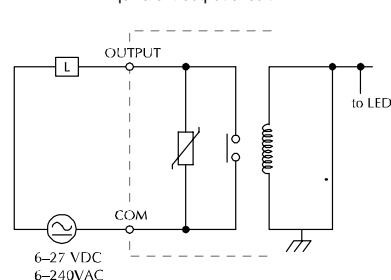
Configuration shown is for current sinking



Output Derating Chart

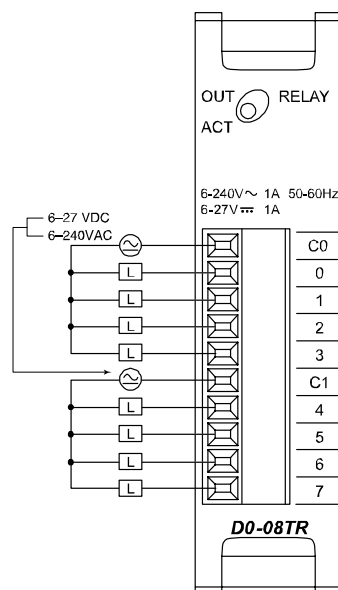
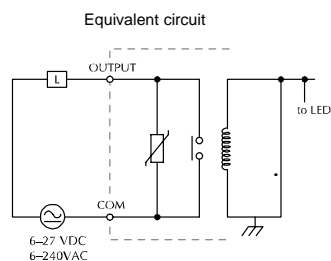
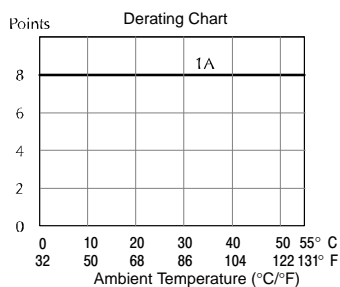


Equivalent output circuit

**Note:** The DL05 must have firmware version V4.10 (or later) for this module to function properly.

## D0-08TR Relay Output

Outputs per module	8
Operating voltage range	6-27 VDC/6-240 VAC
Output type	Relay, form A, SPST
Peak voltage	30.0 VDC/264 VAC
Maximum current (resistive)	1 A/point, 4 A/common
Minimum load current	0.5 mA
Maximum leakage current	0.1 mA @ 264 VAC
ON voltage drop	N/A
Maximum inrush current	Output: 3 A for 10 ms Common: 10 A for 10 ms
OFF to ON response	< 15 ms
ON to OFF response	< 10 ms
Status indicators	Module activity: one green LED
Commons per module	2 isolated (4 points/common)
Fuse	N/A
Base power required (5 V)	Maximum 280 mA (all pts. ON)

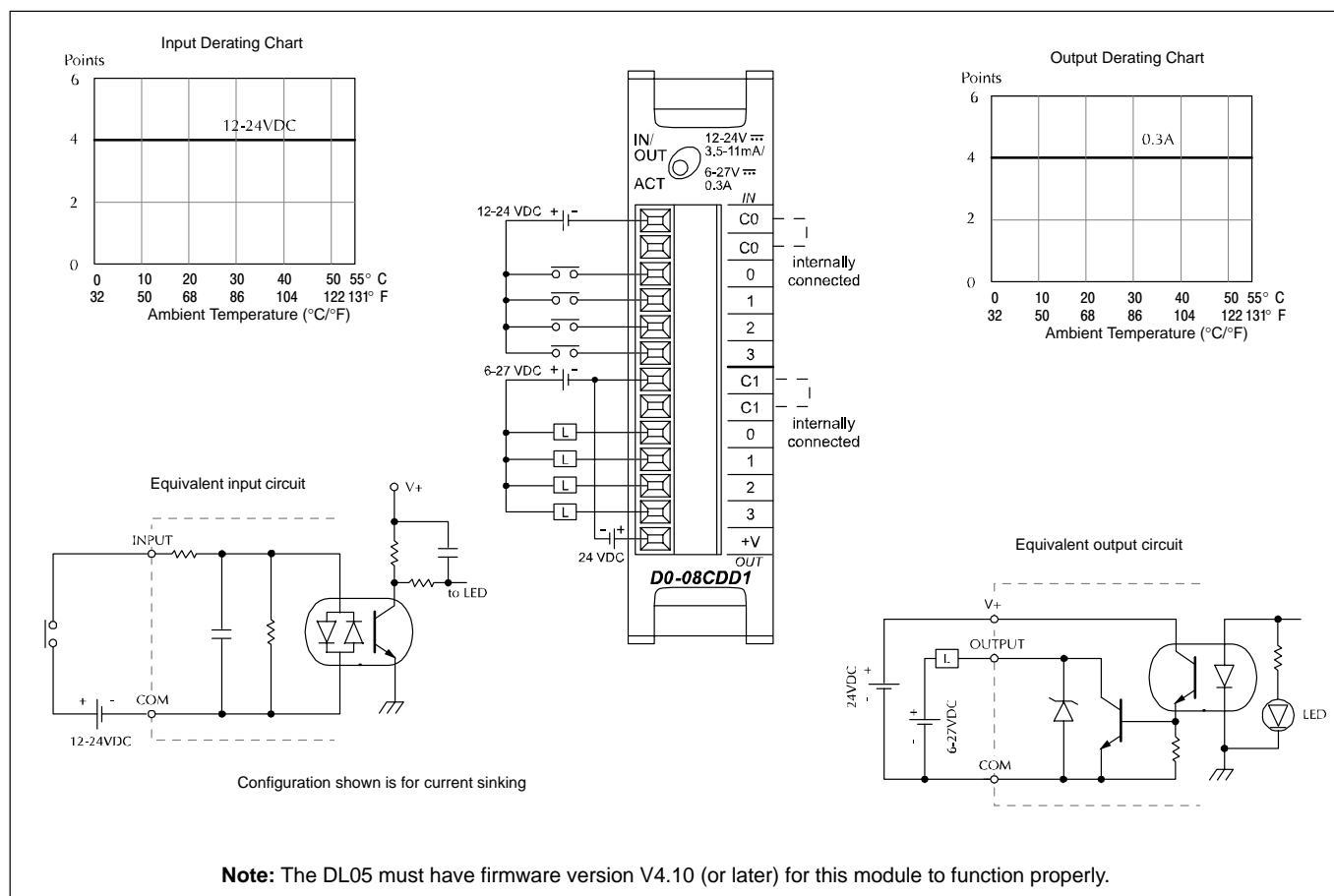


**Note:** The DL05 must have firmware version V4.10 (or later) for this module to function properly.

## D0-08CDD1 DC Input and Output

Input Specification	
Inputs per module	4 (sink/source)
Operating voltage range	10.8–26.4 VDC
Input voltage range	12–24 VDC
Peak voltage	30.0 VDC
Maximum input current	11 mA @ 26.4 VDC
Input current	Typical: 4 mA @ 12 VDC 8.5 mA @ 24 VDC
Input impedance	2.8k $\Omega$ @ 12–24 VDC
ON voltage level	> 10.0 VDC
OFF voltage level	< 2.0 VDC
Minimum ON current	3.5 mA
Maximum OFF current	0.5 mA
ON to OFF response	2–8 ms, typical 4 ms
OFF to ON response	2–8 ms, typical 4 ms
Commons	2 non-isolated (4 pts./common)

Output Specification	
Outputs per module	4 (sinking)
Operating voltage range	6–27 VDC
Output voltage range	5–30 VDC
Peak voltage	50.0 VDC
Maximum output current	0.3 A/point, 1.2 A/common
Minimum output current	0.5 mA
Maximum leakage current	1.5 $\mu$ A @ 30.0 VDC
ON voltage drop	0.5 VDC @ 0.3 A
Maximum inrush current	1 A for 10 ms
OFF to ON response	< 10 $\mu$ s
ON to OFF response	< 60 $\mu$ s
Status indicators	Module activity: one green LED
Commons	2 non-isolated (4 pts./common)
Fuse	N/A
Base power required (5 V)	Max. 200 mA (all points ON)
External DC power required (24V)	20–28 VDC, maximum 80 mA (all pts. ON)



## I/O Addressing

### Module I/O Points and Addressing

Each option module has a set number of I/O points. This holds true for both the discrete modules and the analog modules. The following chart shows the number of I/O points per module when used in the DL05 PLC.

DC Input Modules	I/O Points	Slot 1 I/O Address
D0-10ND3	10 Input	X100 – X107 and X110 – X111
D0-16ND	16 Input	X100 – X107 and X110 – X117
DC Output Modules	I/O Points	Slot 1 I/O Address
D0-10TD1	10 Output	Y100 – Y107 and Y110 – Y111
D0-16TD1	16 Output	Y100 – Y107 and Y110 – Y117
D0-10TD2	10 Output	Y100 – Y107 and Y110 – Y111
D0-16TD2	16 Output	Y100 – Y107 and Y110 – Y117
Relay Output Modules	I/O Points	Slot 1 I/O Address
D0-08TR	8 Output	Y100 – X107
Combination Modules	I/O Points	Slot 1 I/O Address
D0-07CDR	4 Input, 3 Output	X100 – X103 and Y100 – Y102
D0-08CDD1	4 Input, 4 Output	X100 – X103 and Y100 – Y103

# High-Speed Input and Pulse Output Features

---

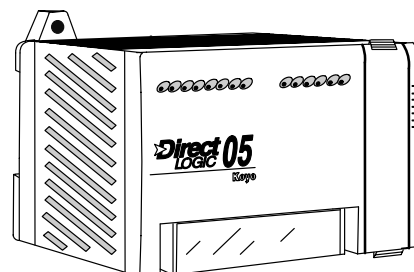
In This Chapter. . . .

- Introduction
  - Choosing the HSIO Operating Mode
  - Mode 10: High-Speed Counter
  - Mode 20: Quadrature Counter
  - Mode 30: Pulse Output
  - Mode 40: High-Speed Interrupt
  - Mode 50: Pulse Catch Input
  - Mode 60: Filtered Inputs
-

## Introduction

### Built-in Motion Control Solution

Many machine control applications require various types of simple high-speed monitoring and control. These applications usually involve some type of motion control, or high-speed interrupts for time-critical events. The DL05 Micro PLC solves this traditionally expensive problem with built-in CPU enhancements. Let's take a closer look at the available high-speed I/O features.



The available **high-speed input features** are:

- High Speed Counter (5 kHz max.) with up to 24 counter presets and built-in interrupt subroutine, counts up only, with reset
- Quadrature encoder inputs to measure counts and clockwise or counter clockwise direction (5 kHz max.), counts up or down, with reset
- High-speed interrupt input for immediate response to critical or time-sensitive tasks
- Pulse catch feature to monitor one input point, having a pulse width as small as 100µS (0.1ms)
- Programmable discrete filtering (both on and off delay up to 99ms) to ensure input signal integrity (this is the default mode for inputs X0–X2)

The available **pulse output features** are:

- Single-axis programmable pulse output (7 kHz max.) with three profile types, including trapezoidal moves, registration, and velocity control

**IMPORTANT:** Please note the following restrictions on availability of features:

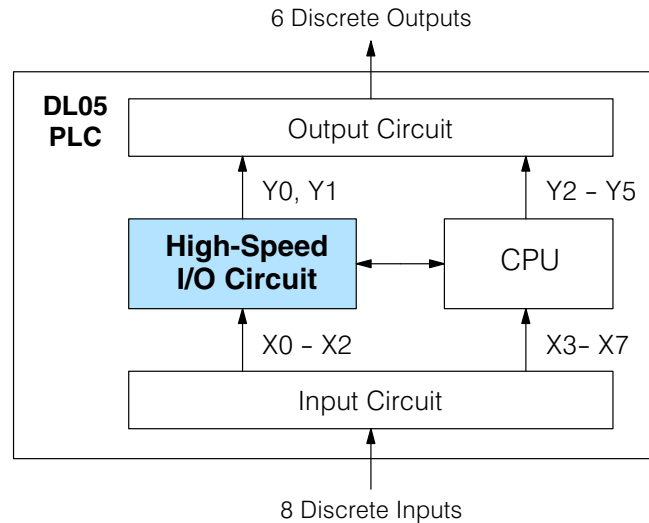
- High-speed input options are available only on DL05s with DC inputs.
- Pulse output options are available only on DL05s with DC outputs.
- Only one HSIO feature may be in use at one time. You cannot use a high-speed input feature and the pulse output at the same time.

### Availability of HSIO Features

DL05 Part Number	Discrete Input Type	Discrete Output Type	High-Speed Input	Pulse Output
D0-05AR	AC	Relay	No	No
D0-05DR	<b>DC</b>	Relay	<b>Yes</b>	No
D0-05AD	AC	<b>DC</b>	No	<b>Yes</b>
D0-05DD	<b>DC</b>	<b>DC</b>	<b>Yes</b>	<b>Yes</b>
D0-05AA	AC	AC	No	No
D0-05DA	<b>DC</b>	AC	<b>Yes</b>	No
D0-05DR-D	<b>DC</b>	Relay	<b>Yes</b>	No
D0-05DD-D	<b>DC</b>	<b>DC</b>	<b>Yes</b>	<b>Yes</b>

## Dedicated High-Speed I/O Circuit

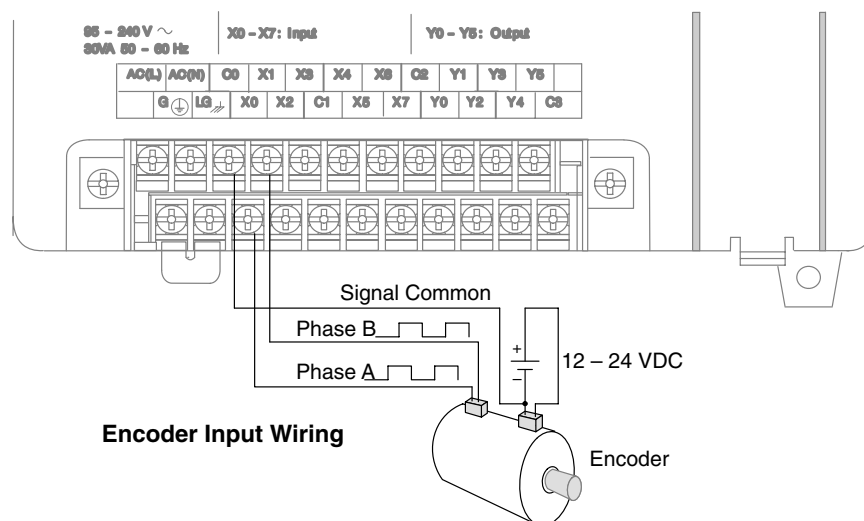
The internal CPU's main task is to execute the ladder program and read/write all I/O points during each scan. In order to service high-speed I/O events, the DL05 includes a special circuit which is dedicated to a portion of the I/O points. Refer to the DL05 block diagram in the figure below.



The high-speed I/O circuit (HSIO) is dedicated to the first three inputs (X0 – X2) and the first two outputs (Y0 – Y1). We might think of this as a “CPU helper”. In the default operation (called “Mode 60”) the HSIO circuit just passes through the I/O signals to or from the CPU, so that all eight inputs behave equally and all six outputs behave equally. When the CPU is configured in any other HSIO Mode, the HSIO circuit imposes a specialized function on the portion of inputs and outputs shown. The HSIO circuit *operates independently of the CPU program scan*. This provides accurate measurement and capturing of high-speed I/O activity while the CPU is busy with ladder program execution.

## Wiring Diagrams for Each HSIO Mode

After choosing the appropriate HSIO mode for your application, you’ll need to refer to the section in this chapter for that specific mode. Each section includes wiring diagram(s) to help you connect the High-Speed I/O points correctly to field devices. An example of the quadrature counter mode diagram is shown below.



## Choosing the HSIO Operating Mode

### Understanding the Six Modes

The High-Speed I/O circuit operates in one of 6 basic modes as listed in the table below. The number in the left column is the mode number (later, we'll use these numbers to configure the PLC). Choose one of the following modes according to the primary function you want from the dedicated High-Speed I/O circuit. You can simply use all eight inputs and six outputs as regular I/O points with Mode 60.

Mode Number	Mode Name	Mode Features
10	High-Speed Counter	5 kHz counter with 24 presets and reset input, counts up only, causes interrupt on preset
20	Quadrature Counter	Channel A / Channel B 5 kHz quadrature input, counts up and down
30	Pulse Output	Stepper control – pulse and direction signals, programmable motion profile (7kHz max.)
40	High-Speed Interrupt	Generates an interrupt based on input transition or time
50	Pulse Catch	Captures narrow pulses on a selected input
60	Discrete/Filtered Input	Rejects narrow pulses on selected inputs

In choosing one of the six high-speed I/O modes, the I/O points listed in the table below operate only as the function listed. If an input point is not specifically used to support a particular mode, it usually operates as a filtered input by default. Similarly, output points operate normally unless Pulse Output mode is selected.

Physical I/O Point Usage					
Mode	DC Input Points			DC Output Points	
	X0	X1	X2	Y0	Y1
High-Speed Counter	Counter clock	Filtered Input	Filtered Input or Reset Cnt	Regular Output	Regular Output
Quadrature Counter	Phase A Input	Phase B Input	Filtered Input or Reset Cnt	Regular Output	Regular Output
High-Speed Interrupt	Interrupt Input	Filtered Input	Filtered Input	Regular Output	Regular Output
Pulse Catch	Pulse Input	Filtered Input	Filtered Input	Regular Output	Regular Output
Pulse Output	Filtered Input	Filtered Input	Filtered Input,	Pulse or CW Pulse	Direction or CCW Pulse
Filtered Input	Filtered Input	Filtered Input	Filtered Input	Regular Output	Regular Output

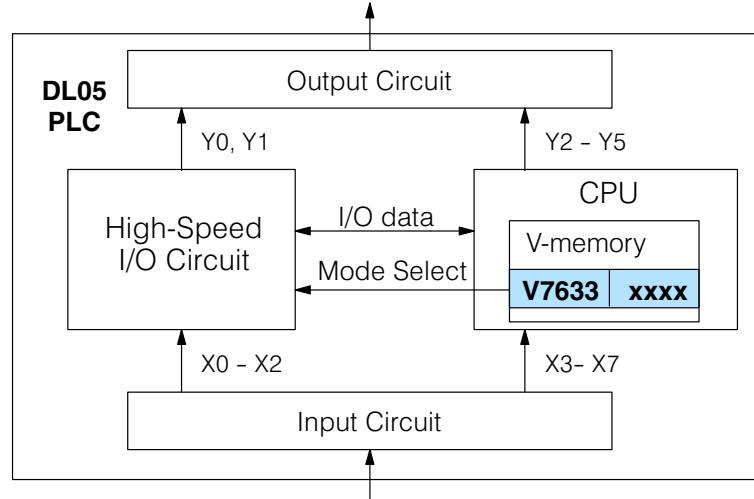
### Default Mode

Mode 60 (Filtered Inputs) is the default mode. The DL05 is initialized to this mode at the factory, and any time you reset V-memory scratchpad. In the default condition, X0–X2 are filtered inputs (10 mS delay) and Y0–Y1 are standard outputs.

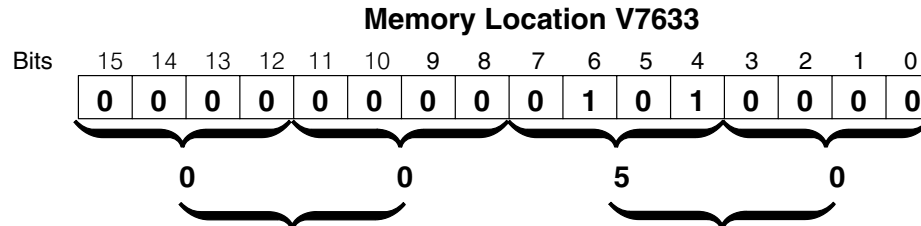


## Configuring the HSIO Mode

If you have chosen a mode suited to the high-speed I/O needs of your application, we're ready to proceed to configure the PLC to operate accordingly. In the block diagram below, notice the V-memory detail in the expanded CPU block. V-memory location V7633 determines the functional mode of the high-speed I/O circuit. *This is the most important V-memory configuration value for HSIO functions!*



The contents of V7633 is a 16-bit word, to be entered in binary-coded decimal. The figure below defines what each 4-bit BCD digit of the word represents.



Bits 8 - 15 are not used in V7633.

### HSIO Mode Setup (BCD)

00 = Not Used

10 = High-Speed Counting Mode

20 = Quadrature Counting Mode

30 = Pulse Output Train

40 = High-Speed Interrupts

50 = Pulse Catching

60 = Discrete Filtered Inputs (default)

Bits 0 – 7 define the mode number 00.. 60 previously referenced in this chapter. The example data “2050” shown selects Mode 50 – Pulse Catch (BCD = 50). The DL05 PLC ignores bits 8 - 15 in V7633.

## Configuring Inputs X0 – X2

In addition to configuring V7633 for the HSIO mode, you'll need to program the next three locations in certain modes according to the desired function of input points X0 – X2. Other memory locations may require configuring, depending on the HSIO mode (see the corresponding section for particular HSIO modes).

V-memory		
Mode	V7633	xxxx
X0	V7634	xxxx
X1	V7635	xxxx
X2	V7636	xxxx

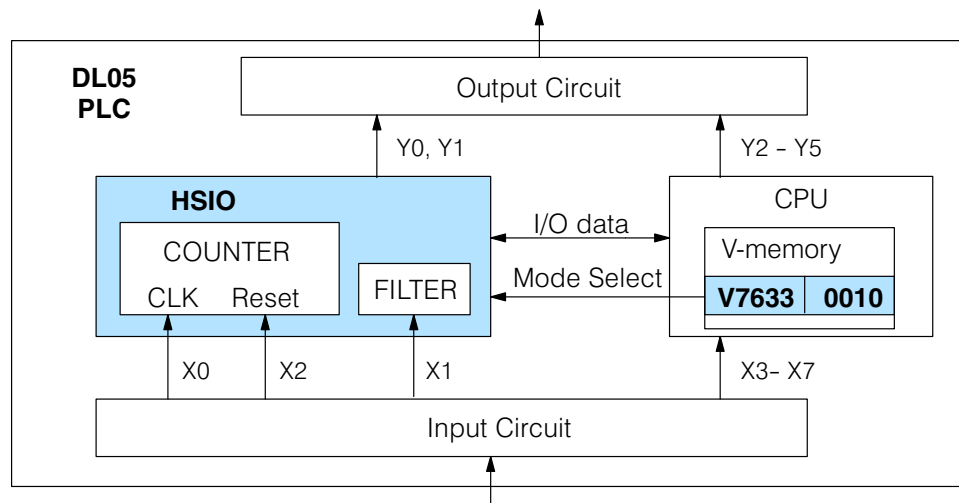
## Mode 10: High-Speed Counter

### Purpose

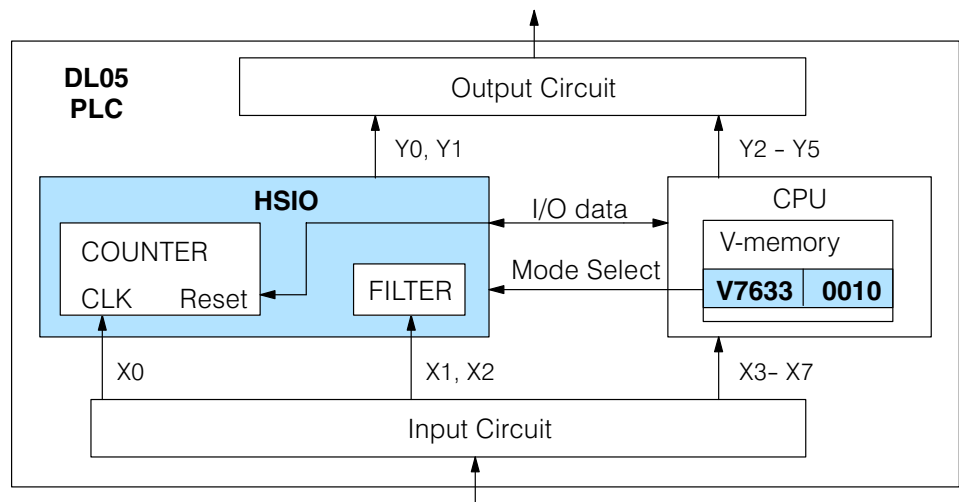
The HSIO circuit contains one high-speed counter. A single pulse train from an external source (X0) clocks the counter on each signal leading edge. The counter counts only upwards, from 0 to 99999999. The counter compares the current count with up to 24 preset values, which you define. The purpose of the presets is to quickly cause an action upon arrival at specific counts, making it ideal for such applications as cut-to-length. It uses counter registers CT76 and CT77 in the CPU.

### Functional Block Diagram

Refer to the block diagram below. When the lower byte of HSIO Mode register V7633 contains a BCD “10”, the high-speed up counter in the HSIO circuit is enabled. X0 automatically becomes the “clock” input for the high-speed counter, incrementing it upon each off-to-on transition. The external reset input on X2 is the default configuration for Mode 10. Input X1 is the filtered input, available to the ladder program.



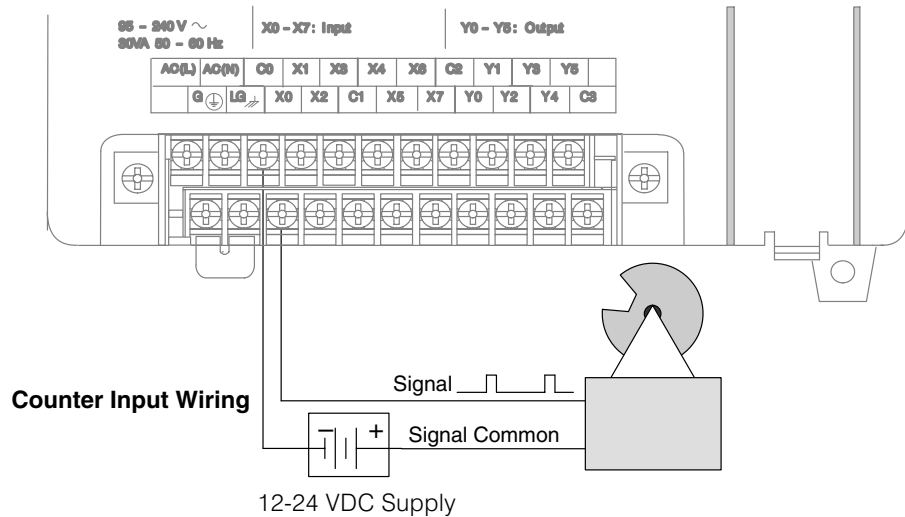
Instead of using X2 as a dedicated reset input, you can configure X2 as a normal filtered input. In this way, the counter reset must be generated in ladder logic.



Next, we will discuss how to program the high-speed counter and its presets.

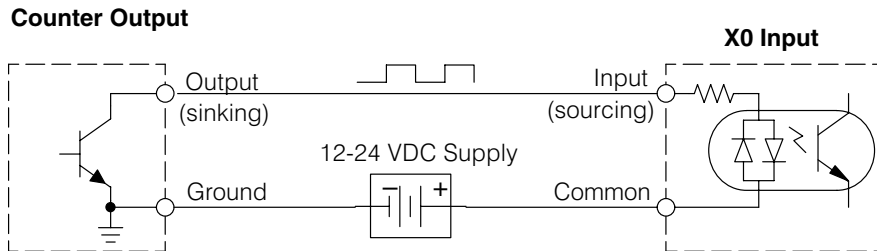
## Wiring Diagram

A general wiring diagram for counters/encoders to the DL05 in HSIO Mode 10 is shown below. Many types of pulse-generating devices may be used, such as proximity switches, single-channel encoders, magnetic or optical sensors, etc. Devices with sinking outputs (NPN open collector) are probably the best choice for interfacing. If the counter sources to the inputs, it must output 12 to 24 VDC. Note that devices with 5V sourcing outputs will not work with DL05 inputs.

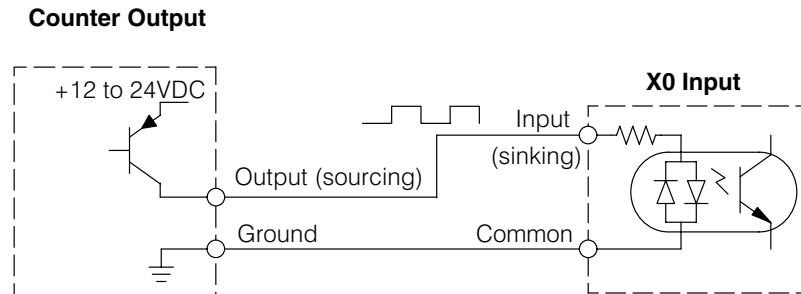


## Interfacing to Counter Outputs

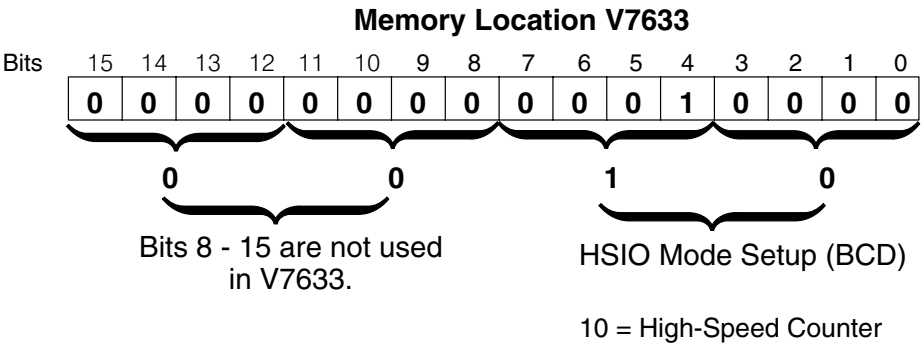
The DL05's DC inputs are flexible in that they detect current flow in either direction, so they can be wired to a counter with either sourcing or sinking outputs. In the following circuit, a counter has open-collector NPN transistor outputs. It sinks current from the PLC input point, which sources current. The power supply can be the FA-24PS or another supply (+12VDC or +24VDC), as long as the input specifications are met.



In the next circuit, an encoder has open-emitter PNP transistor outputs. It sources current to the PLC input point, which sinks the current back to ground. Since the encoder sources current, no additional power supply is required. However, note that the encoder output must be 12 to 24 volts (5V encoder outputs will not work).



**Setup for Mode 10** Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 10 in the lower byte of V7633 to select the High-Speed Counter Mode. The DL05 does not use bits 8 - 15 in V7633.



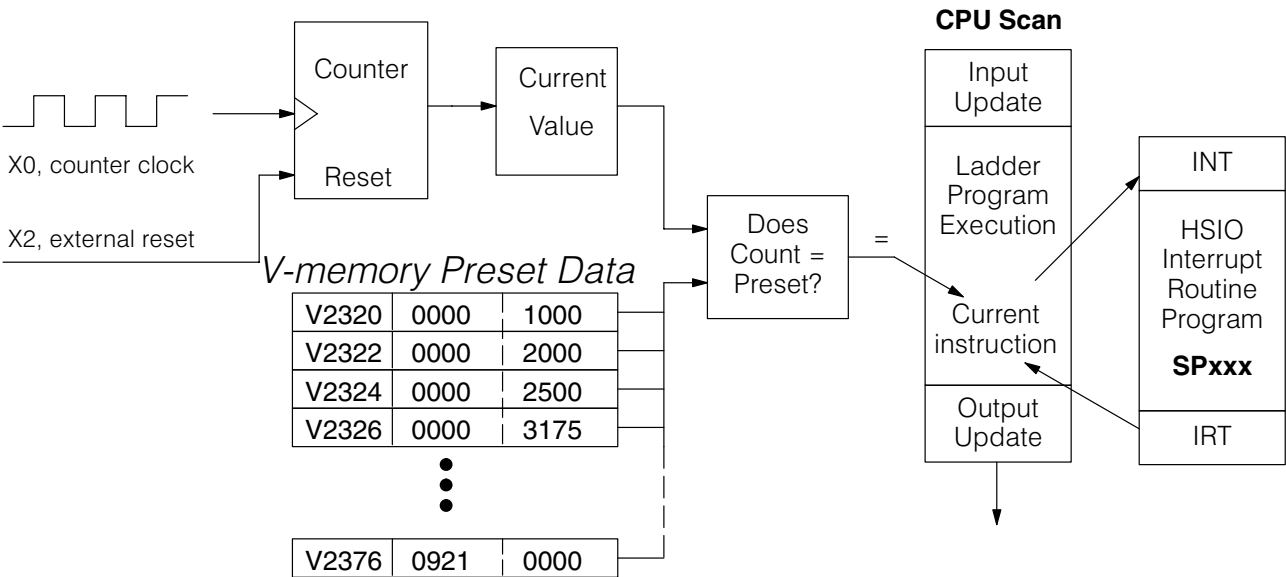
Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- **DirectSOFT**'s memory editor or Data View
- Use the Handheld Programmer D2-HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

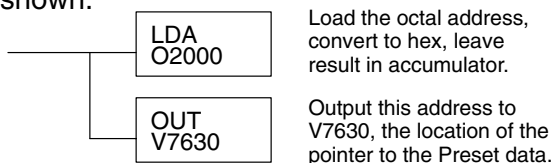
### Presets and Special Relays

The goal of counting is to do a special action when the count reaches a preset value. Refer to the figure below. The counter features 24 presets, which you can program. A preset is a number you derive and store so that the counter will constantly compare the current count with the preset. When the two are equal, a special relay contact is energized and program execution jumps to the interrupt routine. We recommend using the special relay(s) in the interrupt service routine to cause any immediate action you desire. After the interrupt service routine is complete, the CPU returns to the ladder program, resuming program execution from the point of interruption. The compare function is ready for the next preset event.



### Preset Data Starting Location

V7630 is a pointer location which points to the beginning of the Preset Data Table. The default starting location for the Preset Data Table is V2320 (default after initializing scratchpad V-memory). However, you may change this by programming a different value in V7630. Use the LDA and OUT instructions as shown:



Preset Table Pointer

V7630	2000
-------	------

Preset Data

V2000	0000	1000
V2002	0000	2000
V2004	0000	2500
V2006	0000	3175

⋮

V2076	0000	0000
-------	------	------

### Using Fewer than 24 Presets

When using fewer than 24 preset registers, the HSIO looks for “0000 FFFF” (use LDD Kffff) in the next preset location to indicate the last preset has been reached. The example to the right uses four presets. The 0000 FFFF in V2331-V2330 indicates the previous preset was the last.

Preset Data

V2320	0000	1000
V2322	0000	2000
V2324	0000	2500
V2326	0000	3175
V2330	0000	FFFF



**NOTE:** Each successive preset must be greater than the previous preset value. If a preset value is less than a lower-numbered preset value, the CPU cannot compare for that value, since the counter can only count upwards.

### Equal Relay Numbers

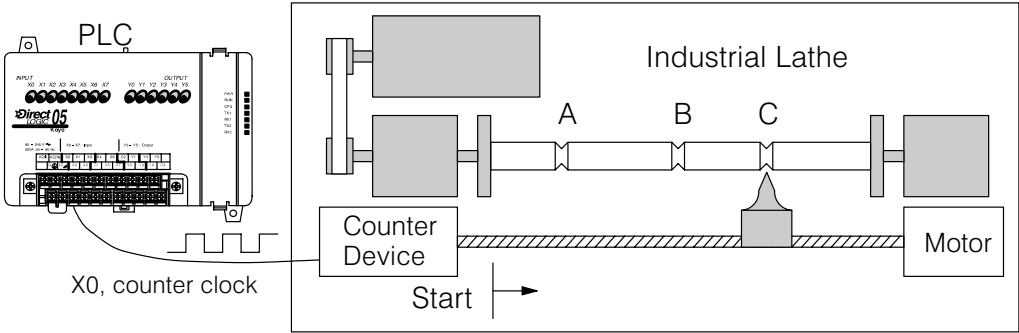
The following table lists all 24 preset register default locations. Each occupies two 16-bit V-memory registers. The corresponding special relay contact number is in the next column. We might also call these “equal” relay contacts, because they are true (closed) when the present high-speed counter value is equal to the preset value. Each contact remains closed until the counter value equals the next preset value.

Preset	Preset V-memory Register	Special Relay Number	Preset	Preset V-memory Register	Special Relay Number
1	V2321 / V2320	SP540	13	V2351 / V2350	SP554
2	V2323 / V2322	SP541	14	V2353 / V2352	SP555
3	V2325 / V2324	SP542	15	V2355 / V2354	SP556
4	V2327 / V2326	SP543	16	V2357 / V2356	SP557
5	V2331 / V2330	SP544	17	V2361 / V2360	SP560
6	V2333 / V2332	SP545	18	V2363 / V2362	SP561
7	V2335 / V2334	SP546	19	V2365 / V2364	SP562
8	V2337 / V2336	SP547	20	V2367 / V2366	SP563
9	V2341 / V2340	SP550	21	V2371 / V2370	SP564
10	V2343 / V2342	SP551	22	V2373 / V2372	SP565
11	V2345 / V2344	SP552	23	V2375 / V2374	SP566
12	V2347 / V2346	SP553	24	V2377 / V2376	SP567

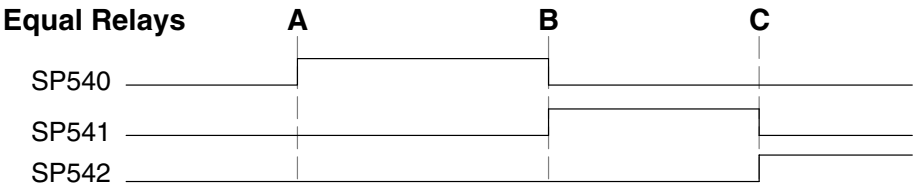
Calculating Your Preset Values

The preset values occupy two data words each. They can range in value from 0000 0000 to 9999 9999, just like the high-speed counter value. All 24 values are *absolute* values, meaning that each one is an offset from the counter zero value.

The preset values must be individually derived for each application. In the industrial lathe diagram below, the PLC monitors the position of the lead screw by counting pulses. At points A, B, and C along the linear travel, the cutter head pushes into the work material and cuts a groove.



The timing diagram below shows the duration of each equal relay contact closure. Each contact remains on until the next one closes. All go off when the counter resets.



**NOTE:** Each successive preset must be two numbers greater than the previous preset value. In the industrial lathe example,  $B > A + 1$  and  $C > B + 1$ .

X Input Configuration

The configurable discrete input options for High-Speed Counter Mode are listed in the table below. Input X0 is dedicated for the counter clock input. Input X1 can be a normal or filtered input. The section on Mode 60 operation at the end of this chapter describes programming the filter time constants. Input X2 can be configured as the counter reset, with or without the interrupt option. The interrupt option allows the reset input (X2) to cause an interrupt like presets do, but there is no SP relay contact closure (instead, X2 will be on during the interrupt routine, for 1 scan). Or finally, X2 may be left simply as a filtered input.

Input	Configuration Register	Function	Hex Code Required
X0	V7634	Counter Clock	0001
X1	V7635	Filtered Input	xx06, xx = filter time 0 - 99 ms (BCD)
X2	V7636	Counter Reset (no interrupt)	0007* (default) 0207*
		Counter Reset (with interrupt)	0107* 0307*
		Filtered Input	xx06, xx = filter time 0 - 99 ms (BCD)

\*With the counter reset, you have the option of a normal reset or a faster reset. However, the fast reset does not recognize changed preset values during program execution. When '0007' or '0107' are set in V7636 and preset values are changed during program execution, the DL05 recognizes the changed preset values at the time of the reset. When '0207' or '0307' are set in V7636 the CPU does not check for changed preset values, so the DL05 has a faster reset time.

### Writing Your Control Program

You may recall that the counter instruction is a standard instruction in the DL05 instruction set. Refer to the figure below. The mnemonic for the counter is **UDC** (up-down counter). The DL05 can have up to 128 counters, labeled CT0 through CT177. The high speed counter in the HSIO circuit is accessed in ladder logic by using UDC CT76. It uses counter registers CT76 and CT77 exclusively when the HSIO mode 10 is active (otherwise, CT76 and CT77 are available for standard counter use). The HSIO counter needs two registers because it is a double-word counter. It has three inputs as shown. The first input (Enable) allows counting when active. The middle input is a dummy and has no function other than it is required by the built-in compiler. The bottom signal is the reset. The Dummy Input must be off while the counter is counting.

#### Standard Counter Function

UP Count	UDC CTxx
DOWN Count	Kxxxxxxxx
Reset Input	

- Counts UP and DOWN
- Preload counter by write to value
- Reset input is internal only

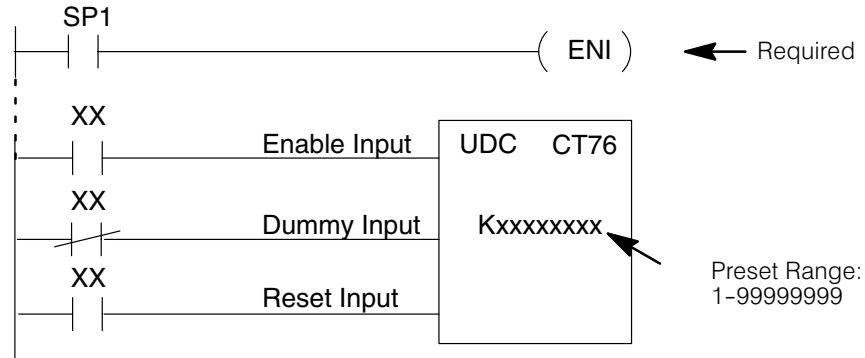
#### HSIO Counter Function

Enable Input	UDC CT76
Dummy Input	Kxxxxxxxx
Reset Input	

- Counts UP only
- Can use Dummy Input to change count
- Reset may be internal or external

The next figure shows how the HSIO counter will appear in a ladder program. Note that the Enable Interrupt (ENI) command must execute before the counter value reaches the first preset value. We do this at powerup by using the first scan relay. When using the counter but not the presets and interrupt, we can omit the ENI.

#### DirectSOFT



When the enable input is energized, the high-speed counter will respond to pulses on X0 and increment the counter at CT76 – CT77. The reset input contact behaves in a logical OR fashion with the physical reset input X2 (when selected). So, the high speed counter can receive a reset from either the contact(s) on the reset rung in the ladder, OR the external reset X2 if you have configured X2 as an external reset.

#### Program Example: Counter Without Preset

The following example is the simplest way to use the high-speed counter, which does not use the presets and special relays in the interrupt routine. The program configures the HSIO circuit for Mode 10 operation, so X0 is automatically the counter clock input. It uses the Compare-double (CMPD) instruction to cause action at certain count values. Note that this allows you to have more than 24 “presets”. Then it configures X2 to be the external reset of the counter.



# Program Example Cont'd

*DirectSOFT*

**First Scan Only**

SP0

**Mode 10**

**Configure Inputs**

LD  
K10

Load constant K10 into the accumulator. This selects Mode 10 as the HSIO mode.

OUT  
V7633

Output the constant K10 to V7633, the location of HSIO Mode select register.

LD  
K1

Load the constant required to configure X0 as the counter clock.

OUT  
V7634

Output the constant K1 to V7634, the location of the setup parameter for X0.

LD  
K1006

Load the constant required to configure input as filtered inputs.

OUT  
V7635

Output the constant K1006 to V7635, the location of setup parameter for X1.

LD  
K7

Load the constant required to configure X2 as an external reset without interrupt.

OUT  
V7636

Output the constant K7 to V7636, the location of the setup parameter for X2.

SP1

SP1

SP1

SP1

UDC CT76  
Kxxxxxxx

CT76 is the HSIO counter. The first rung's SP1 always enables the counter. The dummy input in the middle is always off. The third rung's Reset input is always off, because we will use the external reset.

LDD  
V1076

Load the current count of the HSIO counter in V1076 and V1077 into the accumulator

CMPD  
K309482

Use the Compare-double instruction to compare the double word in the accumulator to the constant K309482

SP62

Y0

(OUT)

The execution of the above CMPD instruction turns on special relay contact SP62 if the current count is greater than the comparison number (K309482).

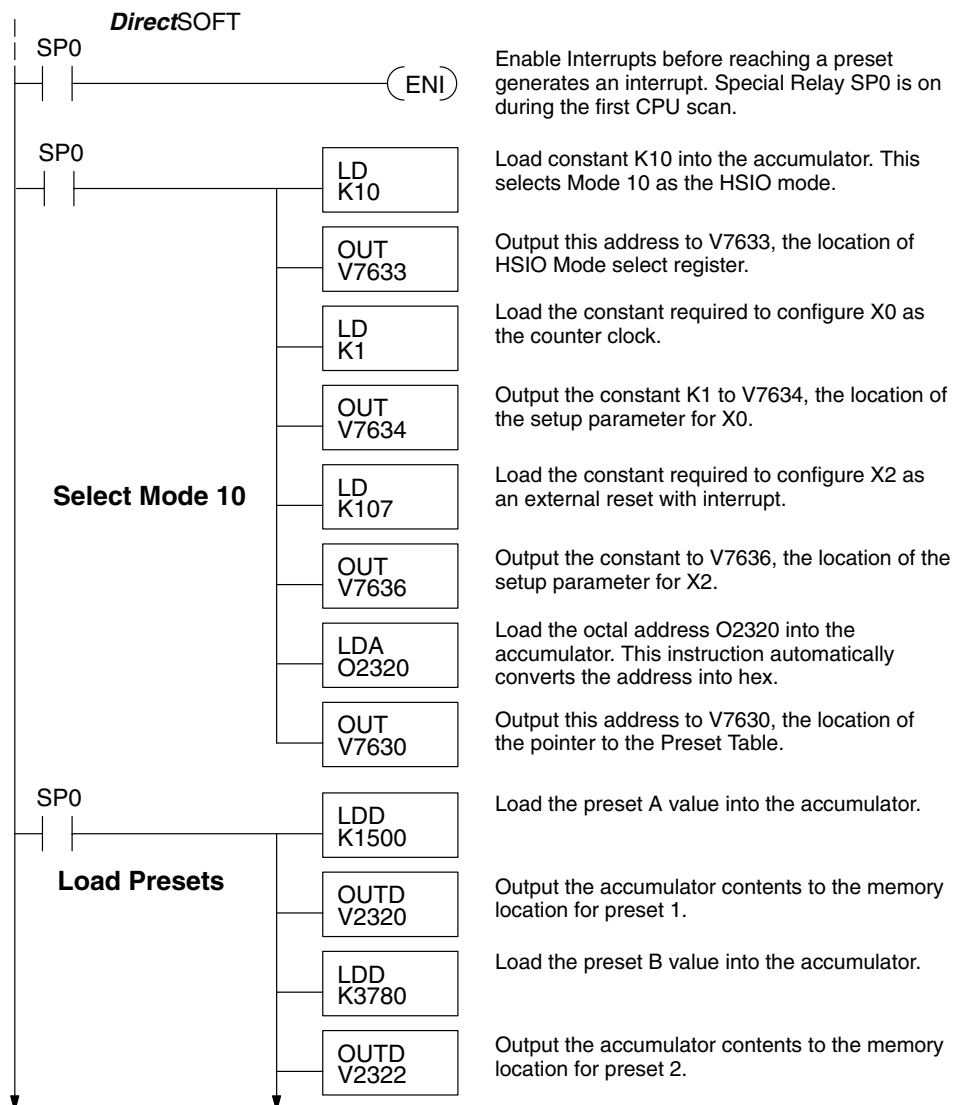
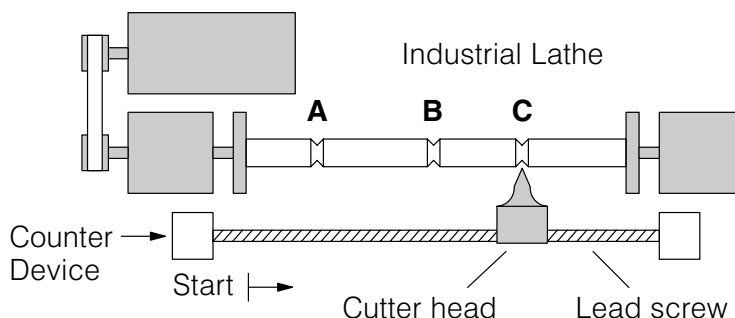
(END)

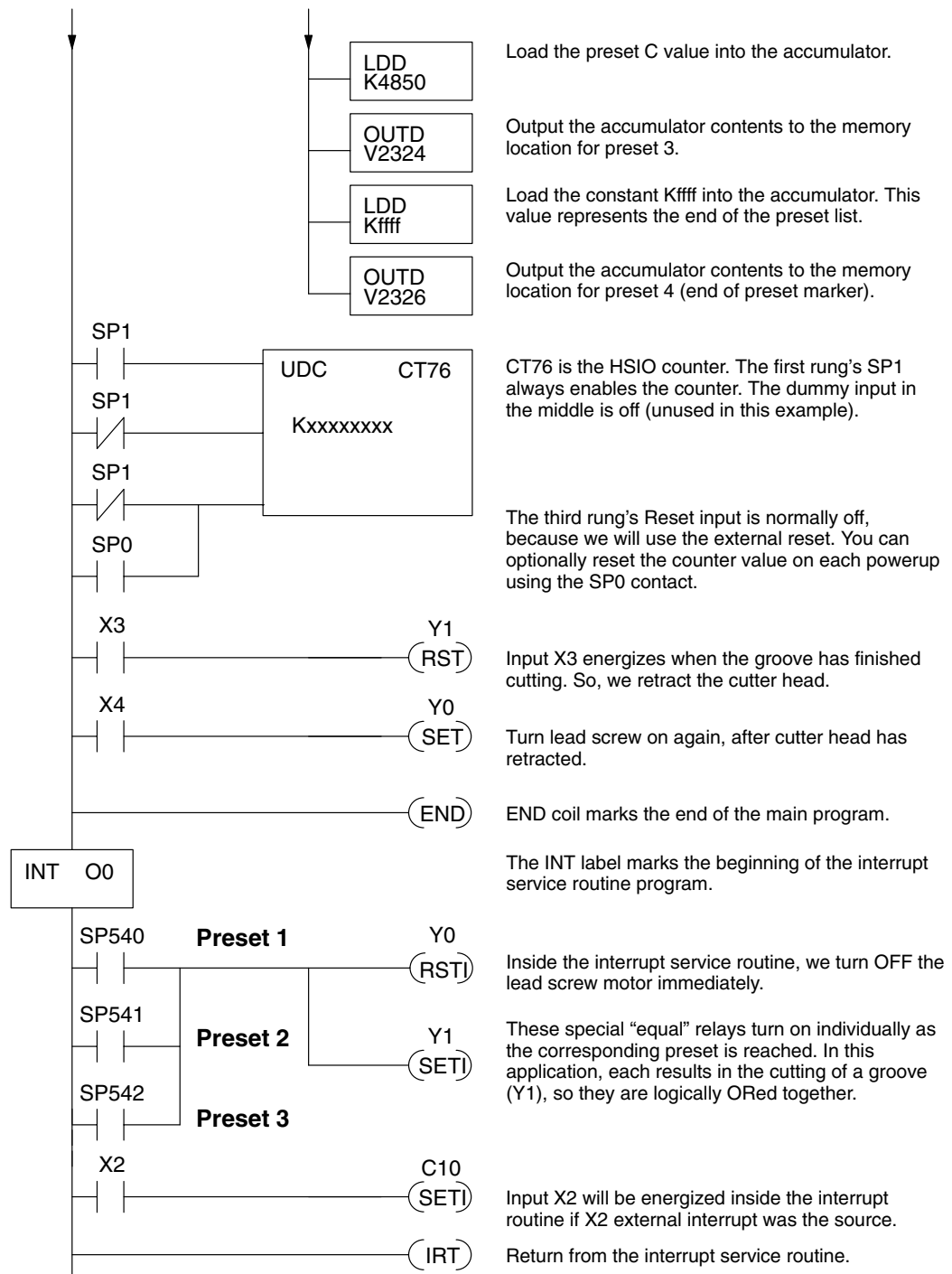
END coil marks the end of the main program.

The compare double instruction above uses the current count of the HSIO counter to turn on Y0. This technique can make more than 24 comparisons, but it is scan-time dependent. However, use the 24 built-in presets with the interrupt routine if your application needs a very fast response time, as shown in the next example.

The following example shows how to program the HSIO circuit to trigger on three preset values. You may recall the industrial lathe example from the beginning of this chapter. This example program shows how to control the lathe cutter head to make three grooves in the work-piece at precise positions. When the lead screw turns, the counter device generates pulses which the DL05 can count. The three preset variables A, B, and C represent the positions (number of pulses) corresponding to each of the three grooves.

I/O	X3 - Cutter head extended
Assignments	X4 - Cutter head retracted
	Y0 - Lead screw motor
	Y1 - Cutter head solenoid

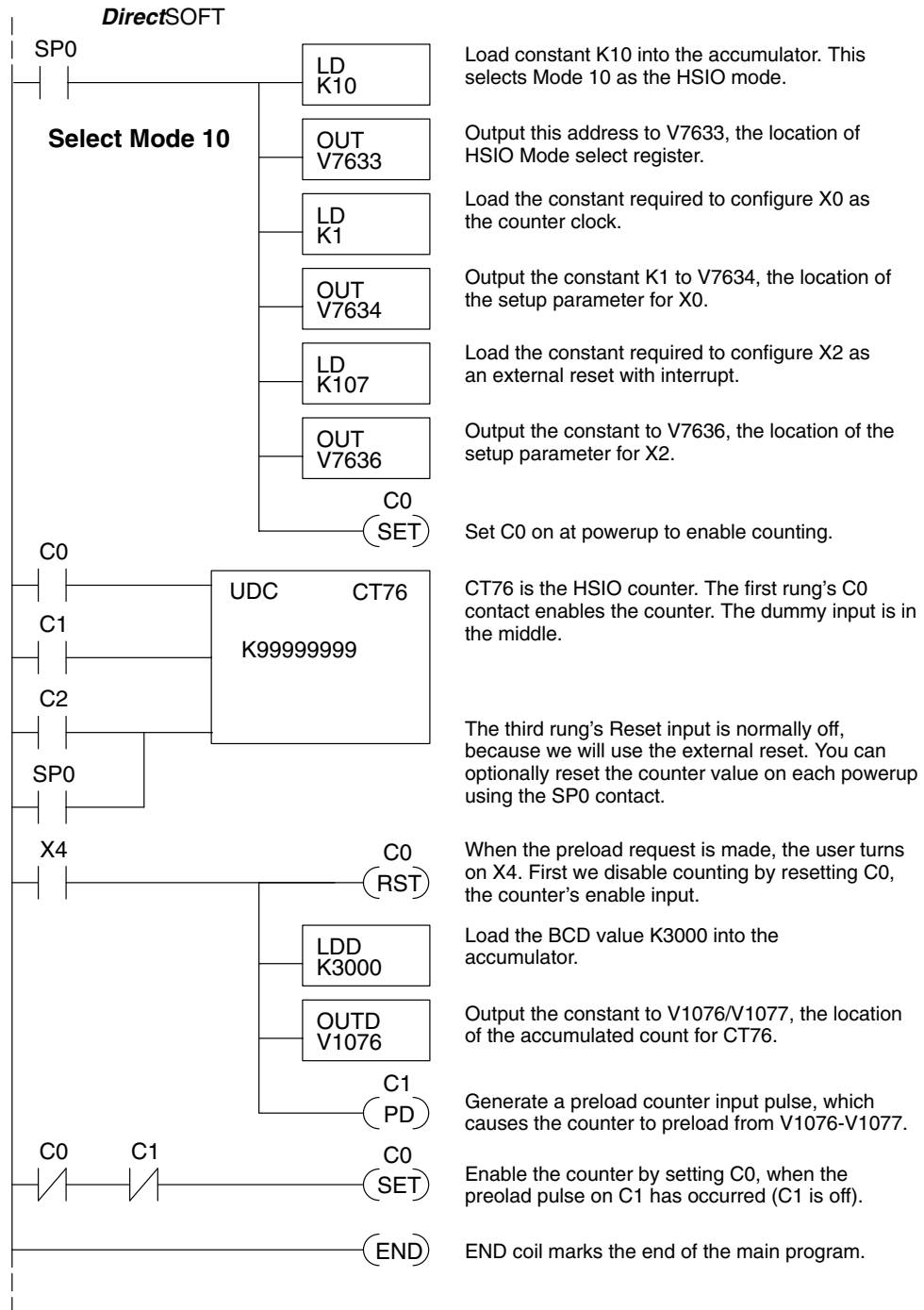




Some applications will require a different type of action at each preset. It is possible for the interrupt routine to distinguish one preset event from another, by turning on a unique output for each equal relay contact SPxxx. We can determine the source of the interrupt by examining the equal relay contacts individually, as well as X2. The X2 contact will be on (inside the interrupt routine only) if the interrupt was caused by the external reset, X2 input.

### Counter With Preload Program Example

The following example shows how you can preload the current count with another value. When the preload command input (X4 in this example) is energized, we disable the counter from counting with C0. Then we write the value K3000 to the count register (V1076-V1077). We preload the current count of the counter with K3000. When the preload command (X4) is turned off, the counter resumes counting any pulses, but now starting from K3000.



**Troubleshooting Guide for Mode 10** If you're having trouble with Mode 10 operation, please study the following symptoms and possible causes. The most common problems are listed below.

**Symptom: The counter does not count.**

Possible causes:

1. **Field sensor and wiring** – Verify that the encoder, proximity switch, or counter actually turns on and illuminates the status LED for X0. The problem could be due to sinking-sourcing wiring problem, etc. Remember to check the signal ground connection. Also verify that the pulse on-time is long enough for the PLC to recognize it.
2. **Configuration** – use the Data View window to check the configuration parameters. V7633 must be set to 10, and V7634 must be set to 1 to enable the HSIO counter mode.
3. **Stuck in reset** – check the input status of the reset input, X2. If X2 is on, the counter will not count because it is being held in reset.
4. **Ladder program** – make sure you are using counter CT76 in your program. The top input is the enable signal for the counter. It must be on before the counter will count. The middle input is the dummy input. The bottom input is the counter reset, and must be off during counting.

**Symptom: The counter counts but the presets do not function.**

Possible causes:

1. **Configuration** – Ensure the preset values are correct. The presets are 32-bit BCD values having a range of 0 to 99999999. Make sure you write all 32 bits to the reserved locations by using the LDD and OUTD instructions. Use only even-numbered addresses, from V2320 to V2376. If using less than 24 presets, be sure to place "0000FFFF" in the location after the last preset used.
2. **Interrupt routine** – Only use Interrupt #0. Make sure the interrupt has been enabled by executing an ENI instruction prior to needing the interrupt. The interrupt routine must be placed after the main program, using the INT label and ending with an interrupt return IRT.
3. **Special relays** – Check the special relay numbers in your program. Use SP540 for Preset 1, SP541 for Preset 2, etc. Remember that only one special equal relay contact is on at a time. When the counter value reaches the next preset, the SP contact which is on now goes off and the next one turns on.

**Symptom: The counter counts up but will not reset.**

Possible causes:

1. Check the LED status indicator for X2 to make sure it is active when you want a reset. Or, if you are using an internal reset, use the status mode of **DirectSOFT** to monitor the reset input to the counter.

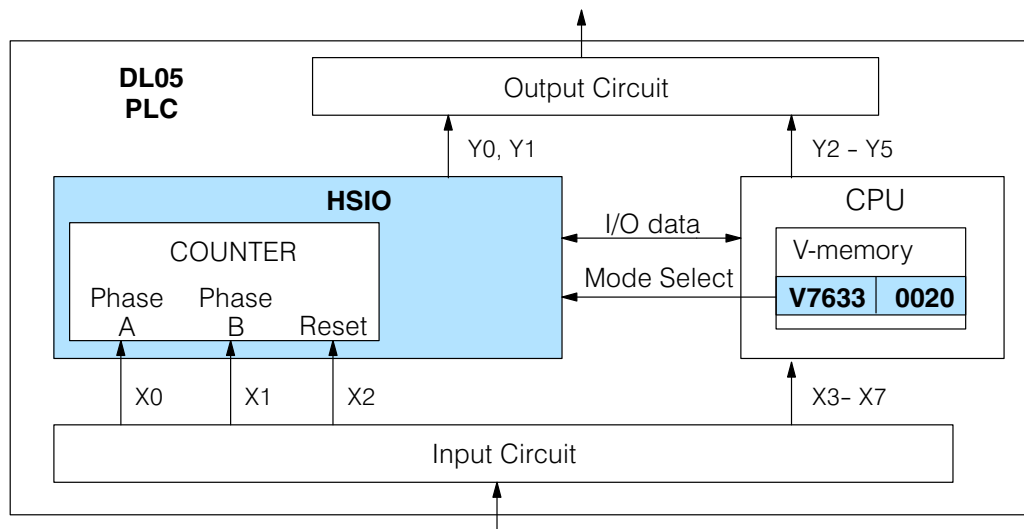
## Mode 20: Quadrature Counter

### Purpose

The counter in the HSIO circuit can count two quadrature signal pulses instead of a single pulse train (mode 10 operation). Quadrature signals are commonly generated from incremental encoders, which may be rotary or linear. The quadrature counter has two ranges from 0 to 99999999 or -8388608 to 8388607. Using CT76 and CT77, the quadrature counter can count at up to a 5 kHz rate. Unlike Mode 10 operation, Mode 20 operation can count UP or DOWN, but *does not feature automated preset values or “interrupt on external reset” capability*. However, you have the standard ladder instruction preset of CT76.

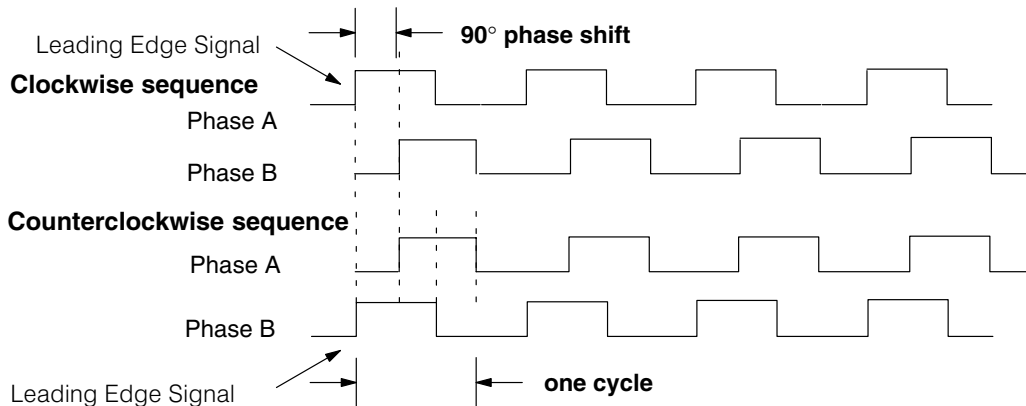
### Functional Block Diagram

The diagram below shows HSIO functionality in Mode 20. When the lower byte of HSIO Mode register V7633 contains a BCD “20”, the quadrature counter in the HSIO circuit is enabled. Input X0 is dedicated to the Phase A quadrature signal, and input X1 receives Phase B signal. X2 is dedicated to reset the counter to zero value when energized.



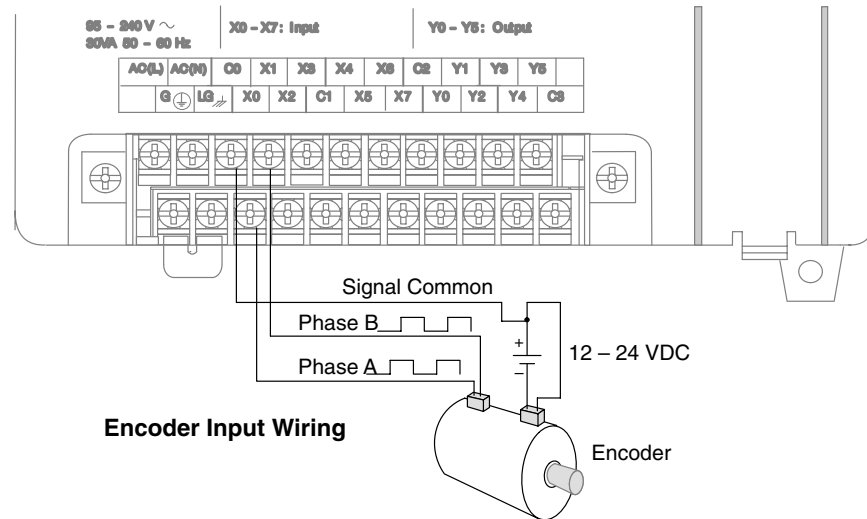
### Quadrature Encoder Signals

Quadrature encoder signals contain position and direction information, while their frequency represents speed of motion. Phase A and B signals shown below are phase-shifted 90 degrees, thus the quadrature name. When the rising edge of Phase A precedes Phase B's leading edge (indicates clockwise motion by convention), the HSIO counter counts UP. If Phase B's rising edge precedes Phase A's rising edge (indicates counter-clockwise motion), the counter counts DOWN.



## Wiring Diagram

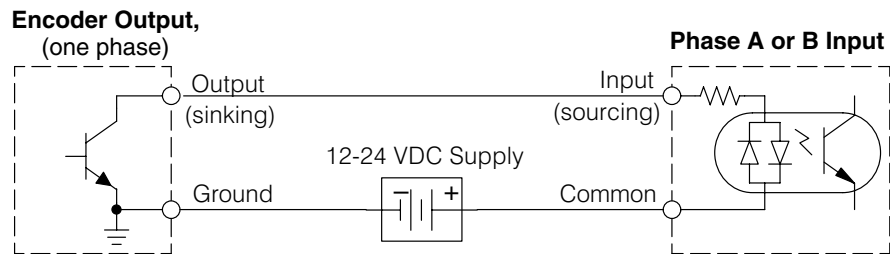
A general wiring diagram for encoders to the DL05 in HSIO Mode 20 is shown below. Encoders with sinking outputs (NPN open collector) are probably the best choice for interfacing. If the encoder sources to the inputs, it must output 12 to 24 VDC. Note that encoders with 5V sourcing outputs will not work with DL05 inputs.



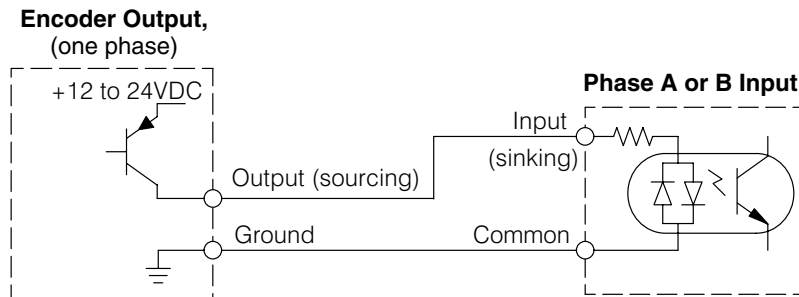
Encoder Input Wiring

## Interfacing to Encoder Outputs

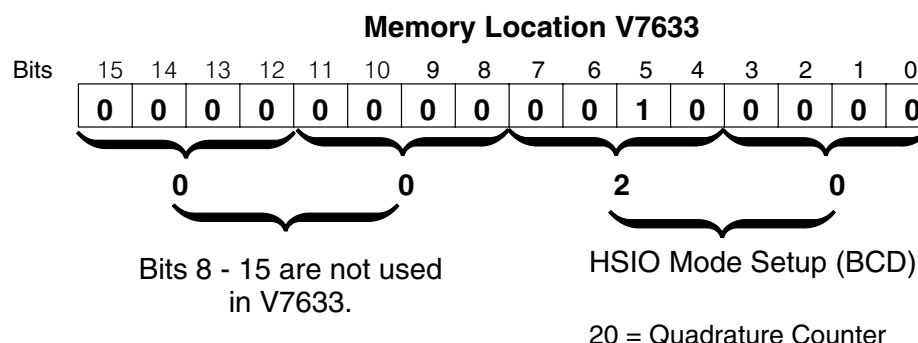
The DL05's DC inputs are flexible in that they detect current flow in either direction, so they can be wired to an encoder with either sourcing or sinking outputs. In the following circuit, an encoder has open-collector NPN transistor outputs. It sinks current from the PLC input point, which sources current. The power supply can be the +24VDC auxiliary supply or another supply (+12VDC or +24VDC), as long as the input specifications are met.



In the next circuit, an encoder has open-emitter PNP transistor outputs. It sources current to the PLC input point, which sinks the current back to ground. Since the encoder sources current, no additional power supply is required. However, note that the encoder output must be 12 to 24 volts (5V encoder outputs will not work).



**Setup for Mode 20** Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 20 in the lower byte of V7633 to select the High-Speed Counter Mode. The DL05 does not use bits 8 - 15 in V7633.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- **DirectSOFT**'s memory editor
- Use the Handheld Programmer D2-HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

### X Input Configuration

The configurable discrete input options for High-Speed Counter Mode are listed in the table below. Input X0 is dedicated for Phase A, and input X1 is for Phase B. Input X2 is the reset input to the quadrature counter, but it does not cause an interrupt. The section on Mode 60 operation at the end of this chapter describes programming the filter time constants.

Input	Configuration Register	Function	Hex Code Required
X0	V7634	Phase A	0002 (default) quadrature, absolute 0 to 99999999
			0012 quadrature, absolute -8388608 to 8388607
X1	V7635	Phase B	0000
X2	V7636	Counter Reset (no interrupt)	0007
		Discrete filtered input	1006

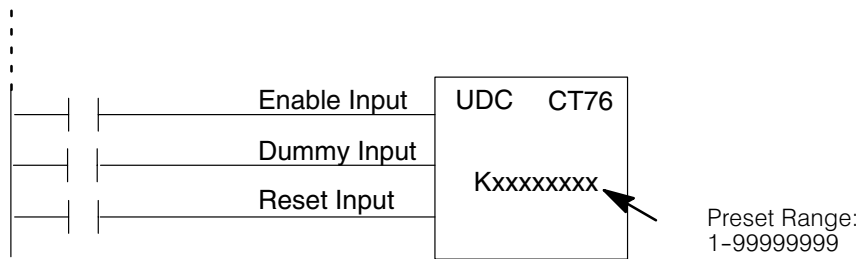


**Writing Your Control Program**

You may recall that the Up-Down counter instruction is standard in the DL05 instruction set. Refer to the figure below. The mnemonic for the counter is **UDC** (up-down counter). The DL05 can have up to 128 counters, labeled CT0 through CT177. The quadrature counter in the HSIO circuit is accessed in ladder logic by using UDC CT76. It uses counter registers CT76 and CT77 exclusively when the HSIO mode 20 is active (otherwise, CT76 and CT77 are available for standard counter use). The HSIO counter needs two registers because it is a double-word counter. It also has three inputs as shown, but they are redefined. The first input is the enable signal, the middle is a preload (write), and the bottom is the reset. The enable input must be on before the counter will count. The enable input must be off during a preload.

Standard Counter Function		HSIO Counter Function	
UP Count	UDC   CTxx  Kxxxxxxxx	Enable Input	UDC   CT76  Kxxxxxxxx
DOWN Count		Dummy Input	
Reset Input		Reset Input	
<ul style="list-style-type: none"><li>● Counts UP and DOWN</li><li>● Preload counter by write to value</li><li>● Reset input is internal only</li></ul>		<ul style="list-style-type: none"><li>● Counts UP and DOWN (from X0, X1)</li><li>● Can use Dummy Input to change count</li><li>● Reset may be internal or external</li></ul>	

The next figure shows the how the HSIO quadrature counter will appear in a ladder program.



When the enable input is energized, the counter will respond to quadrature pulses on X0 and X1, incrementing or decrementing the counter at CT76 – CT77. The reset input contact behaves in a logical OR fashion with the physical reset input X2. This means the quadrature counter can receive a reset from either the contact(s) on the reset rung in the ladder, OR the external reset X2.

**Quadrature Counter w/Preload Program Example**

Since presets are not available in quadrature counting, this mode is best suited for simple counting and measuring. The example program on the following page shows how to configure the quadrature counter. The program configures the HSIO circuit for Mode 20 operation, so X0 is Phase A and X1 is Phase B clock inputs.

# Program Example Cont'd

## DirectSOFT

SP0

### Select Mode 20

LD  
K20

Load constant K20 into the accumulator. This selects Mode 20 as the HSIO mode.

OUT  
V7633

Output this address to V7633, the location of the HSIO Mode select register.

LD  
K2

Load the constant required to configure X0 as Phase A input.

OUT  
V7634

Output the constant to V7634, the location of the setup register for X0.

LD  
K0

Load the constant required to configure X1 as Phase B input.

OUT  
V7635

Output the constant to V7635, the location of the setup register for X1.

LD  
K7

Load the constant required to configure X2 as an external reset.

OUT  
V7636

Output the constant to V7636, the location of the setup register for X2.

C0  
(SET)

Set C0 on at powerup to enable counting.

C0

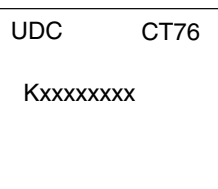
C1

C2

SP0

SP1

### Select Mode 20



CT76 is the HSIO quadrature counter. The first rung's SP1 always enables the counter. The dummy input is used by the built-in compiler.

The third rung's Reset input is normally off, because we will use the external reset. You can optionally reset the counter value on each powerup using the SP0 contact.

LDD  
V1076

Load the current value of the counter into the accumulator on each scan.

CMPD  
K44292

Compare the value in the accumulator with the constant K44292. If they are equal, the SP61 contact will be turned on.

SP61

SP62

SP61

SP60

Y0  
(SET) \*

Set Y0 to ON when the counter reaches or exceeds our comparison value while COUNTING UP.

Y1  
(SET) \*

Set Y1 to ON when the counter reaches or goes below our comparison value while COUNTING DOWN.

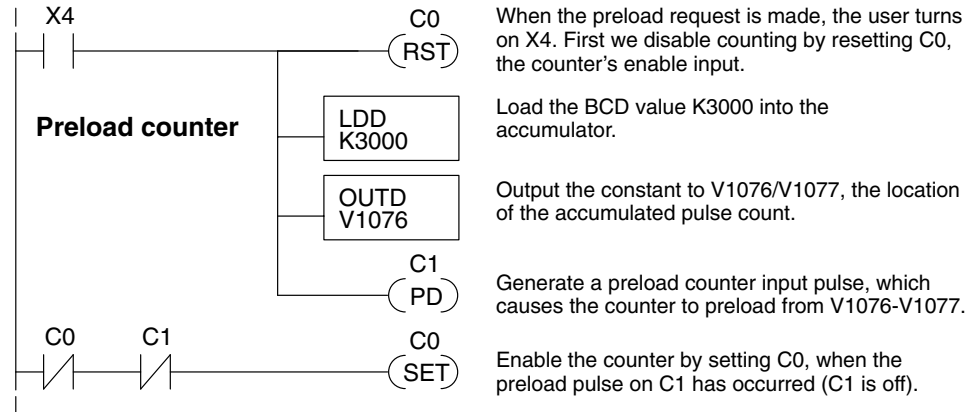
(END)

END coil marks the end of the main program..

\* Note: You can reset Y0 later in the program by using the RST instruction.

To preload the counter, just add the following example rungs to the program above.

### Counter Preload Program Example



**Troubleshooting Guide for Mode 20** If you're having trouble with Mode 20 operation, please study the following symptoms and possible causes. The most common problems are listed below.

#### Symptom: The counter does not count.

Possible causes:

1. **Field sensor and wiring** – Verify that the encoder or other field device inputs actually turn on and illuminates the status LEDs for X0 and X1. A standard incremental encoder will visibly, alternately turn on the LEDs for X0 and X1 when rotating slowly (1 RPM). Or, the problem could be due to a sinking-sourcing wiring problem, etc. Remember to check the signal ground connection. Also verify that the pulse on-time, duty cycle, voltage level, and frequency are within the input specifications.
2. **Configuration** – make sure all of the configuration parameters are correct. V7633 must be set to 20, and V7634 must be set to "0002" to enable the Phase A input, and V7635 must be set to "0000" to enable the Phase B input.
3. **Stuck in reset** – check the input status of the reset input, X2. If X2 is on, the counter will not count because it is being held in reset.
4. **Ladder program** – make sure you are using counter CT76 in your program. The top input is the enable signal for the counter. It must be on before the counter will count. The middle input is the dummy input and must be off for the counter to count. The bottom input is the counter reset, and must be off during counting.

#### Symptom: The counter counts in the wrong direction (up instead of down, and visa-versa).

Possible causes:

1. **Channel A and B assignment** – It's possible that Channel A and B assignments of the encoder wires is backwards from the desired rotation/counting orientation. Just swap the X0 and X1 inputs, and the counting direction will be reversed.

#### Symptom: The counter counts up and down but will not reset.

Possible causes:

1. Check the LED status indicator for X2 to make sure it is active when you want a reset. Also verify the configuration register V7636 for X2 is set to 7. Or, if you are using an internal reset, use the status mode of **DirectSOFT** to monitor the reset input to the counter.

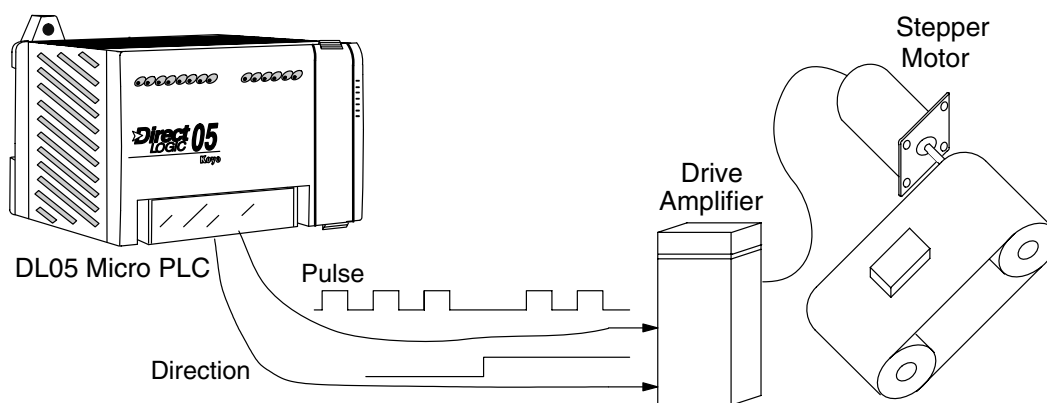
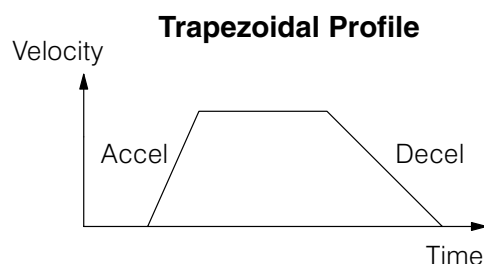
## Mode 30: Pulse Output

### Purpose

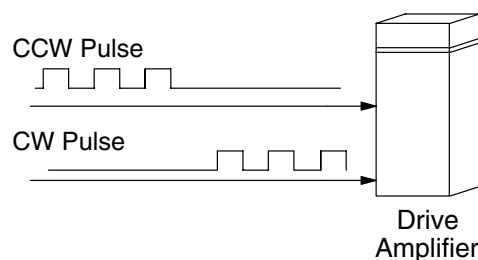
The HSIO circuit in Mode 30 generates output pulse trains suitable for open-loop control of a single-axis motion positioning system. It generates pulse (stepper increment) and direction signals which you can connect to motor drive systems and perform various types of motion control. Using Mode 30 Pulse Output, you can select from three profile types detailed later in this chapter:

- **Trapezoidal** – Accel Slope to Target Velocity to Decel Slope
- **Registration** – Velocity to Position Control on Interrupt (also used for home search moves)
- **Velocity Control** – Speed and Direction only

The HSIO circuit becomes a high-speed pulse generator (up to 7 kHz) in Mode 30. By programming acceleration and deceleration values, position and velocity target values, the HSIO function automatically calculates the entire motion profile. The figure below shows the DL05 generating pulse and direction signals to the drive amplifier of a stepper positioning system. The pulses accomplish the profile independently and without interruption to ladder program execution in the CPU.



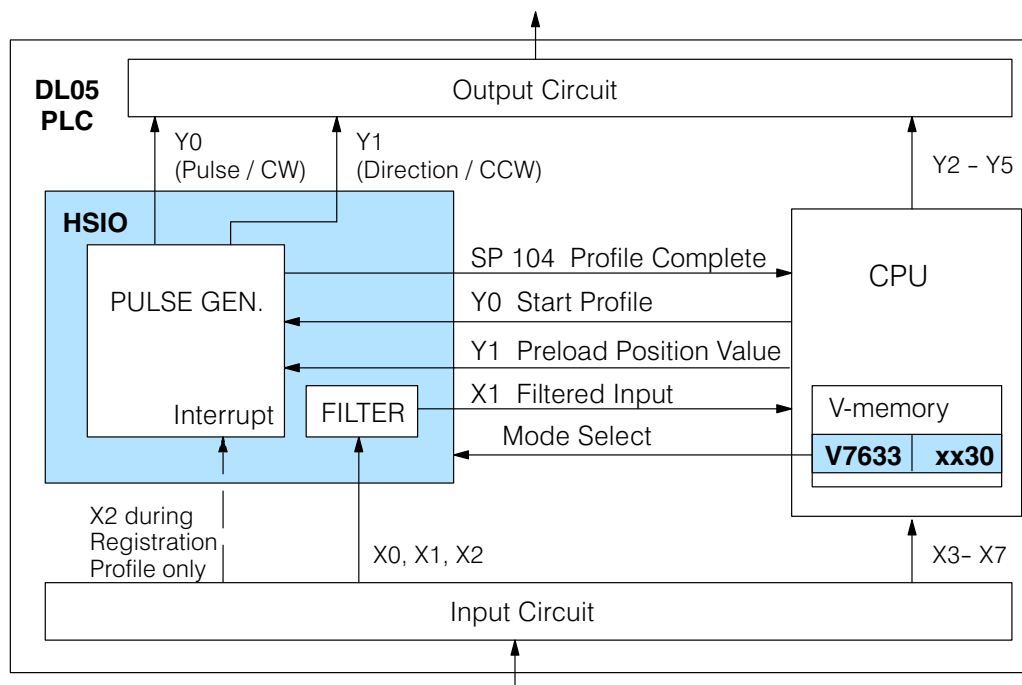
In the figure above, the DL05 generates pulse and direction signals. Each pulse represents the smallest increment of motion to the positioning system (such as one step or micro-step to a stepper system). Alternatively, the HSIO Pulse Output Mode may be configured to deliver counter clock-wise (CCW) and clock-wise (CW) pulse signals as shown to the right.



**NOTE:** The pulse output is designed for open loop stepper motor systems. This, plus its minimum velocity of 40 pps make it unsuitable for servo motor control.

### Functional Block Diagram

The diagram below shows HSIO functionality in Mode 30. When the lower byte of HSIO Mode register V7633 contains a BCD “30”, the pulse output capability in the HSIO circuit is enabled. The pulse outputs use Y0 and Y1 terminals on the output connector. Remember that the outputs can only be DC type to operate.



**IMPORTANT NOTE:** In Pulse Output Mode, Y0 and Y1 references are redefined or are used differently in two ways. *Physical* references refer to terminal screws, while *logical* references refer to I/O references in the ladder program. Please read the items below to understand this very crucial point.

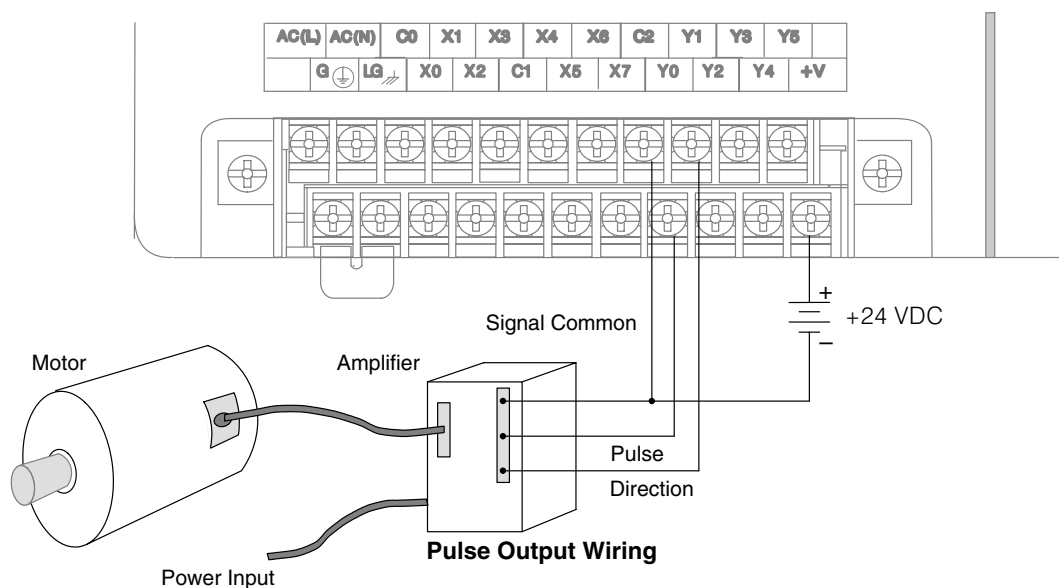
Notice the I/O point assignment and usage in the above diagram:

- X0 and X1 can only be filtered inputs in Pulse Output Mode, and they are available as an input contacts to the ladder program.
- X2 behaves as an external interrupt to the pulse generator for registration profiles. In other profile modes, it can be used as a filtered input just like X1 (registration mode configuration shown above).
- References “Y0” and “Y1” are used in two different ways. At the discrete output connector, Y0 and Y1 terminals deliver the pulses to the motion system. The ladder program uses logical references Y0 and Y1 to initiate “Start Profile” and “Load Position Value” HSIO functions in Mode 30.

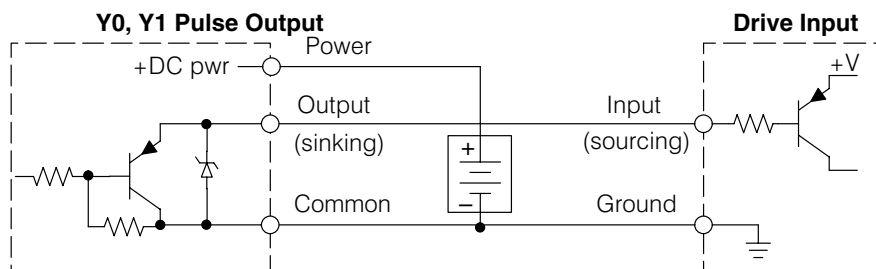
Hopefully, the above discussion will explain why some I/O reference names have dual meanings in Pulse Output Mode. **Please read the remainder of this section with care**, to avoid confusion about which actual I/O function is being discussed.

**Wiring Diagram**

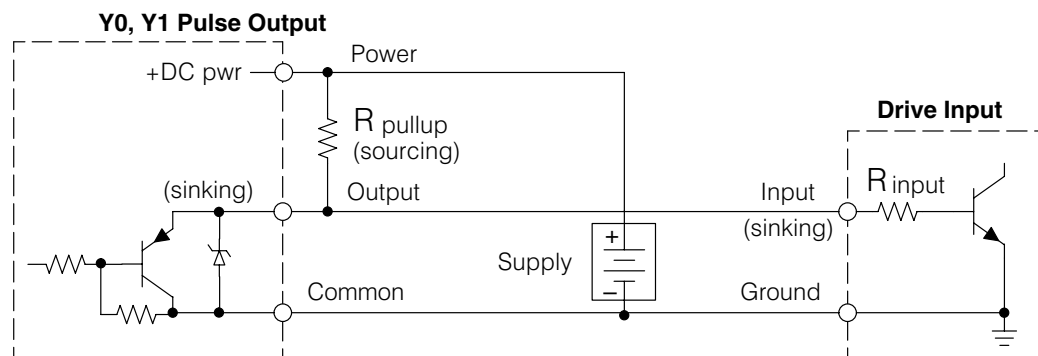
The generalized wiring diagram below shows pulse outputs Y0 and Y1 connected to the drive amplifier inputs of a motion control system.

**Interfacing to Drive Inputs**

The pulse signals from Y0 and Y1 outputs will typically go to drive input circuits as shown above. Remember that the DL05's DC outputs are sinking-only. It will be helpful to locate equivalent circuit schematics of the drive amplifier. The following diagram shows how to interface to a sourcing drive input circuit.



The following circuit shows how to interface to a sinking drive input using a pullup resistor. Please refer to Chapter 2 to learn how to calculate and install  $R_{pullup}$ .



**Motion Profile Specifications**

The motion control profiles generated in Pulse Output Mode have the following specifications:

Parameter	Specification
Profiles	Trapezoidal – Accel Slope / Target Velocity / Decel Slope
	Registration – Velocity to Position Control on Interrupt
	Velocity Control – Speed and Direction only
Position Range	–88388608 to 88388607
Positioning	Absolute / relative command
Velocity Range	40 Hz to 7 kHz
V-memory registers	V2320 to V2325 (Profile Parameter Table)
Current Position	CT76 and CT77 (V1076 and V1077)

**Physical I/O Configuration**

The configurable discrete I/O options for Pulse Output Mode are listed in the table below. The CPU uses SP 104 contact to sense “profile complete”. V7637 is used to select pulse/direction or CCW/CW modes for the pulse outputs. Input X2 is dedicated as the external interrupt for use in registration mode.

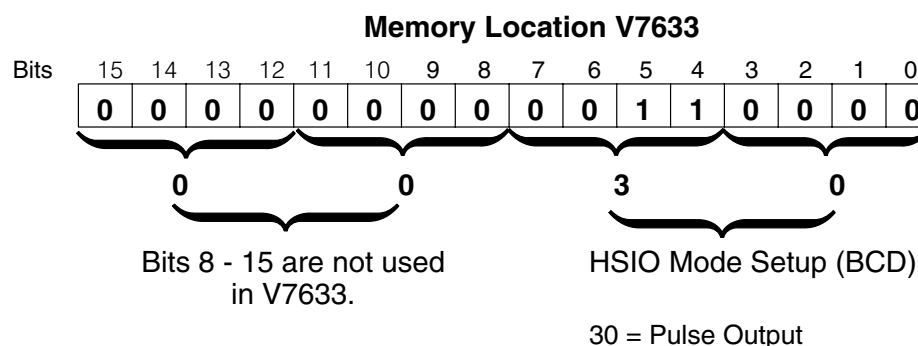
Physical Input	Configuration Register	Function	Hex Code Required
–	V7637	Y0 = Pulse Y1 = Direction	0103
		Y0 = CW Pulse Y1 = CCW Pulse	0003
X0	V7634	Discrete filtered input	xx06, xx = filter time 0 - 99 ms (BCD)
X1	V7635	Discrete filtered input	
X2	V7636	Discrete filtered input	

**Logical I/O Functions**

The following logical I/O references define functions that allow the HSIO to communicate with the ladder program.

Logical I/O	Function
SP 104	Profile Complete – the HSIO turns on SP104 to the CPU when the profile completes. Goes back off when Start Profile (Y0) turns on.
Y0	Start Profile – the ladder program turns on Y0 to start motion. If turned off before the move completes, motion stops. Turning it on again will start another profile, unless the current position equals the target position.
Y1	Preload Position Value – if motion is stopped and Start Profile is off, you can load a new value in CT76/CT77, and turn on Y1. At that transition, the value in CT76/CT77 becomes the current position.

**Setup for Mode 30** Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 30 in the lower byte of V7633 to select the High-Speed Counter Mode. The DL05 does not use bits 8 - 15 in V7633.



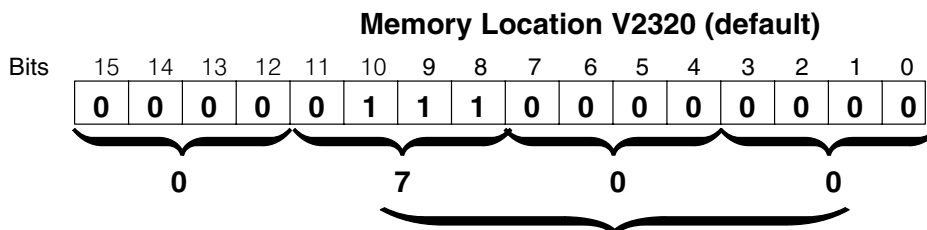
Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- **DirectSOFT**'s memory editor
- Use the Handheld Programmer D2-HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

### Profile / Velocity Select Register

The first location in the Profile Parameter Table stores two key pieces of information. The upper four bits (12–15) select the type of profile required. The lower 12 bits (0–11) select the Target Velocity.



Profile Select (BCD)

- 0 = Trapezoidal Profile, Absolute Position
- 8 = Trapezoidal Profile, Relative Position
- 9 = Registration Profile, Relative Position
- 2 = Velocity Profile

Target Velocity Value

Range = 4 to 700, representing  
40 Hz to 7 kHz pulse rate

The ladder program must program this location before initiating any of the three profiles. The LD and OUT instruction will write all 16 bits, so be sure to fully specify the full four-digit BCD value for the Position / Velocity Select Register each time.

The absolute and relative selection determines how the HSIO circuit will interpret your specified target position. Absolute position targets are referenced to zero. Relative position targets are referenced to the current position (previous target position). You may choose whichever reference method that is most convenient for your application.



**Profile  
Parameter Table**

V7630 is a pointer location which points to the beginning of the Profile Parameter Table. The default starting location for the profile parameter table is V2320. However, you may change this by programming a different value in V7630. Remember to use the LDA (load address) instruction, converting octal into hex.

The HSIO uses the next V-memory register past the bottom of the profile parameter table to indicate profile errors. See the error table at the end of this section for error code definitions.

Profile Table Pointer

V7630	2320
-------	------

Profile Parameter Table

V2320	xxxx	
V2321	xxxx	xxxx
V2323	xxxx	
V2324	xxxx	
V2325	xxxx	

Pulse Output Error Code

V2326	00xx
-------	------

**Trapezoidal Profile**

V-Memory	Function	Range	Units
V2320, bits 12–15	Trapezoidal Profile	0=absolute, 8=relative	–
V2320, bits 0–11	Target Velocity Value	4 to 700	x 10 pps
V2321/ 2322	Target Position Value	–8388608 to 8388607	Pulses
V2323	Starting Velocity	4 to 100	x 10 pps
V2324	Acceleration Time	1 to 100	x 100 mS
V2325	Deceleration Time	1 to 100	x 100 mS
V2326	Error Code	(see end of section)	–

**Registration Profile**

V-Memory	Function	Range	Units
V2320, bits 12–15	Registration Profile	9=relative	–
V2320, bits 0–11	Target Velocity Value	4 to 700	x 10 pps
V2321/ 2322	Target Position Value	–8388608 to 8388607	Pulses
V2323	Starting Velocity	4 to 100	x 10 pps
V2324	Acceleration Time	1 to 100	x 100 mS
V2325	Deceleration Time	1 to 100	x 100 mS
V2326	Error Code	(see end of section)	–

**Velocity Profile**

V-Memory	Function	Range	Units
V2320	Velocity Profile	2000 only	–
V2321/ 2322	Direction Select	80000000=CCW, 0=CW	Pulses
V2323	Velocity	4 to 700	x 10 pps
V2326	Error Code	(see end of section)	–

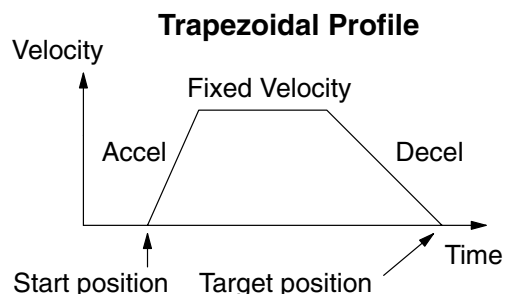
### Choosing the Profile Type

Pulse Output Mode generates three types of motion profiles. Most applications use one type for most moves. However, each move can be different if required.

- Trapezoidal – Accel Slope to Target Velocity to Decel Slope
- Registration – Velocity to Position Control on Interrupt
- Velocity Control – Speed and Direction only

### Trapezoidal Profile Defined

The trapezoidal profile is the most common positioning profile. It moves the load to a pre-defined target position by creating a move profile. The acceleration slope is applied at the starting position. The deceleration slope is applied backwards from the target position. The remainder of the move in the middle is spent traveling at a defined velocity.



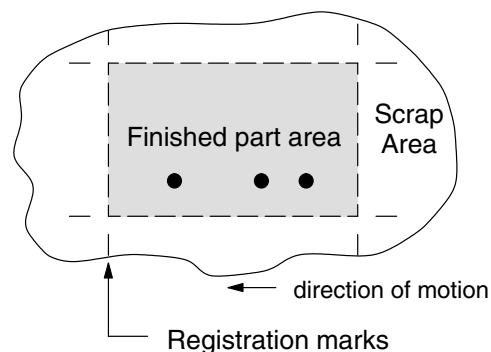
Trapezoidal profiles are best for simple point-to-point moves, when the distance between the starting and ending positions of the move is known in advance.

### Registration and Home Search Profiles Defined

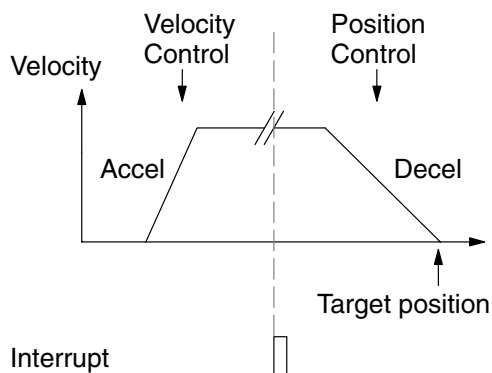
Registration profiles solve a class of motion control problems. In some applications, product material in work moves past a work tool such as a drill station. Shown to the right, registration marks on the scrap area of the work-piece allow a machine tool to register its position relative to the rectangle, to drill properly.

Home search moves allow open-loop motion systems to re-calibrate (preload) the current position value at powerup.

Registration profiles are a combination of velocity and position control modes. The move begins by accelerating to a programmed velocity. The velocity is sustained and the move is of indefinite duration. When an external interrupt signal occurs (due to registration sensing), the profile switches from velocity to position control. The move ends by continuing motion a pre-defined distance past the interrupt point (such as a drill hole location). The deceleration ramp is applied in advance of the target position.

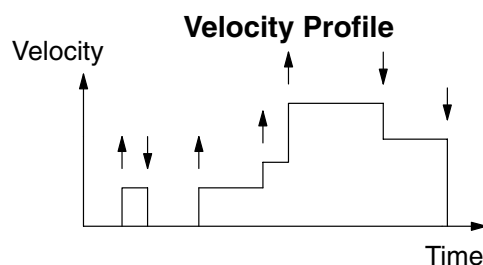


### Registration Profile



### Velocity Profile Defined

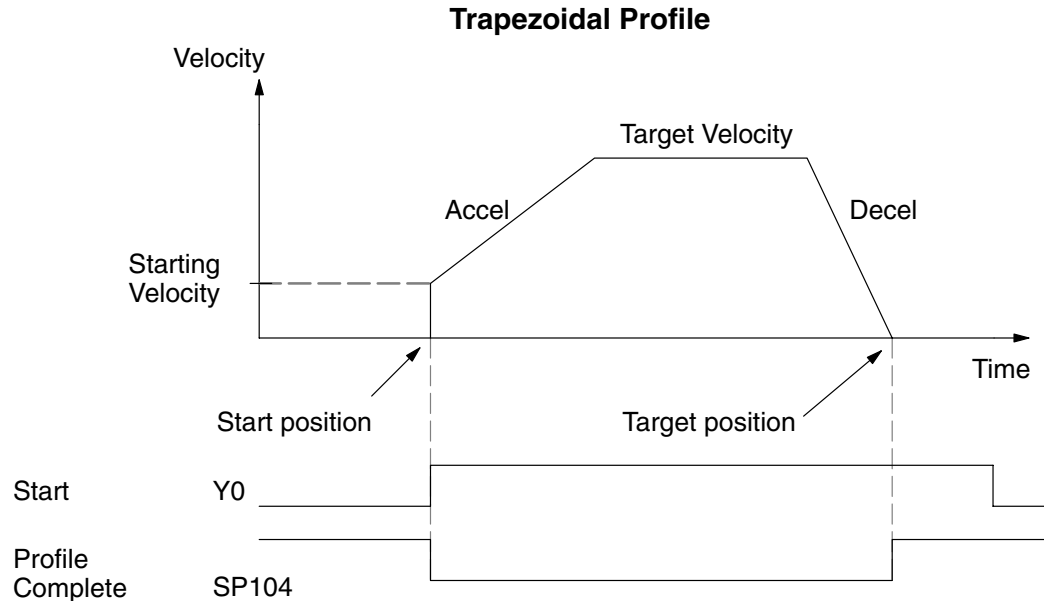
The velocity profile controls only the direction and speed of motion. There is no target position specified, so the move can be of indefinite length. Only the first velocity value needs to be defined. The remaining velocity values can be created while motion is in progress. Arrows in the profile shown indicate velocity changes.



## Trapezoidal Profile Operation

### Trapezoidal Profile Applications

The trapezoidal profile is best suited for simple point-to-point moves, when the target position is known in advance. Starting velocities must be within the range of 40 pps to 1k pps. The remainder of the profile parameters are in the profile parameter table.



The time line of signal traces below the profile indicates the order of events.

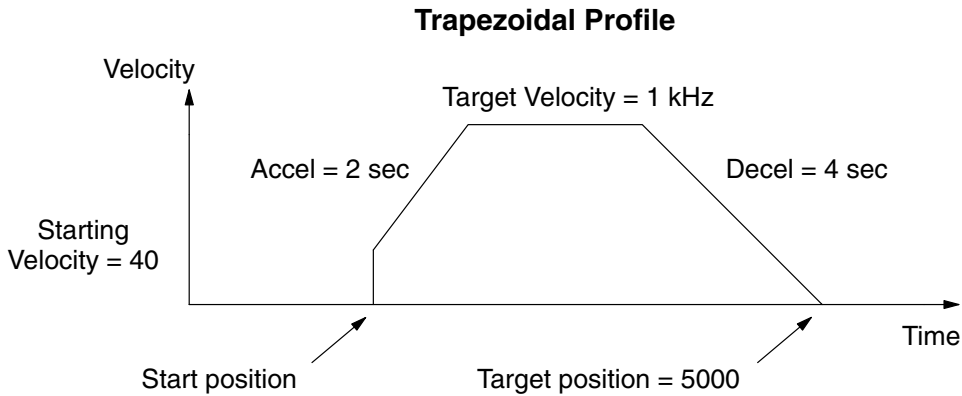
The HSIO uses logical output Y0 as the Start input to the HSIO, which starts the profile. Immediately the HSIO turns off the Profile Complete signal (SP104), so the ladder program can monitor the progress of the move. Typically a ladder program will monitor this bit so it knows when to initiate the next profile move.

If you are familiar with motion control, you'll notice that we do not have to specify the direction of the move. The HSIO function examines the target position relative to the current position, and automatically outputs the correct direction information to the motor drive.

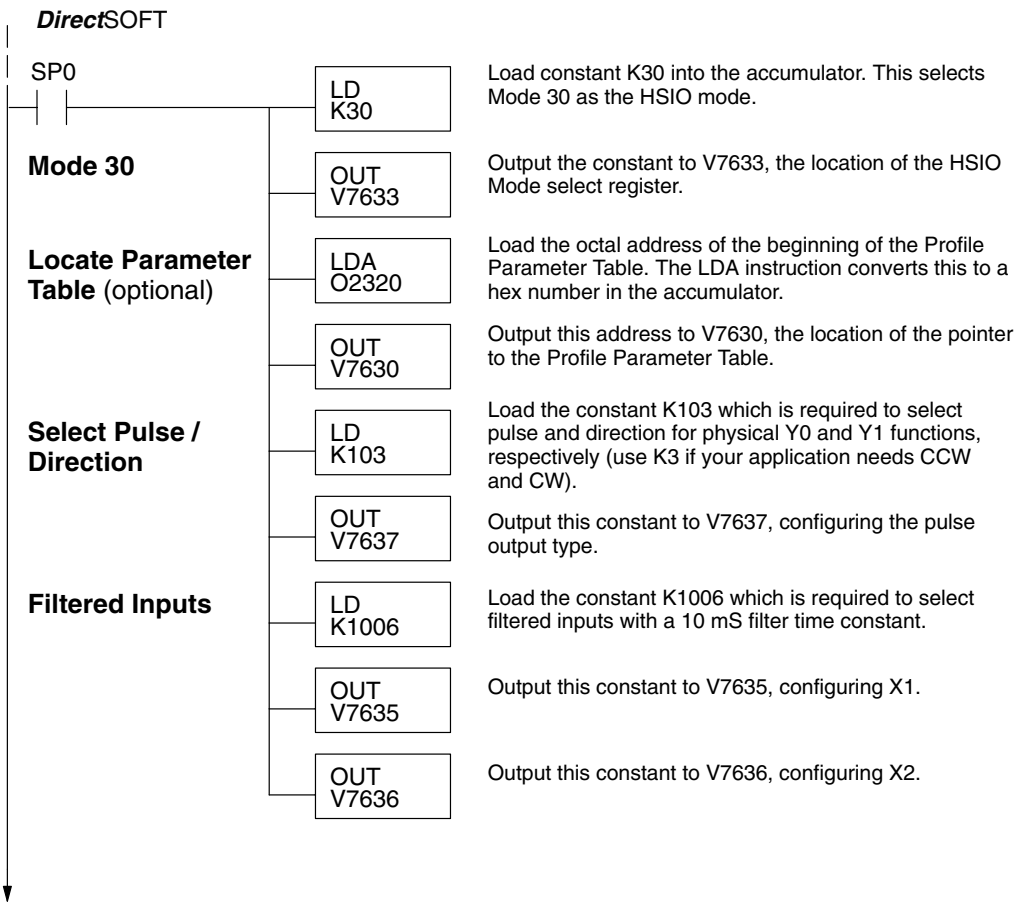
Notice that the motion accelerates immediately to the starting velocity. This segment is useful in stepper systems so we can jump past low speed areas when low-torque problems or a resonant point in the motor might cause a stall. (When a stepper motor stalls, we have lost the position of the load in open-loop positioning systems). However, it is preferable not to make the starting velocity too large, because the stepper motor will also "slip" some pulses due to the inertia of the system.

When you need to change the current position value, use logical Y1 output coil to load a new value into the HSIO counter. If the ladder program loads a new value in CT76/CT77 (V1076/V1077), then energizing Y1 will copy that value into the HSIO circuit counter. This must occur before the profile begins, because the HSIO ignores Y1 during motion.

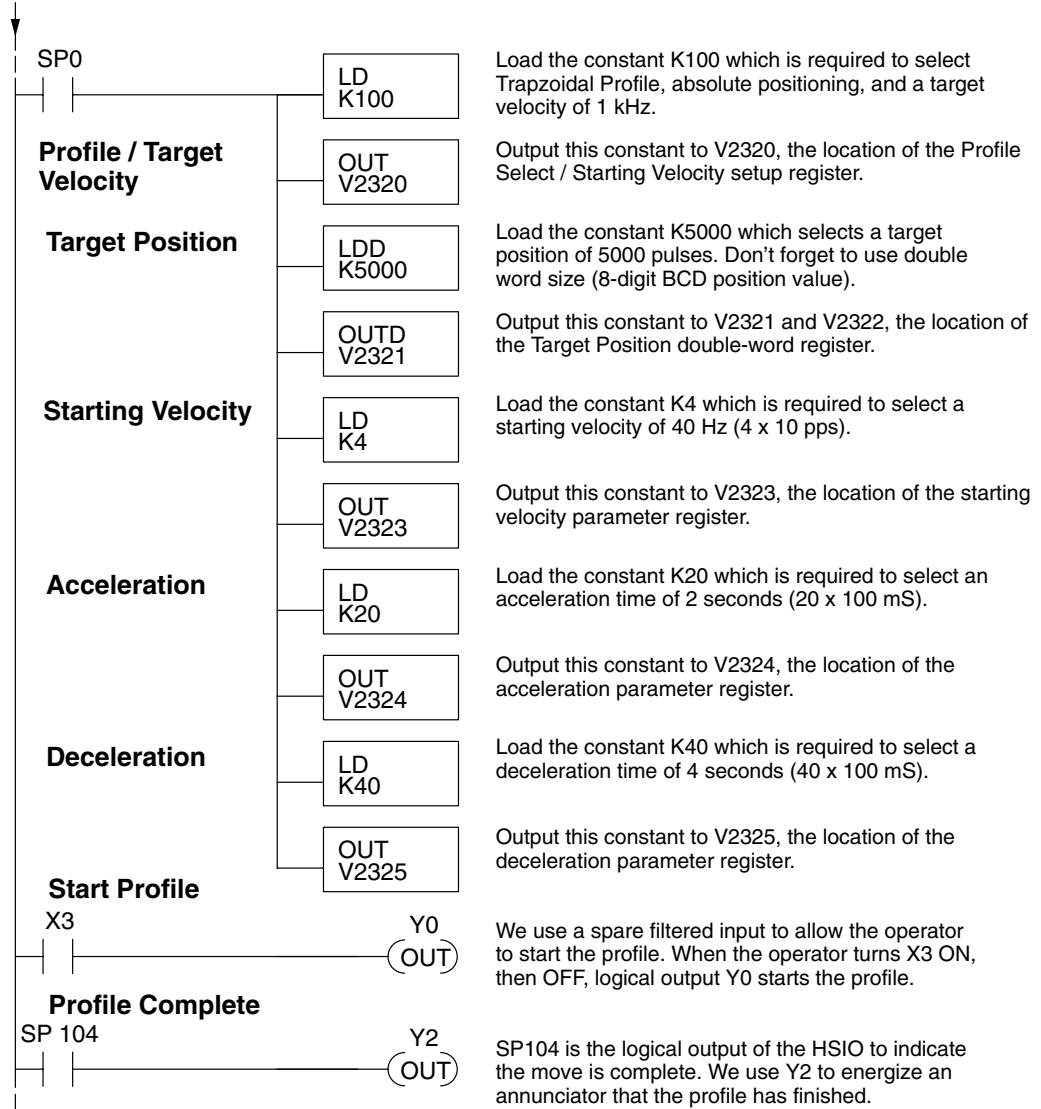
**Trapezoidal Profile Program Example** The trapezoidal profile we want to perform is drawn and labeled in the following figure. It consists of a non-zero starting velocity, and moderate target velocity.



The following program will realize the profile drawn above, when executed. The beginning of the program contains all the necessary setup parameters for Pulse Output Mode 30. We only have to do this once in the program, so we use first-scan contact SP0 to trigger the setup.

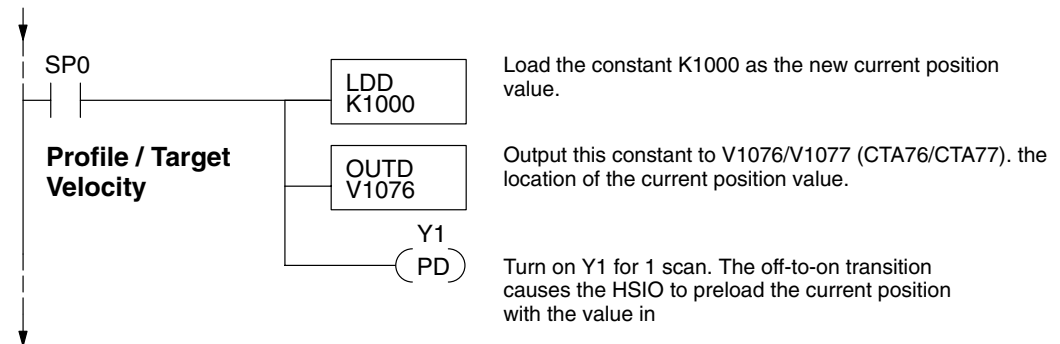


Program  
Example Cont'd



Preload  
Position Value

At any time you can write (preload) a new position into the current position value. This often done after a home search (see the registration example programs).

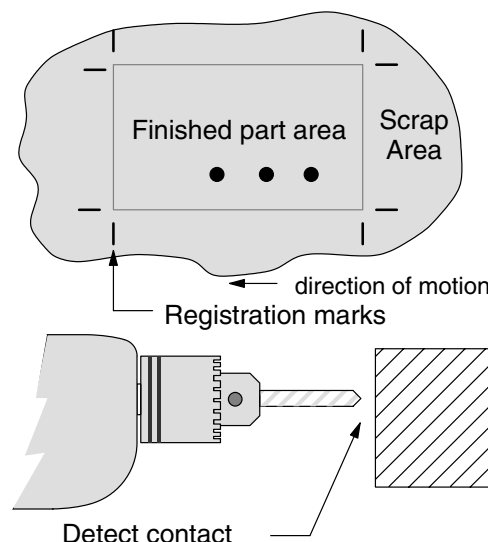


## Registration Profile Operation

### Registration Applications

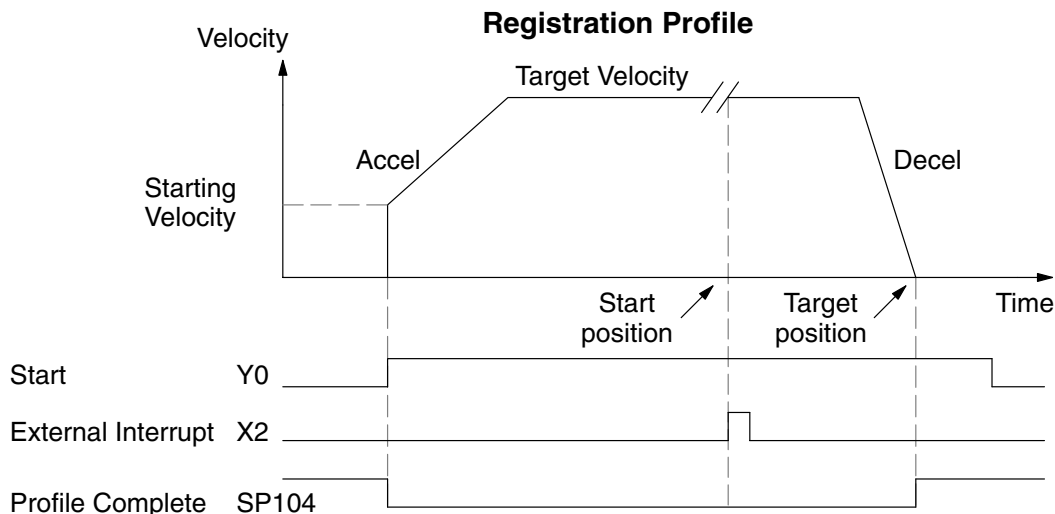
1. In a typical application shown to the right, product material in work moves past a work tool such as a drill. Registration marks on the scrap area of the work-piece allow a machine tool to register its position relative to the rectangle, to drill properly.

2. In other examples of registration, the work piece is stationary and the tool moves. A drill bit may approach the surface of a part in work, preparing to drill a hole of precise depth. However, the drill bit length gradually decreases due to tool wear. A method to overcome this is to detect the moment of contact with the part surface on each drill, moving the bit into the part a constant distance after contact.



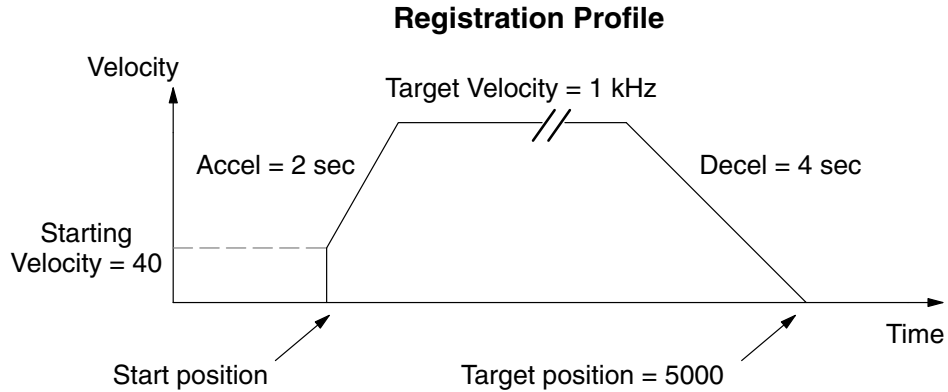
3. The home search move allows a motion system to calibrate its position on startup. In this case, the positioning system makes an indefinite move and waits for the load to pass by a home limit switch. This creates an interrupt at the moment when the load is in a known position. We then stop motion and preload the position value with a number which equates to the physical “home position”.

The registration profile begins with only velocity control. When an interrupt pulse occurs on physical input X2, the starting position is declared to be the present count (current load position). The velocity control switches to position control, moving the load to the target position. Note that the minimum starting velocity is 40 pps. This instantaneous velocity accommodates stepper motors that can stall at low speeds.

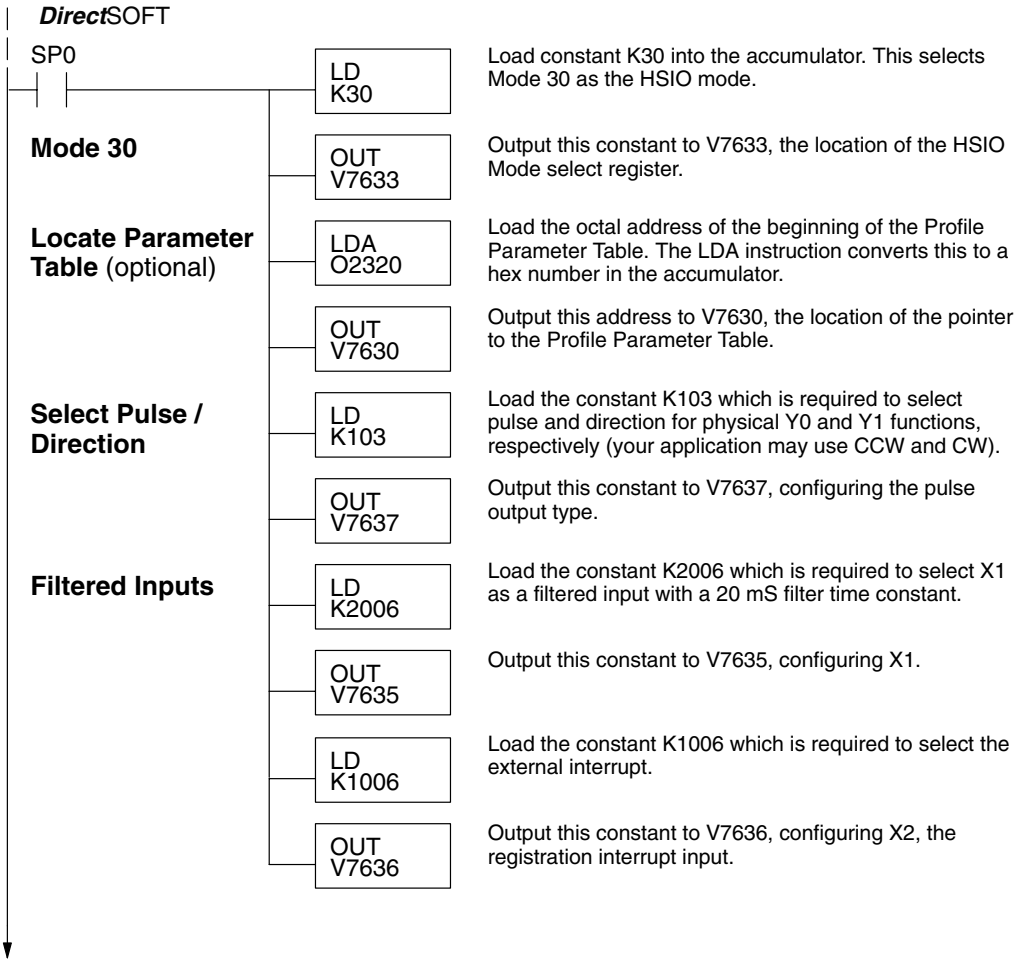


The time line of signal traces below the profile indicates the order of events. The CPU uses logical output Y0 to start the profile. Immediately the HSIO turns off the Profile Complete signal (SP104), so the ladder program can monitor the move's completion by sensing the signal's on state.

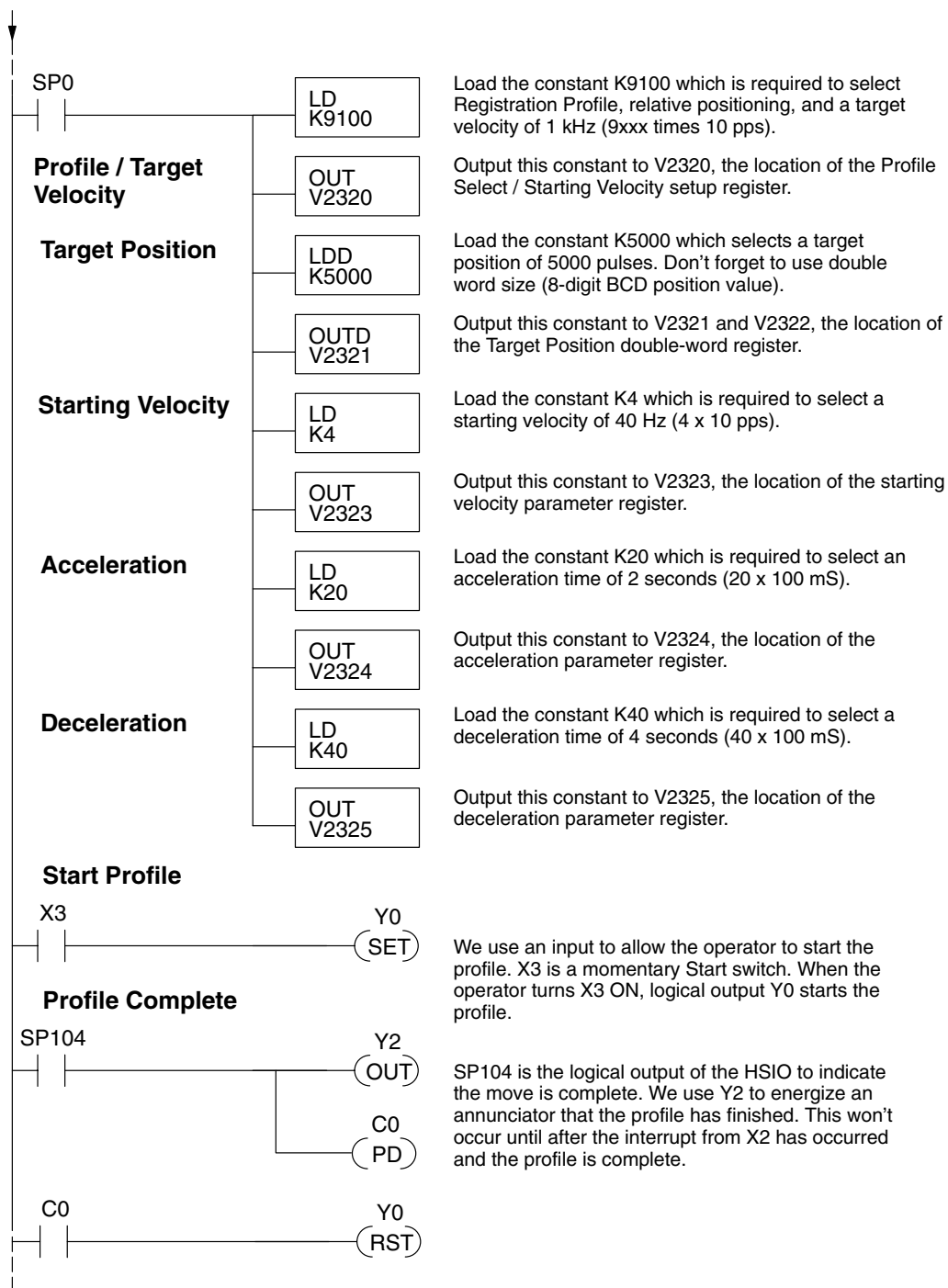
**Registration Profile Program Example** The registration profile we want to perform is drawn and labeled in the following figure. It consists of a non-zero starting velocity, and moderate target velocity.



The following program will realize the profile drawn above, when executed. The first program rung contains all the necessary setup parameters. We only have to do this once in the program, so we use first-scan contact SP0 to trigger the setup.



### Program Example Cont'd

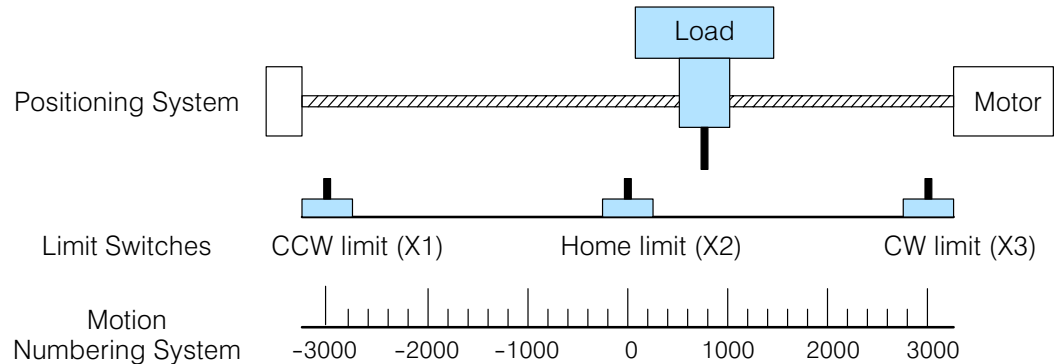


The profile will begin when the start input (X3) is given. Then the motion begins an indefinite move, which lasts until an external interrupt on X2 occurs. Then the motion continues on for 5000 more pulses before stopping.



## Home Search Program Example

One of the more challenging aspects of motion control is the establishment of actual position at powerup. This is especially true for open-loop systems which do not have a position feedback device. However, a simple limit switch located at an exact location on the positioning mechanism can provide “position feedback” at one point. For most stepper control systems, this method is a good and economical solution.

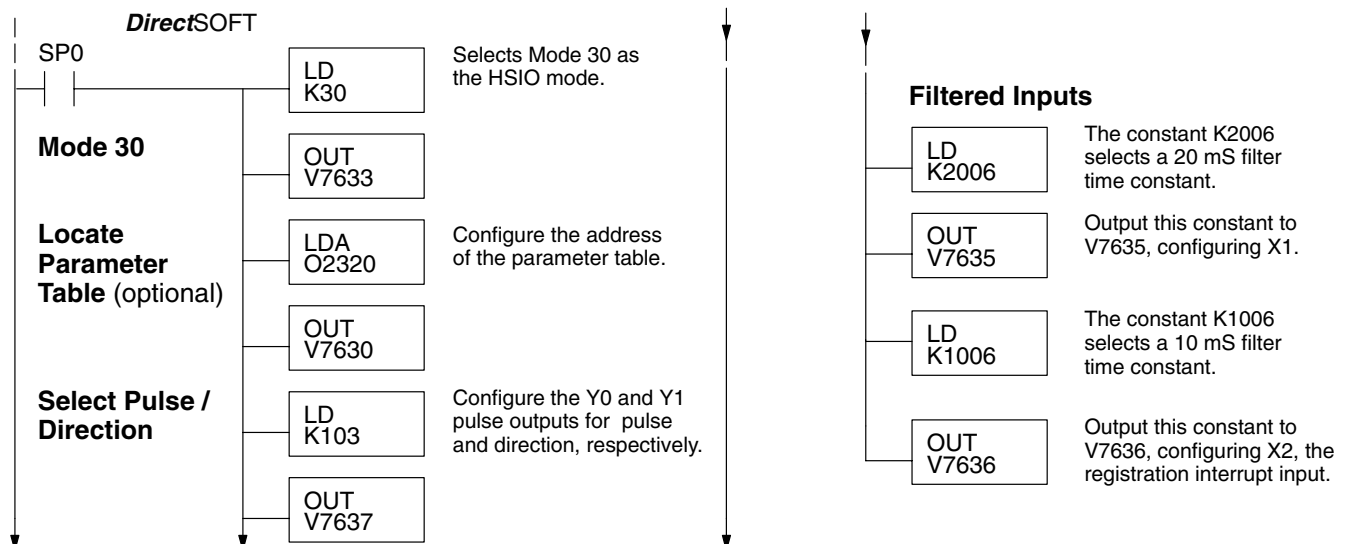


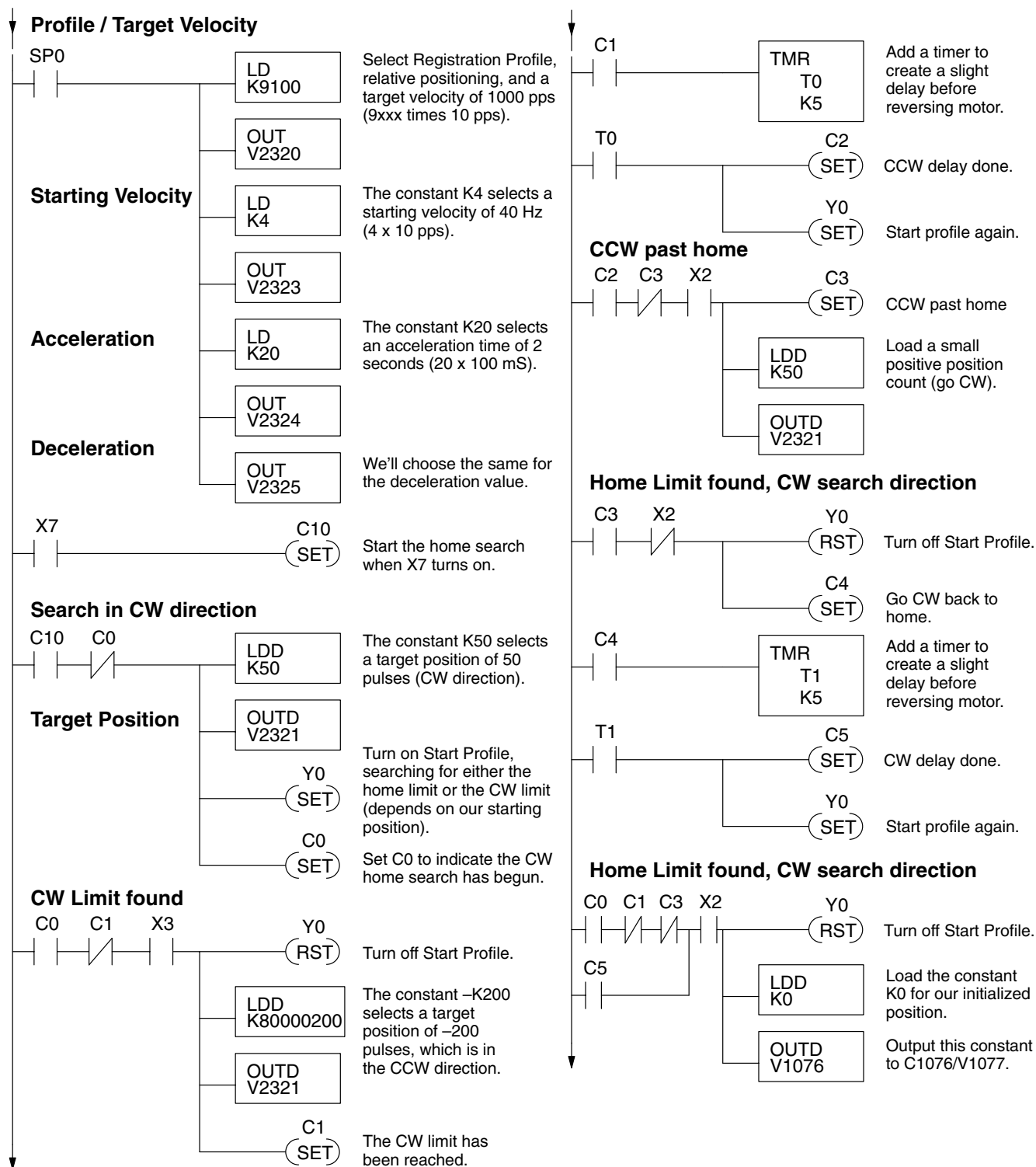
In the drawing above, the load moves left or right depending on the CCW/CW direction of motor rotation. The PLC ladder program senses the CCW and CW limit switches to stop the motor, before the load moves out-of-bounds and damages the machine. The home limit switch is used at powerup to establish the actual position. The numbering system is arbitrary, depending on a machine's engineering units.

At powerup, we do not know whether the load is located to the left or to the right of the home limit switch. Therefore, we will initiate a *home search profile*, using the registration mode. The home limit switch is wired to X2, causing the interrupt. We choose an arbitrary initial search direction, moving in the CW (left-to-right) direction.

- If the home limit switch closes first, then we stop and initialize the position (this value is typically “0”, but it may be different if preferred).
- However, if the CW limit switch closes first, we must reverse the motor and move until the home limit switch closes, stopping just past it.

In the latter case, we repeat the first move, because we always need to make the final approach to the home limit switch *from the same direction*, so that the final *physical* position is the same in either case!





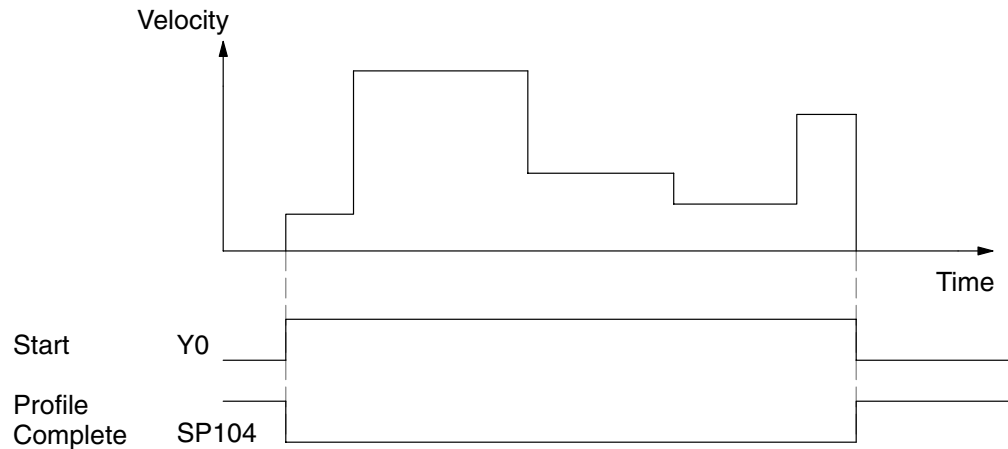
The home search profile will execute specific parts of the program, based on the order of detection of the limit switches. Ladder logic sets C0 to initiate a home search in the CW direction. If the CW limit is encountered, the program searches for home in the CCW direction, passes it slightly, and does the final CW search for home. After reaching home, the last ladder rung preloads the current position to "0".

## Velocity Profile Operation

### Velocity Profile Applications

The velocity profile is best suited for applications which involve motion but do not require moves to specific points. Conveyor speed control is a typical example.

#### Velocity Profile



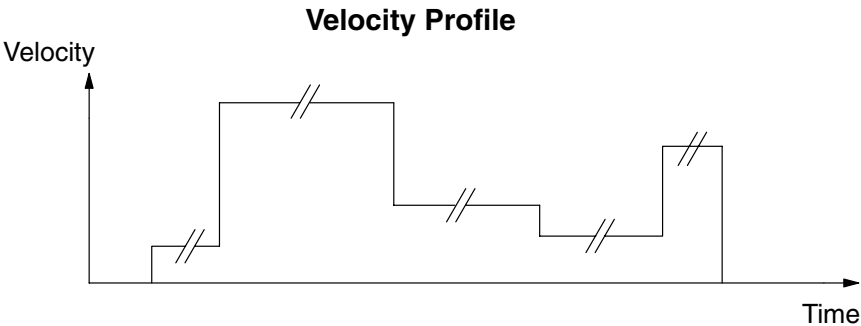
The time line of signal traces below the profile indicates the order of events. Assuming the velocity is set greater than zero, motion begins when the Start input (Y0) energizes. Since there is no end position target, the profile is considered in progress as long as the Start input remains active. The profile complete logical input to ladder logic (X0) correlates directly to the Start input status when velocity profiles are in use.

While the Start input is active, the ladder program can command a velocity change by writing a new value to the velocity register (V2323 by default). The full speed range of 40 Hz to 7 kHz is available. Notice from the drawing that there are no acceleration or deceleration ramps between velocity updates. This is how velocity profiling works with the HSIO. However, the ladder program can command more gradual velocity changes by incrementing or decrementing the velocity value more slowly. A counter or timer can be useful in creating your own acceleration/deceleration ramps. Unless the load must do a very complex move, it is easier to let the HSIO function generate the accel/decel ramps by selecting the trapezoidal or registration profiles instead.

Unlike the trapezoidal and registration profiles, you must specify the desired direction of travel with velocity profiles. Load the direction select register (V2321/V2322 by default) with 8000 0000 hex for CCW direction, or 0 for CW direction.

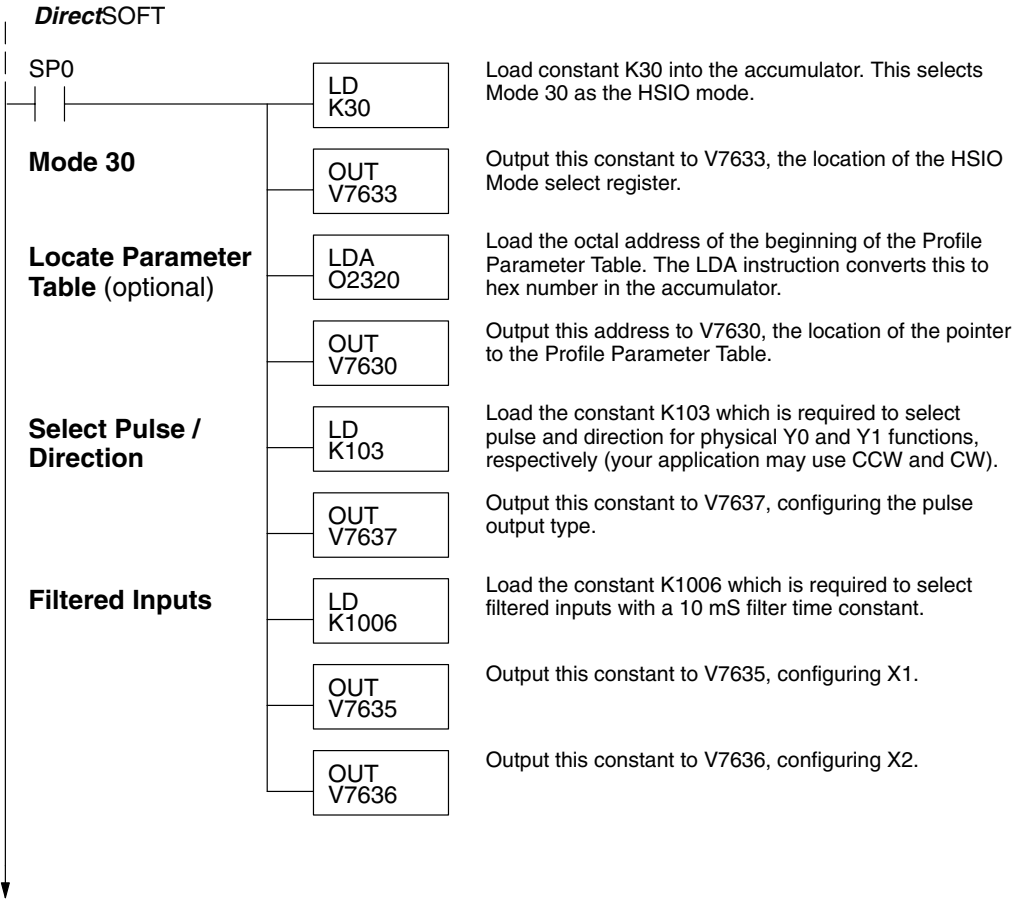
Velocity Profile  
Program Example

The velocity profile we want to perform is drawn and labeled in the following figure. Each velocity segment is of indefinite length. The velocity only changes when ladder logic (or other device writing to V-memory) updates the velocity parameter.

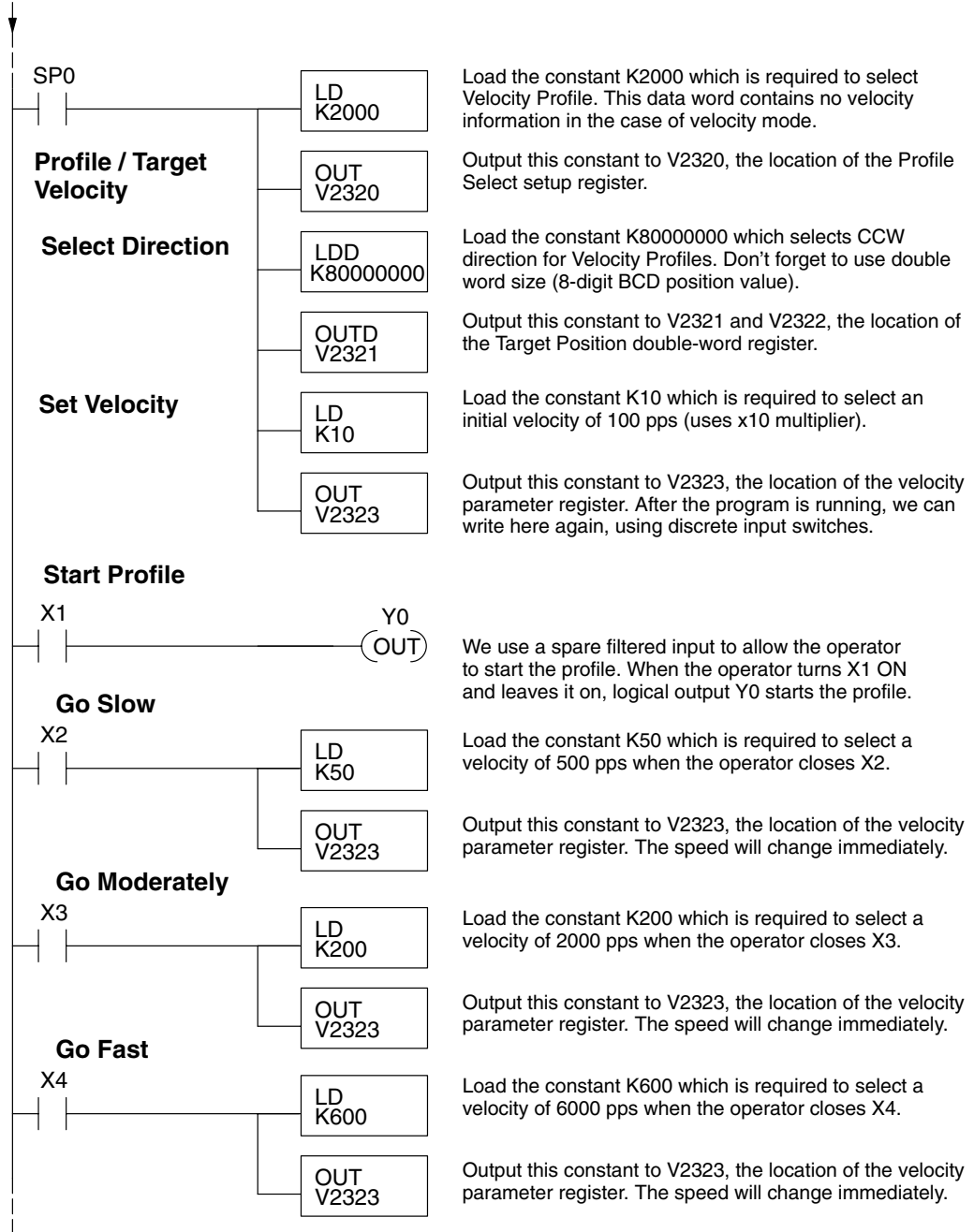


The following program uses dedicated discrete inputs to load in new velocity values. This is a fun program to try, because you can create an infinite variety of profiles with just two or three input switches. The intent is to turn on only one of X1, X2, or X3 at a time. The beginning of the program contains all the necessary setup parameters for Pulse Output Mode 30. We only have to do this once in the program, so we use first-scan contact SP0 to trigger the setup.

High-Speed Input and  
Pulse Output Features



Program  
Example Cont'd



**Pulse Output Error Codes** The Profile Parameter Table starting at V2320 (default location) defines the profile. Certain numbers will result in an error when the HSIO attempts to use the parameters to execute a move profile. When an error occurs, the HSIO writes an error code in V2326.

Error Code	Error Description
0000	No error
0010	Requested profile type code is invalid (must use 0, 1, 2, 8, or 9)
0020	Target Velocity is not in BCD
0021	Target Velocity is specified to be less than 40 pps
0022	Target Velocity is specified to be greater than 7,000 pps
0030	Target Position value is not in BCD
0040	Starting Velocity is not in BCD
0041	Starting Velocity is specified to be less than 40 pps
0042	Starting Velocity is specified to be greater than 1,000 pps
0050	Acceleration Time is not in BCD
0051	Acceleration Time is zero
0052	Acceleration Time is greater than 10 seconds
0060	Deceleration Time is not in BCD
0061	Deceleration Time is zero
0062	Deceleration Time is greater than 10 seconds

Most errors can be corrected by rechecking the Profile Parameter Table values. The error is automatically cleared at powerup and at Program-to-Run Mode transitions.

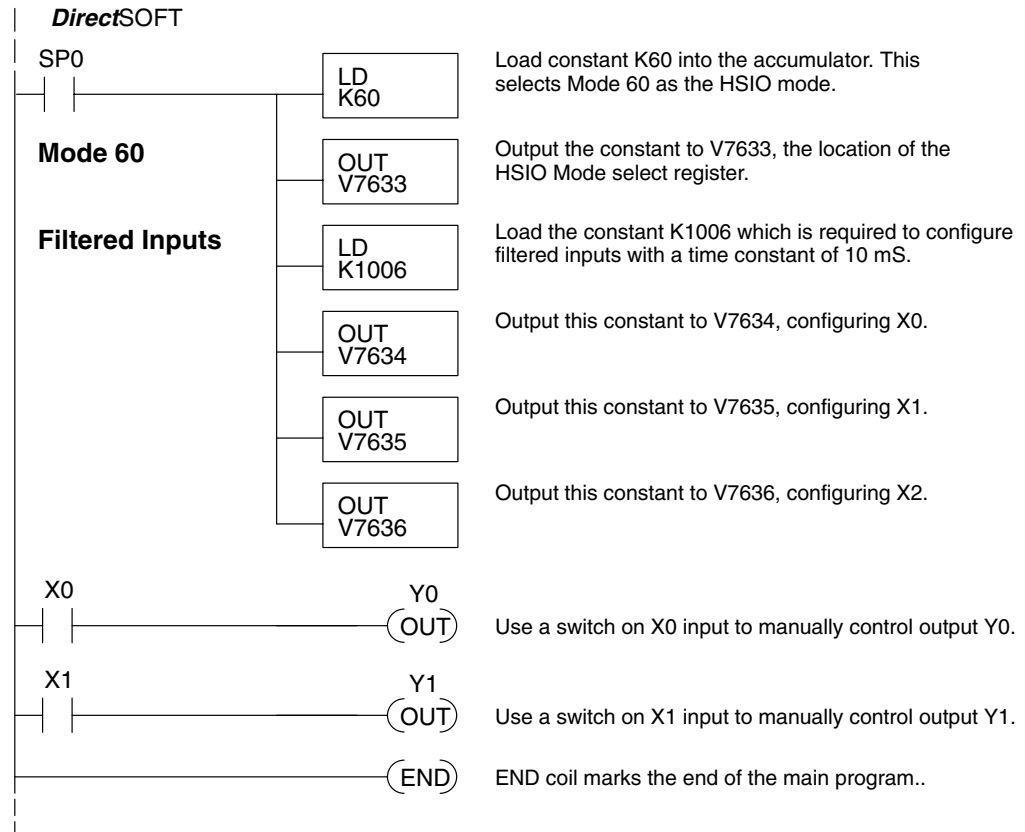
**Troubleshooting Guide for Mode 30** If you're having trouble with Mode 30 operation, please study the following symptoms and possible causes. The most common problems are listed below:

**Symptom: The stepper motor does not rotate.**

Possible causes:

1. **Configuration** – Verify that the HSIO actually generates pulses on outputs Y0 and Y1. Watch the status LEDs for Y0 and Y1 when you start a motion profile. If the LEDs flicker on and off or are steadily on, the configuration is probably correct.
2. **Programming error** – If there are no pulses on Y0 or Y1 you may have a programming error. Check the contents of V2326 for an error code that may be generated when the PLC attempts to do the move profile. Error code descriptions are given above.
3. **Check target value** – The profile will not pulse if the count value is equal to the target value (ex. count =0, target=0)

4. **Wiring** – Verify the wiring to the stepper motor is correct. Remember the signal ground connection from the PLC to the motion system is required.
5. **Motion system** – Verify that the drive is powered and enabled. To verify the motion system is working, you can use Mode 60 operation (normal PLC inputs/outputs) as shown in the test program below. With it, you can manually control Y0 and Y1 with X0 and X1, respectively. Using an input simulator is ideal for this type of manual debugging. With the switches you can single-step the motor in either direction. If the motor will not move with this simple control, Mode 30 operation will not be possible until the problem with the motor drive system or wiring is corrected.



6. **Memory Error** – HSIO configuration parameters are stored in the CPU system memory. Corrupted data in this memory area can sometimes interfere with proper HSIO operation. If all other corrective actions fail, initializing the scratchpad memory may solve the problem. With **DirectSOFT**, select the *PLC* menu, then *Setup*, then *Initialize Scratchpad*.

### Symptom: The motor turns in the wrong direction.

Possible causes:

1. **Wiring** – If you have selected CW and CCW type operation, just swap the wires on Y0 and Y1 outputs.
2. **Direction control** – If you have selected Pulse and Direction type operation, just change the direction bit to the opposite state.

## Mode 40: High-Speed Interrupts

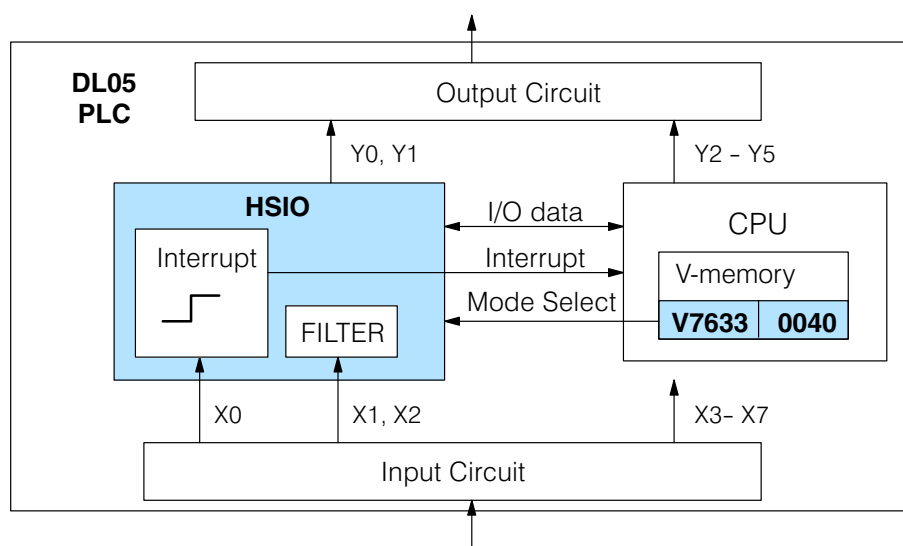
### Purpose

The HSIO Mode 40 provides a high-speed interrupt to the ladder program. This capability is provided for your choice of the following application scenarios:

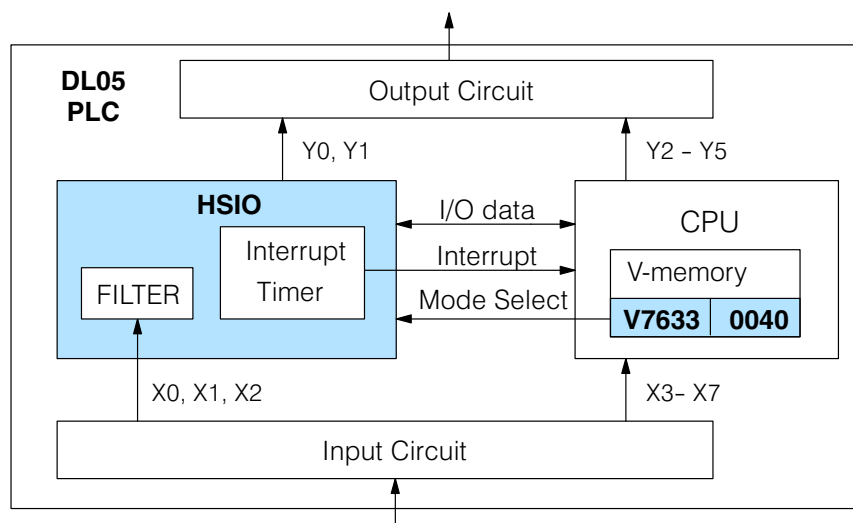
- An external event needs to trigger an interrupt subroutine in the CPU. Using immediate I/O instructions in the subroutine is typical.
- An interrupt routine needs to occur on a timed basis which is different from the CPU scan time (either faster or slower). The timed interrupt is programmable, from 5 to 999 mS.

### Functional Block Diagram

The HSIO circuit creates the high-speed interrupt to the CPU. The following diagram shows the external interrupt option, which uses X0. In this configuration X1 and X2 are normal filtered inputs.

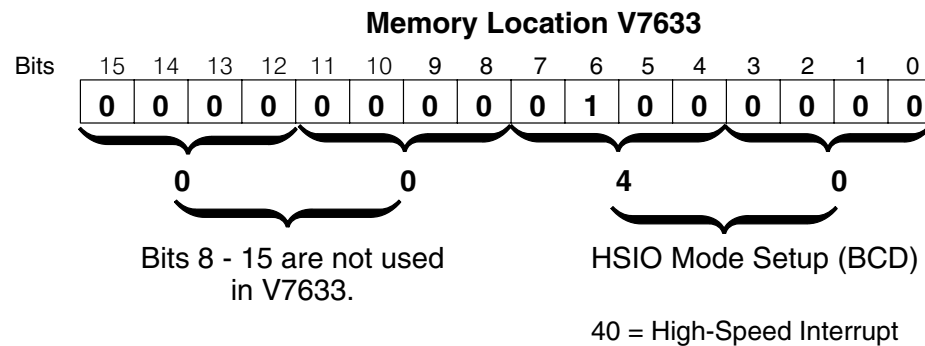


Alternately, you may configure the HSIO circuit to generate interrupts based on a timer, as shown below. In this configuration, inputs X0 through X2 are filtered inputs.





**Setup for Mode 40** Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 40 in the lower byte of V7633 to select the High-Speed Counter Mode. The DL05 does not use bits 8 - 15 in V7633.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- **DirectSOFT**'s memory editor
- Use the Handheld Programmer D2-HPP

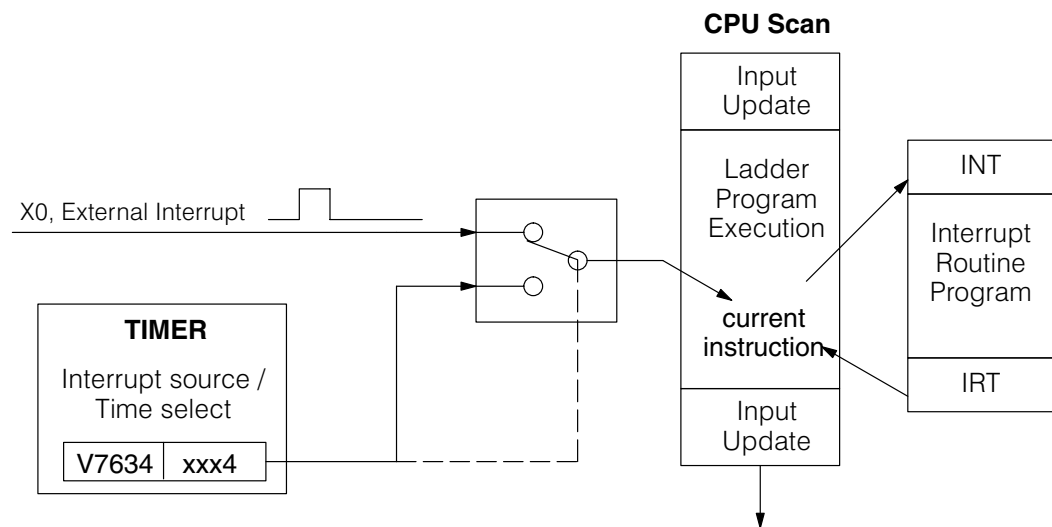
We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

### Interrupts and the Ladder Program

Refer to the drawing below. The source of the interrupt may be external (X0), or the HSIO timer function. The setup parameter in V7634 serves a dual purpose;

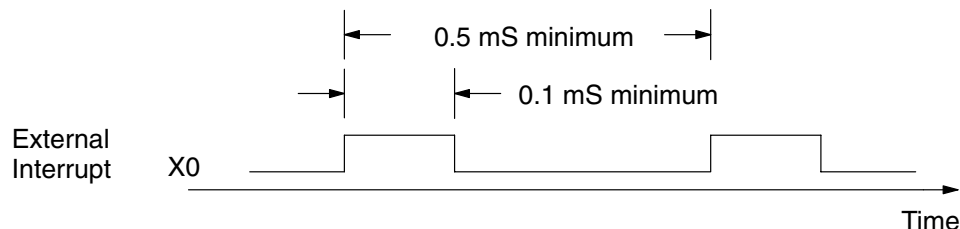
- It selects between the two interrupt sources, external (X0) or an internal timer.
- In the case of the timer interrupt, it programs the interrupt timebase between 5 and 999 mS.

The resulting interrupt uses label INT 0 in the ladder program. Be sure to include the Enable Interrupt (ENI) instruction at the beginning of your program. Otherwise, the interrupt routine will not be executed.

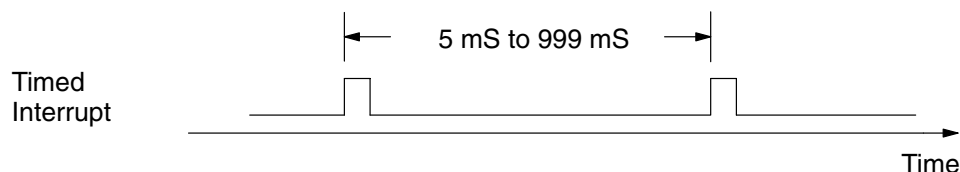


**External Interrupt Timing Parameters**

Signal pulses at X0 must meet certain timing criteria to guarantee an interrupt will result. Refer to the timing diagram below. The input characteristics of X0 are fixed (it is not a programmable filtered input). The minimum pulse width is 0.1 mS. There must be some delay before the next interrupt pulse arrives, such that the interrupt period cannot be smaller than 0.5 mS.

**Timed Interrupt Parameters**

When the timed interrupt is selected, the HSIO generates the interrupt to ladder logic. There is no interrupt “pulse width” in this case, but the interrupt period can be adjusted from 5 to 999 mS.

**X Input / Timed INT Configuration**

The configurable discrete input options for High-Speed Interrupt Mode are listed in the table below. Input X0 is the external interrupt when “0004” is in V7634. If you need a timed interrupt instead, then V7634 contains the interrupt time period, and input X0 becomes a filtered input (uses X1’s filter time constant by default). Inputs X1, and X2, can only be filtered inputs, having individual configuration registers and filter time constants. However, X0 will have the same filter time constant as X1 when the timed interrupt is selected.

Input	Configuration Register	Function	Hex Code Required
X0	V7634	External Interrupt	0004 (default)
	Uses X1’s time setting in V7635	Filtered Input (when timed interrupt is in use)	xxx4, xxx = INT timebase 5 - 999 ms (BCD)
X1	V7635	Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)
X2	V7636	Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)

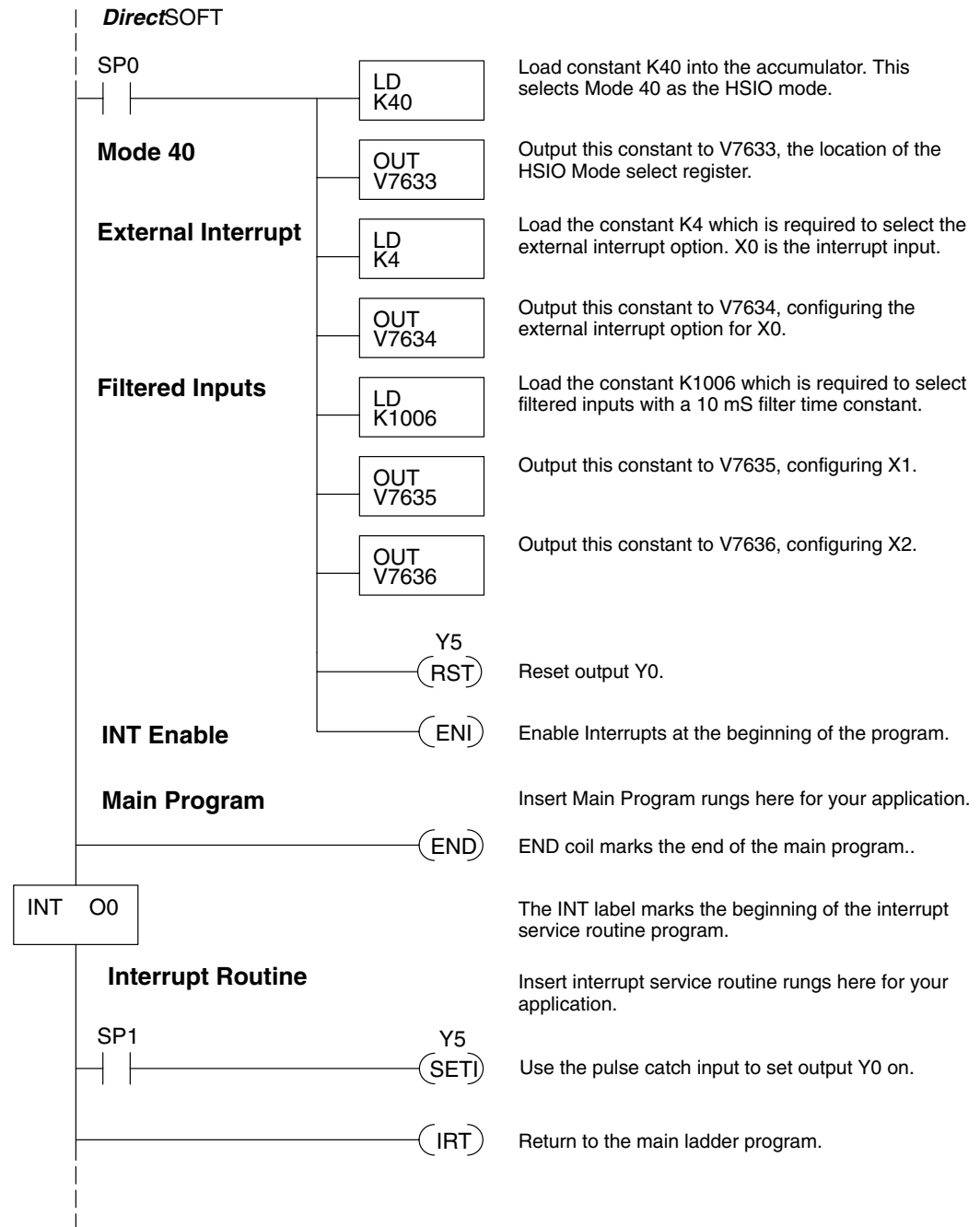
**Independent Timed Interrupt**

Interrupt O1 is also available as an interrupt. This interrupt is independent of the HSIO features. Interrupt O1 uses an internal timer that is configured in V memory location V7647. The interrupt period can be adjusted from 5 to 9999 mS. Once the interrupt period is set and the interrupt is enabled in the program, the CPU will continuously call the interrupt routine based on the time setting in V7647.

Input	Configuration Register	Function	Hex Code Required
—	V7647	High-Speed Timed Interrupt	xxxx (xxxx = timer setting) 5 - 9999 mS (BCD)

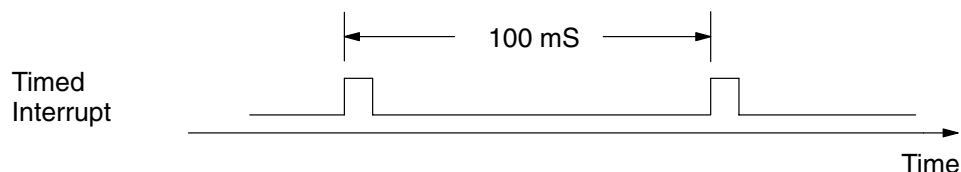
### External Interrupt Program Example

The following program selects Mode 40, then selects the external interrupt option. Inputs X1 and X2 are configured as filtered inputs with a 10 mS time constant. The program is otherwise generic, and may be adapted to your application.

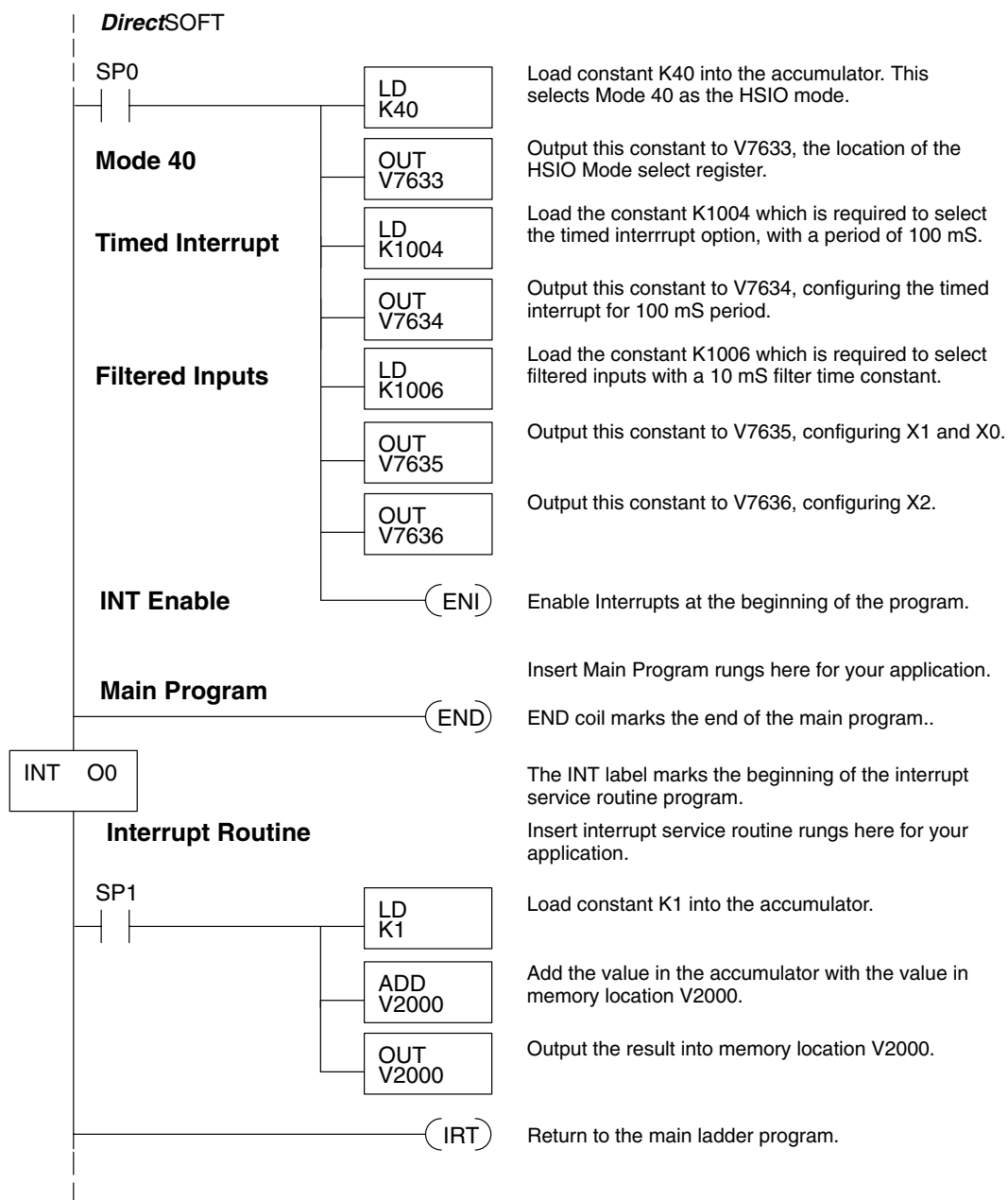


**Timed Interrupt Program Example**

The following program selects Mode 40, then selects the timed interrupt option, with an interrupt period of 100 mS.



Inputs X0, X1, and X2, are configured as filtered inputs with a 10 mS time constant. Note that X0 uses the time constant from X1. The program is otherwise generic, and may be adapted to your application.



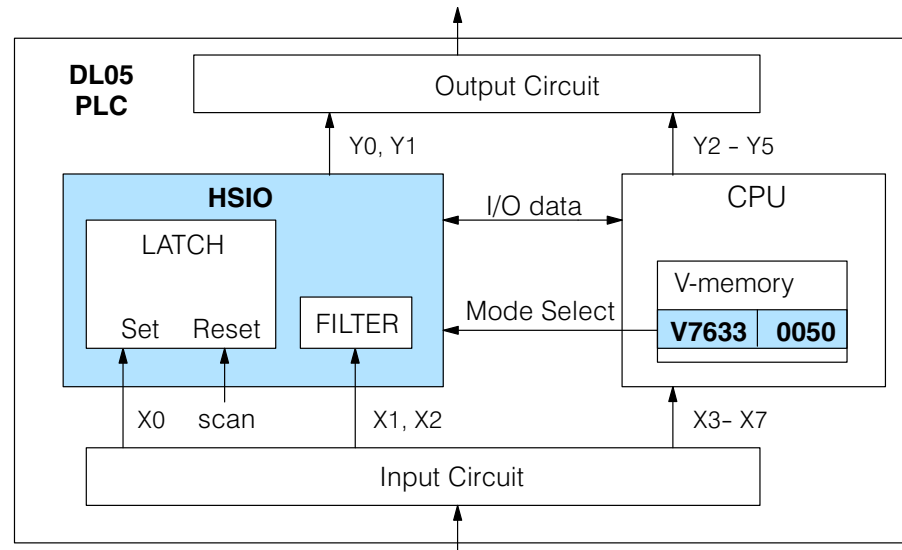
## Mode 50: Pulse Catch Input

### Purpose

The HSIO circuit has a pulse-catch mode of operation. It monitors the signal on input X0, preserving the occurrence of a narrow pulse. The purpose of the pulse catch mode is to enable the ladder program to “see” an input pulse which is shorter in duration than the current scan time. The HSIO circuit latches the input event on input X0 for one scan. This contact automatically goes off after one scan.

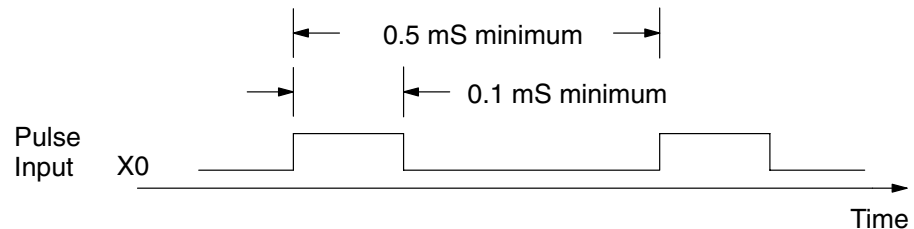
### Functional Block Diagram

Refer to the block diagram below. When the lower byte of HSIO Mode register V7633 contains a BCD “50”, the pulse catch mode in the HSIO circuit is enabled. X0 automatically becomes the pulse catch input, which sets the latch on each rising edge. The HSIO resets the latch at the end of the next CPU scan. Inputs X1 and X2 are available as filtered discrete inputs.



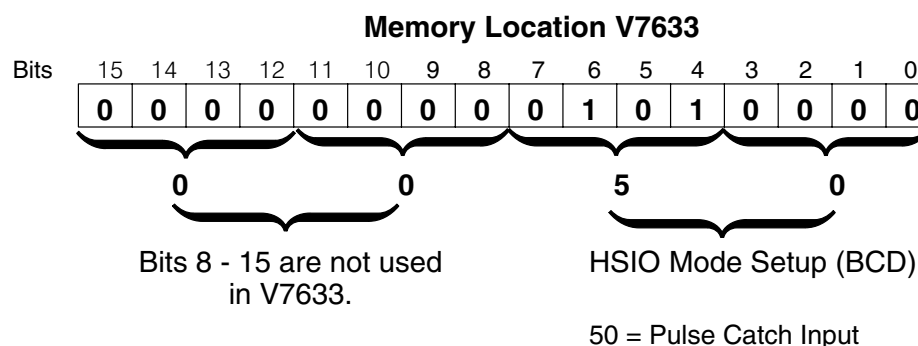
### Pulse Catch Timing Parameters

Signal pulses at X0 must meet certain timing criteria to guarantee a pulse capture will result. Refer to the timing diagram below. The input characteristics of X0 are fixed (it is not a programmable filtered input). The minimum pulse width is 0.1 mS. There must be some delay before the next pulse arrives, such that the pulse period cannot be smaller than 0.5 mS. If the pulse period is smaller than 0.5 mS, the next pulse will be considered part of the current pulse.



Note that the pulse catch and filtered input functions are opposite in nature. The pulse catch feature on X0 seeks to *capture* narrow pulses, while the filter input feature on X1 and X2 seeks to *reject* narrow pulses.

**Setup for Mode 50** Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 50 in the lower byte of V7633 to select the High-Speed Counter Mode. The DL05 does not use bits 8 - 15 in V7633.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- **DirectSOFT**'s memory editor
- Use the Handheld Programmer D2-HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

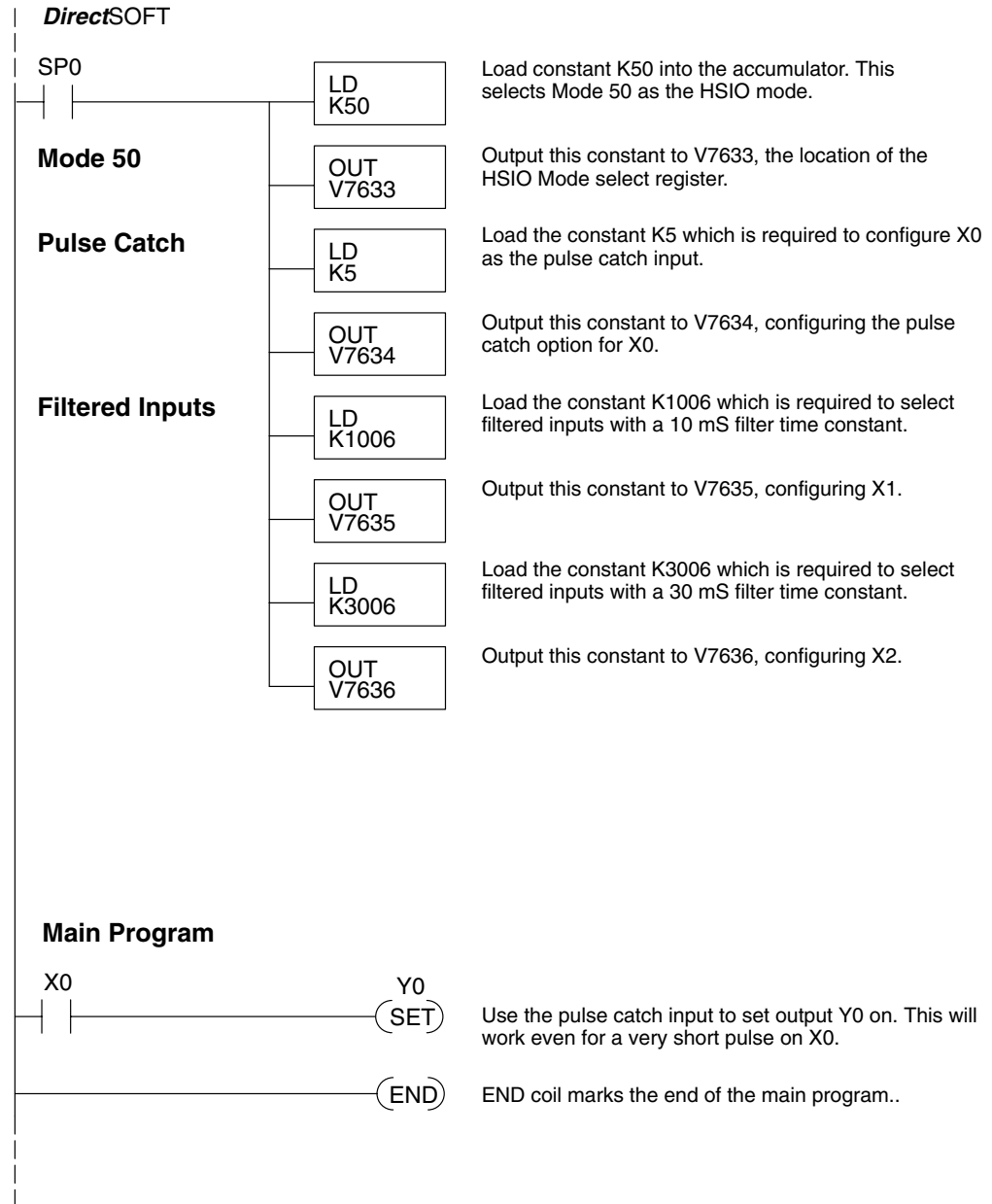
### X Input Configuration

The configurable discrete input options for Pulse Catch Mode are listed in the table below. Input X0 is the pulse input, and must have "0005" loaded into its configuration register V7634. Inputs X1 and X2 can only be filtered inputs. Each input has its own configuration register and filter time constant.

Input	Configuration Register	Function	Hex Code Required
X0	V7634	Pulse Catch Input	0005
X1	V7635	Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)
X2	V7636	Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)

### Pulse Catch Program Example

The following program selects Mode 50, then programs the pulse catch code for X0. Inputs X1 and X2 are configured as filtered inputs with 10 and 30 mS time constants respectively. The program is otherwise generic, and may be adapted to your application.



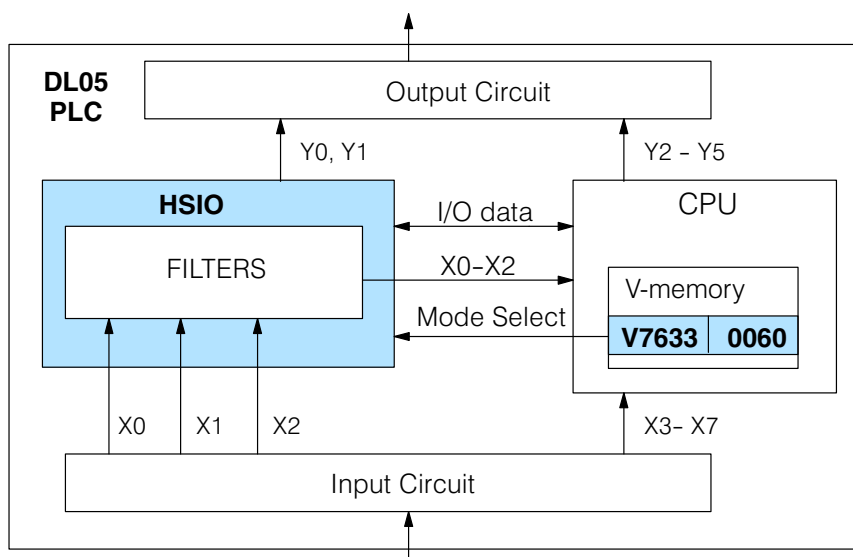
## Mode 60: Discrete Inputs with Filter

### Purpose

The last mode we will discuss for the HSIO circuit is Mode 60, Discrete Inputs with Filter. The purpose of this mode is to allow the input circuit to reject narrow pulses and accept wide ones, as viewed from the ladder program. This is useful in especially noisy environments or other applications where pulse width is important. In all other modes in this chapter, X0 to X2 usually support the mode functions as special inputs. Only spare inputs operate as filtered inputs by default. Now in Mode 60, all three inputs X0 through X2 function only as discrete filtered inputs.

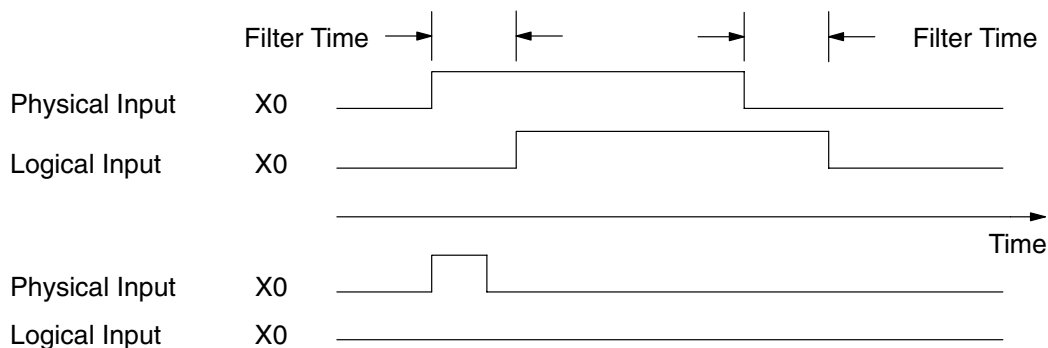
### Functional Block Diagram

Refer to the block diagram below. When the lower byte of HSIO Mode register V7633 contains a BCD “60”, the input filter in the HSIO circuit is enabled. Each input X0 through X2 has its own filter time constant. The filter circuit assigns the outputs of the filters as logical references X0 through X2.



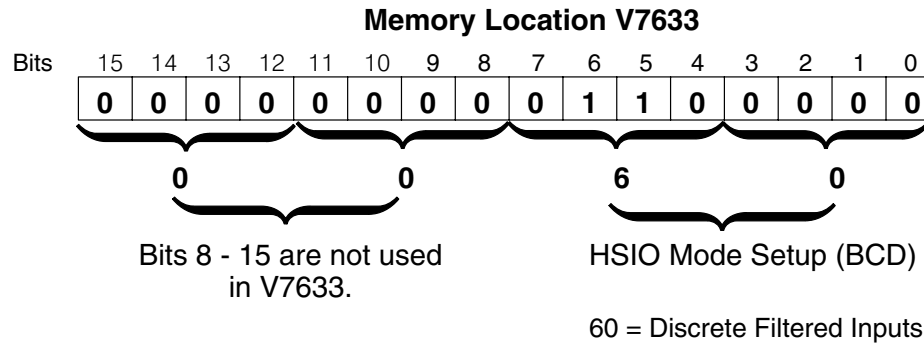
### Input Filter Timing Parameters

Signal pulses at inputs X0 – X2 are filtered by using a delay time. In the figure below, the input pulse on the top line is longer than the filter time. The resultant logical input to ladder is phase-shifted (delayed) by the filter time on both rising and falling edges. In the bottom waveforms, the physical input pulse width is smaller than the filter time. In this case, the logical input to the ladder program remains in the OFF state (input pulse was filtered out).





**Setup for Mode 60** Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 60 in the lower byte of V7633 to select the High-Speed Counter Mode. The DL05 does not use bits 8 - 15 in V7633.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- **DirectSOFT**'s memory editor
- Use the Handheld Programmer D2-HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

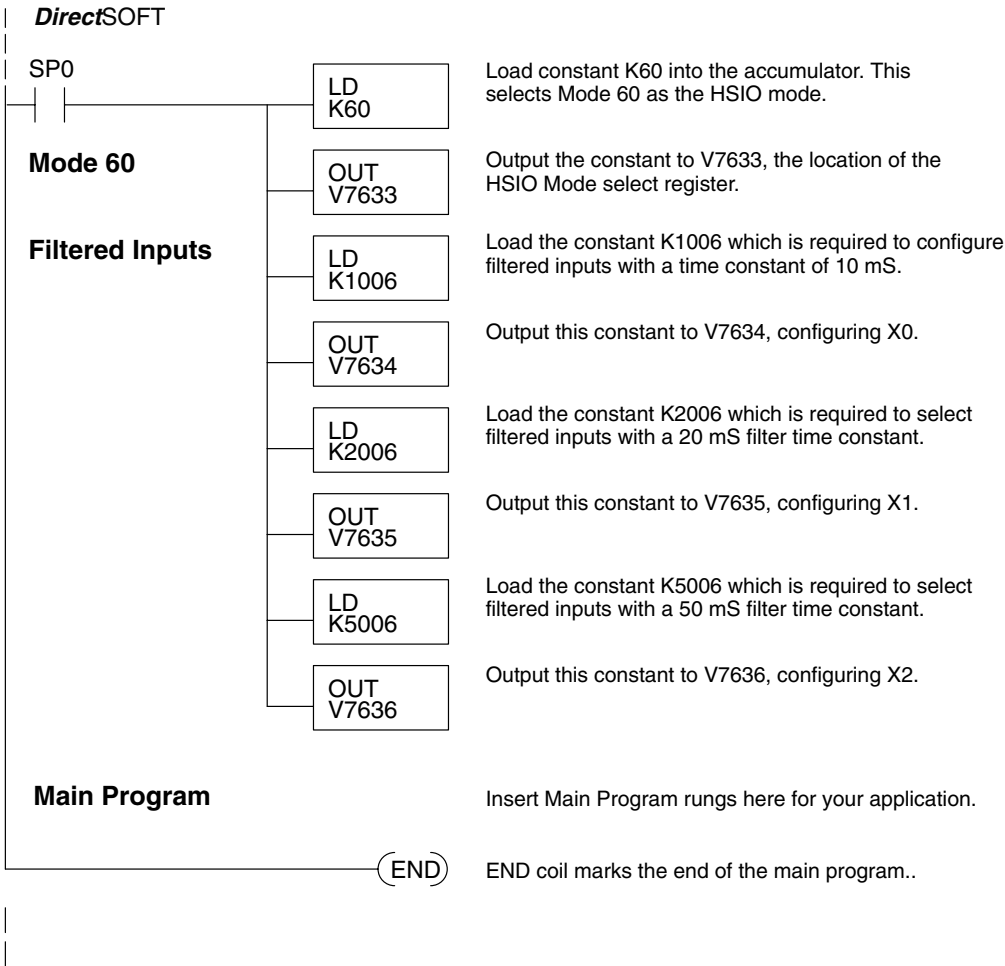
## X Input Configuration

The configurable discrete input options for Discrete Filtered Inputs Mode are listed in the table below. The filter time constant (delay) is programmable from 0 to 99 ms (the input acts as a normal discrete input when the time constant is set to 0). The code for this selection occupies the upper byte of the configuration register in BCD. We combine this number with the required "06" in the lower byte to get "xx06", where xx = 0 to 99. Input X0, X1, and X2 can only be filtered inputs. Each input has its own configuration register and filter time constant.

Input	Configuration Register	Function	Hex Code Required
X0	V7634	Filtered Input	xx06 (xx = filter delay time) 0 - 99 ms (BCD)
X1	V7635	Filtered Input	xx06 (xx = filter delay time) 0 - 99 ms (BCD)
X2	V7636	Filtered Input	xx06 (xx = filter delay time) 0 - 99 ms (BCD)

Filtered Inputs  
Program Example

The following program selects Mode 60, then programs the filter delay time constants for inputs X0, X1, and X2. Each filter time constant is different, for illustration purposes. The program is otherwise generic, and may be adapted to your application.



# CPU Specifications and Operation

---

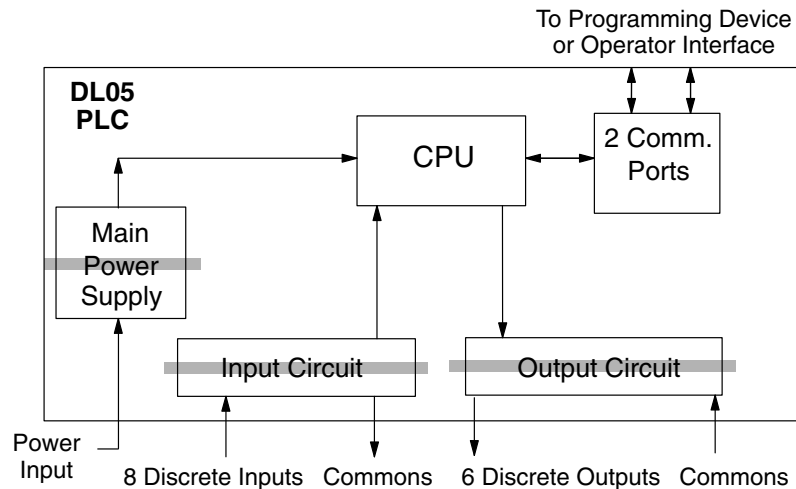
## In This Chapter. . . .

- Introduction
  - CPU Specifications
  - CPU Hardware Setup
  - CPU Operation
  - Program Mode Operation
  - Run Mode Operation
  - I/O Response Time
  - CPU Scan Time Considerations
  - PLC Numbering Systems
  - Memory Map
  - DL05 System V-Memory
  - X Input Bit Map
  - Y Output Bit Map
  - Stage™ Control / Status Bit Map
  - Control Relay Bit Map
  - Timer Status Bit Map
  - Counter Status Bit Map
  - Network Configuration
  - Network Slave Operation
  - Network Master Operation
-

## Introduction

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

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



**NOTE:** The High-Speed I/O function (HSIO) consists of dedicated but configurable hardware in the DL05. It is not considered part of the CPU, because it does not execute the ladder program. For more on HSIO operation, see Chapter 3.

### DL05 CPU Features

The DL05 Micro PLC which has 6K words of memory comprised of 2.0K of ladder memory and 4K words of V-memory (data registers). Program storage is in the FLASH memory which is a part of the CPU board in the PLC. In addition, there is RAM with the CPU which will store system parameters, V-memory, and other data which is not in the application program. The RAM is backed up by a “super-capacitor”, storing the data for several hours in the event of a power outage. The capacitor automatically charges during powered operation of the PLC.

The DL05 supports fixed I/O which includes eight discrete input points and six output points. No provision for expansion beyond these fourteen I/O points is available in the DL05 model PLCs.

Over 120 different instructions are available for program development as well as extensive internal diagnostics that can be monitored from the application program or from an operator interface. Chapters 5, 6, and 7 provide detailed descriptions of the instructions.

The DL05 provides two built-in RS232C communication ports, so you can easily connect a handheld programmer, operator interface, or a personal computer without needing any additional hardware.

## CPU Specifications

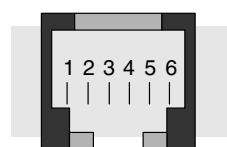
Feature	DL05
Total Program memory (words)	6K
Ladder memory (words)	2048
Total V-memory (words)	4096
User V-memory (words)	3968
Non-volatile V Memory (words)	128
Contact execution (boolean)	2.0uS
Typical scan (boolean)	2.7–3.2mS
RLL Ladder style Programming	Yes
RLL and RLL <sup>PLUS</sup> Programming	Yes
Run Time Edits	Yes
Scan	Variable / fixed
Handheld programmer	Yes
<b>DirectSOFT™</b> programming for Windows™	Yes
Built-in communication ports (RS232C)	Yes
FLASH Memory	Standard on CPU
Local Discrete I/O points available	14
Local Analog input / output channels maximum	None
High-Speed I/O (quad., pulse out, interrupt, pulse catch, etc.)	Yes, 2
I/O Point Density	8 inputs, 6 outputs
Number of instructions available (see Chapter 5 for details)	129
Control relays	512
Special relays (system defined)	512
Stages in RLL <sup>PLUS</sup>	256
Timers	128
Counters	128
Immediate I/O	Yes
Interrupt input (external / timed)	Yes
Subroutines	Yes
For/Next Loops	Yes
Math	Integer
Drum Sequencer Instruction	Yes
Time of Day Clock/Calendar	No
Internal diagnostics	Yes
Password security	Yes
System error log	No
User error log	No
Battery backup	No (built-in super-cap) Yes, with mem cartridge

## CPU Hardware Setup

### Communication Port Pinout Diagrams

Cables are available that allow you to quickly and easily connect a Handheld Programmer or a personal computer to the DL05 PLCs. However, if you need to build your own cables, use the pinout diagrams shown. The DL05 PLCs require an RJ-12 phone plug to fit the built-in jacks.

The Micro PLC has two built-in RS232C communication ports. Port 1 is generally used for connecting to a D2-HPP, **DirectSOFT™**, operator interface, MODBUS slave, or a DirectNET slave. The baud rate is fixed at 9600 baud for port 1. Port 2 can be used to connect to a D2-HPP, **DirectSOFT™**, operator interface, MODBUS master/slave, or a DirectNET master/slave. Port 2 has a range of speeds from 300 baud to 38.4K baud.



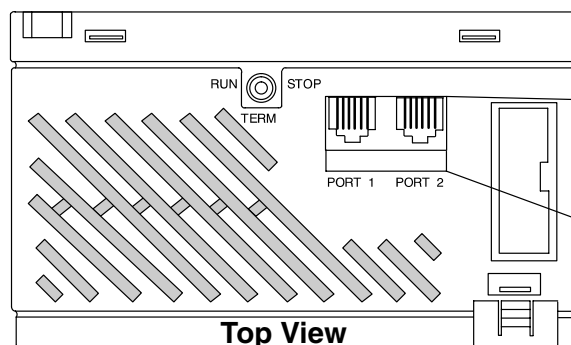
6-pin Female Modular Connector

#### Port 1 Pin Descriptions

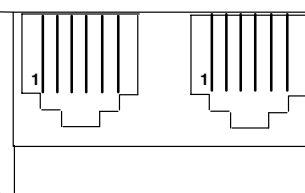
1	0V	Power (–) connection (GND)
2	5V	Power (+) connection
3	RXD	Receive Data (RS232C)
4	TXD	Transmit Data (RS232C)
5	5V	Power (+) connection
6	0V	Power (–) connection (GND)

#### Port 2 Pin Descriptions

1	0V	Power (–) connection (GND)
2	5V	Power (+) connection
3	RXD	Receive Data (RS232C)
4	TXD	Transmit Data (RS232C)
5	RTS	Request to Send
6	0V	Power (–) connection (GND)



Top View



PORT 1 PORT 2

#### Communication Port 1

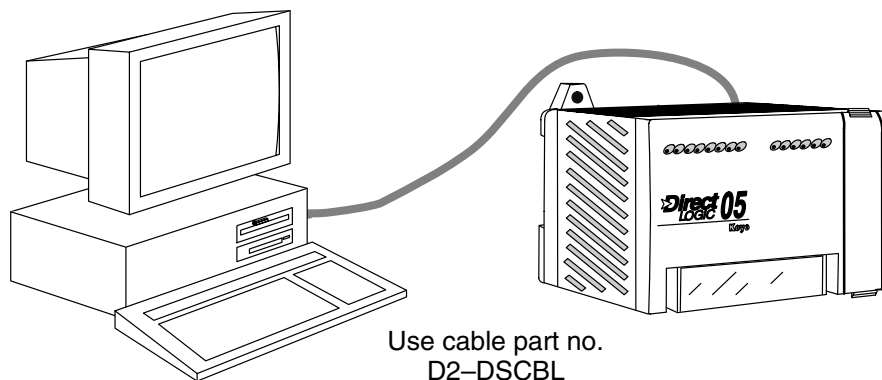
Com 1 Connects to HPP, **DirectSOFT**, operator interfaces, etc.  
 6-pin, RS232C  
 9600 Baud (Fixed)  
 Parity - odd (default)  
 Station address 1 (fixed)  
 8 data bits  
 1 start, 1 stop bit  
 Asynchronous, Half-duplex, DTE  
 Protocol: (Auto-Select)  
     K sequence (Slave only)  
     DirectNET (Slave only)  
     MODBUS (Slave only)

#### Communication Port 2

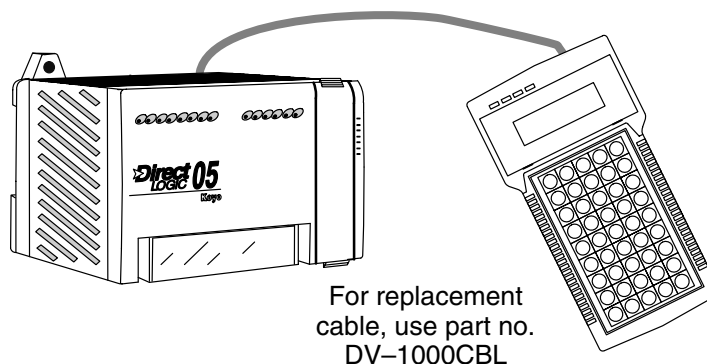
Com 2 Connects to HPP, **DirectSOFT**, operator interfaces, etc.  
 6-pin, RS232C  
 Communication speed (baud)  
     300, 600, 1200, 2400, 4800, 9600, 19200, 38400  
 Parity - odd (default), even, none  
 Station address 1 (default)  
 8 data bits  
 1 start, 1 stop bit  
 Asynchronous, Half-duplex, DTE  
 Protocol: (Auto-Select)  
     K sequence (Slave only)  
     DirectNET (Master/Slave)  
     MODBUS (Master/Slave)  
     Non-sequence/Print

## Connecting the Programming Devices

If you're using a Personal Computer with the **DirectSOFT™** programming package, you can connect the computer to either of the DL05's programming ports. For an engineering office environment (typical during program development), this is the preferred method of programming.



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

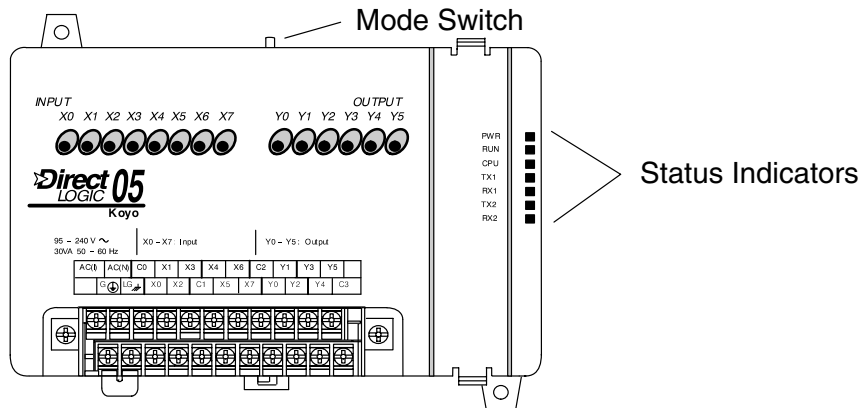


## CPU Setup Information

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

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

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



### Status Indicators

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

Indicator	Status	Meaning
PWR	ON	Power good
	OFF	Power failure
RUN	ON	CPU is in Run Mode
	OFF	CPU is in Stop or program Mode
CPU	ON	CPU self diagnostics error
	OFF	CPU self diagnostics good
TX1	ON	Data is being transmitted by the CPU - Port 1
	OFF	No data is being transmitted by the CPU - Port 1
RX1	ON	Data is being received by the CPU - Port 1
	OFF	No data is being received by the CPU - Port 1
TX2	ON	Data is being transmitted by the CPU - Port 2
	OFF	No data is being transmitted by the CPU - Port 2
RX2	ON	Data is being received by the CPU - Port 2
	OFF	No data is being received by the CPU - Port 2

### Mode Switch Functions

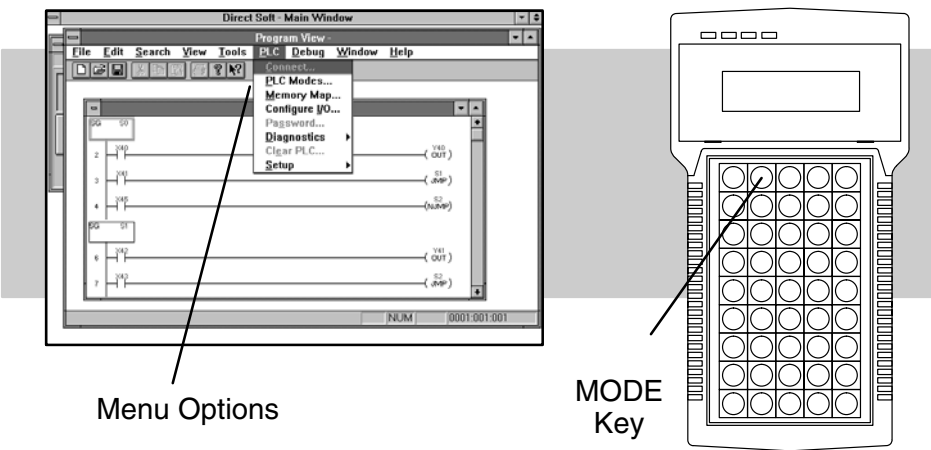
The mode switch on the DL05 PLC provides positions for enabling and disabling program changes in the CPU. Unless the mode switch is in the TERM position, RUN and STOP mode changes will not be allowed by any interface device, (handheld programmer, **DirectSOFT** programing package or operator interface). Programs may be viewed or monitored but no changes may be made. If the switch is in the TERM position and no program password is in effect, all operating modes as well as program access will be allowed through the connected programming or monitoring device.



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

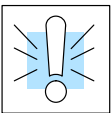
**Changing Modes in the DL05 PLC**

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



**Mode of Operation at Power-up**

The DL05 CPU will normally power-up in the mode that it was in just prior to the power interruption. For example, if the CPU was in Program Mode when the power was disconnected, the CPU will power-up in Program Mode (see warning note below).



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

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

You can access the AUX Functions from **DirectSOFT™** 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 **DirectSOFT** package. The following table shows a list of the Auxiliary functions for the Handheld Programmer.

AUX 2* — RLL Operations		5B	HSIO Configuration
21	Check Program	5D	Scan Control Setup
22	Change Reference	AUX 6* — Handheld Programmer Configuration	
23	Clear Ladder Range	61	Show Revision Numbers
24	Clear All Ladders	62	Beeper On / Off
AUX 3* — V-Memory Operations		65	Run Self Diagnostics
31	Clear V Memory	AUX 7* — EEPROM Operations	
AUX 4* — I/O Configuration		71	Copy CPU memory to HPP EEPROM
41	Show I/O Configuration	72	Write HPP EEPROM to CPU
AUX 5* — CPU Configuration		73	Compare CPU to HPP EEPROM
51	Modify Program Name	74	Blank Check (HPP EEPROM)
53	Display Scan Time	75	Erase HPP EEPROM
54	Initialize Scratchpad	76	Show EEPROM Type (CPU and HPP)
55	Set Watchdog Timer	AUX 8* — Password Operations	
56	Set Communication Port 2	81	Modify Password
57	Set Retentive Ranges	82	Unlock CPU
58	Test Operations	83	Lock CPU
59	Override Setup		

### Clearing an Existing Program

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

You can also use other AUX functions to clear other memory areas.

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

### Initializing System Memory

The DL05 Micro PLC maintain system parameters in a memory area often referred to as the “scratchpad”. In some cases, you may make changes to the system setup that will be stored in system memory. For example, if you specify a range of Control Relays (CRs) as retentive, these changes are stored in system memory.

AUX 54 resets the system memory to the default values.



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

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

### Setting Retentive Memory Ranges

The DL05 PLCs provide certain ranges of retentive memory by default. The default ranges are suitable for many applications, but you can change them if your application requires additional retentive ranges or no retentive ranges at all. The default settings are:

Memory Area	DL05	
	Default Range	Available Range
Control Relays	C400 – C777	C0 – C777
V Memory	V1400 – V7777	V0 – V7777
Timers	None by default	T0 – T177
Counters	CT0 – CT177	CT0 – CT177
Stages	None by default	S0 – S377

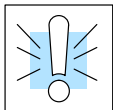
You can use AUX 57 to set the retentive ranges. You can also use **DirectSOFT™** menus to select the retentive ranges. Appendix A contains detailed information about auxiliary



**WARNING:** The DL05 PLCs do not have battery back-up (unless the memory cartridge, D0-01MC, is installed) The super capacitor will retain the values in the event of a power loss, but only for a short period of time, depending on conditions.

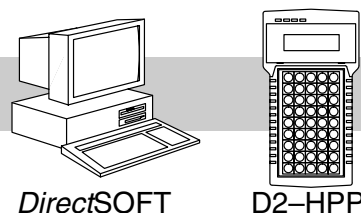
**Using a Password** The DL05 PLCs allow you to use a password to help minimize the risk of unauthorized program and/or data changes. Once you enter a password you can “lock” the PLC against access. Once the CPU is locked you must enter the password before you can use a programming device to change any system parameters.

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



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

You can use the D2-HPP Handheld Programmer or **DirectSOFT™** to enter a password. The following diagram shows how you can enter a password with the Handheld Programmer.



#### Select AUX 81



PASSWORD  
00000000

#### Enter the new 8-digit password



PASSWORD  
XXXXXXXX

#### Press CLR to clear the display

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

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

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

## CPU Operation

Achieving the proper control for your equipment or process requires a good understanding of how DL05 CPUs control all aspects of system operation. There are four main areas to understand before you create your application program:

- **CPU Operating System** — the CPU manages all aspects of system control. A quick overview of all the steps is provided in the next section.
- **CPU Operating Modes** — The two primary modes of operation are Program Mode and Run Mode.
- **CPU Timing** — The two important areas we discuss are the I/O response time and the CPU scan time.
- **CPU Memory Map** — DL05 CPUs offer a wide variety of resources, such as timers, counters, inputs, etc. The memory map section shows the organization and availability of these data types.

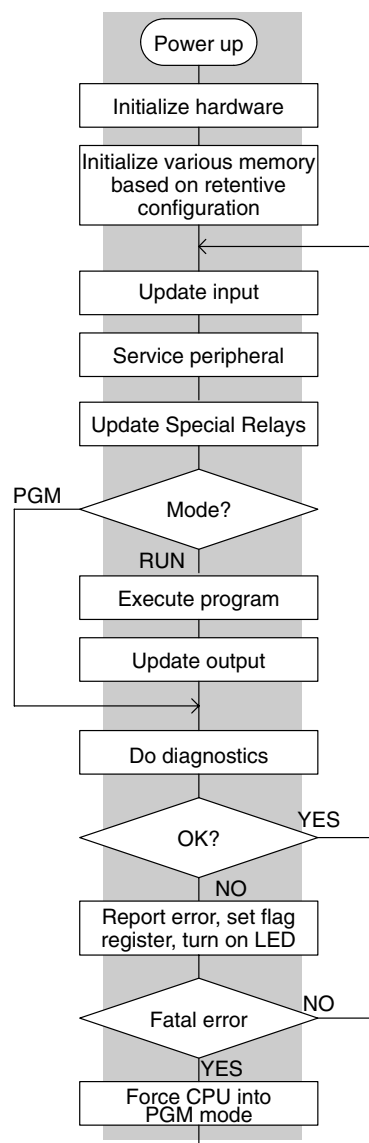
### CPU Operating System

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

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

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

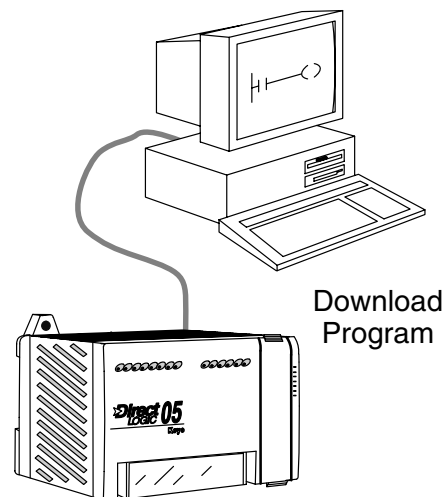
Error detection has two levels. Non-fatal errors are reported, but the CPU remains in its current mode. If a fatal error occurs, the CPU is forced into program mode and the outputs go off.



## Program Mode

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

You can use a programming device, such as **DirectSOFT** or the D2-HPP Handheld Programmer to place the CPU in Program Mode.



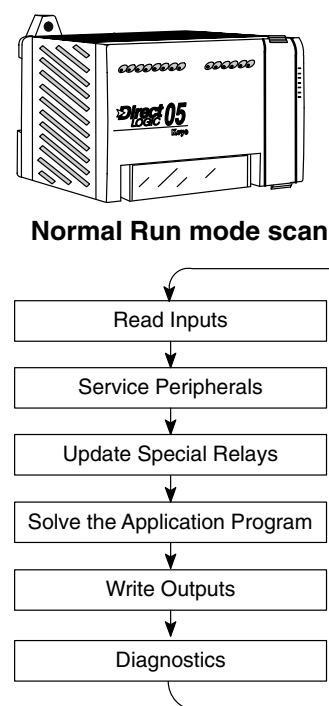
## Run Mode

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

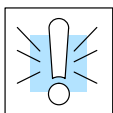
- Monitor and change I/O point status
- Update timer/counter preset values
- Update Variable memory locations

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

You can use **DirectSOFT** or the D2-HPP Handheld Programmer to place the CPU in Run Mode.



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



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

**Read Inputs**

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

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

**Service Peripherals and Force I/O**

After the CPU reads the inputs from the input modules, it reads any attached peripheral devices. This is primarily a communications service for any attached devices. For example, it would read a programming device to see if any input, output, or other memory type status needs to be modified. There are two basic types of forcing available with the DL05 CPUs.

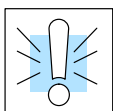
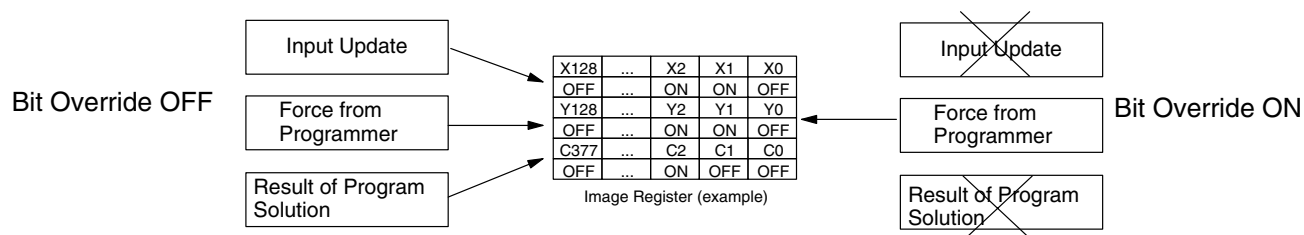
- Forcing from a peripheral – not a permanent force, good only for one scan
- Bit Override – holds the I/O point (or other bit) in the current state. Valid bits are X, Y, C, T, CT, and S. (These memory types are discussed in more detail later in this chapter).

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

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

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

The following diagram shows a brief overview of the bit override feature. Notice the CPU does not update the Image Register when bit override is enabled.



**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.

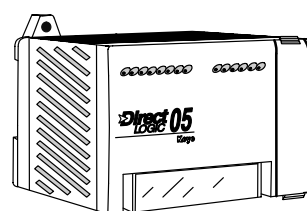
### Update Special Relays and Special Registers

There are certain V-memory locations that contain Special Relays and other dedicated register information. This portion of the execution cycle makes sure these locations get updated on every scan. Also, there are several different Special Relays, such as diagnostic relays, etc., that are also updated during this segment.

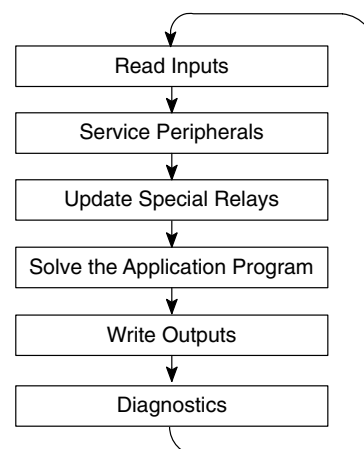
### Solve Application Program

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

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



### Normal Run mode scan



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



**Write Outputs**

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

**Diagnostics**

During this part of the scan, the CPU performs all system diagnostics and other tasks such as calculating the scan time and resetting the watchdog timer. There are many different error conditions that are automatically detected and reported by the DL05 PLCs. Appendix B contains a listing of the various error codes.

Probably one of the more important things that occurs during this segment is the scan time calculation and watchdog timer control. The DL05 CPU has a “watchdog” timer that stores the maximum time allowed for the CPU to complete the solve application segment of the scan cycle. If this time is exceeded the CPU will enter the Program Mode and turn off all outputs. The default value set from the factory is 200 ms. An error is automatically reported. For example, the Handheld Programmer would display the following message “E003 S/W TIMEOUT” when the scan overrun occurs.

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

## I/O Response Time

**Is Timing Important for Your Application?**

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

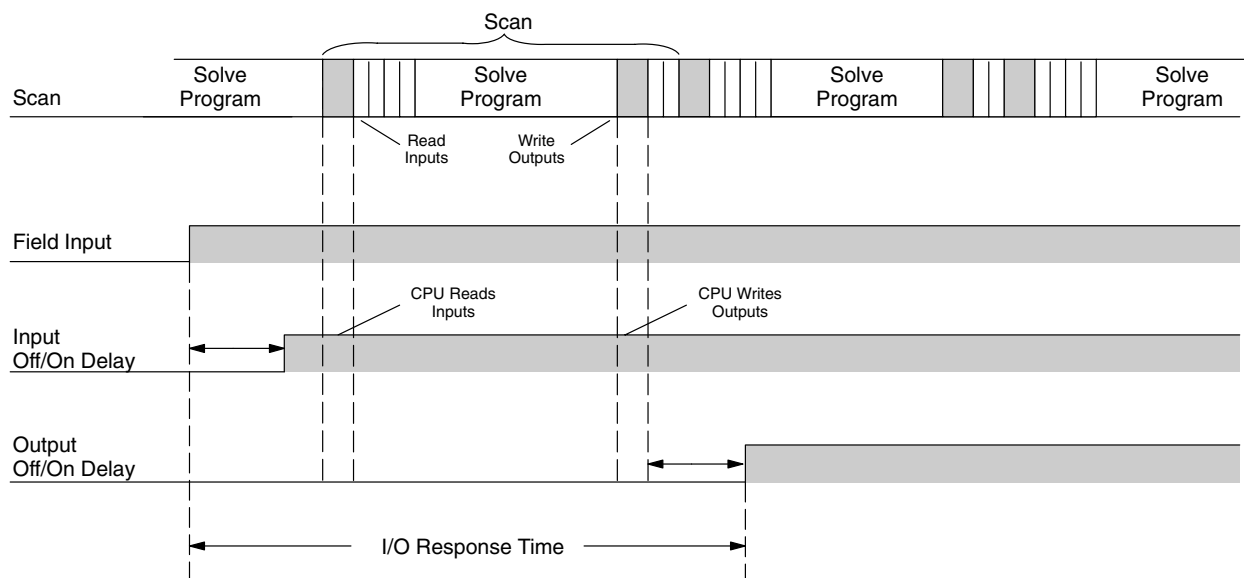
There are four things that can affect the I/O response time.

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

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

**Normal Minimum I/O Response**

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

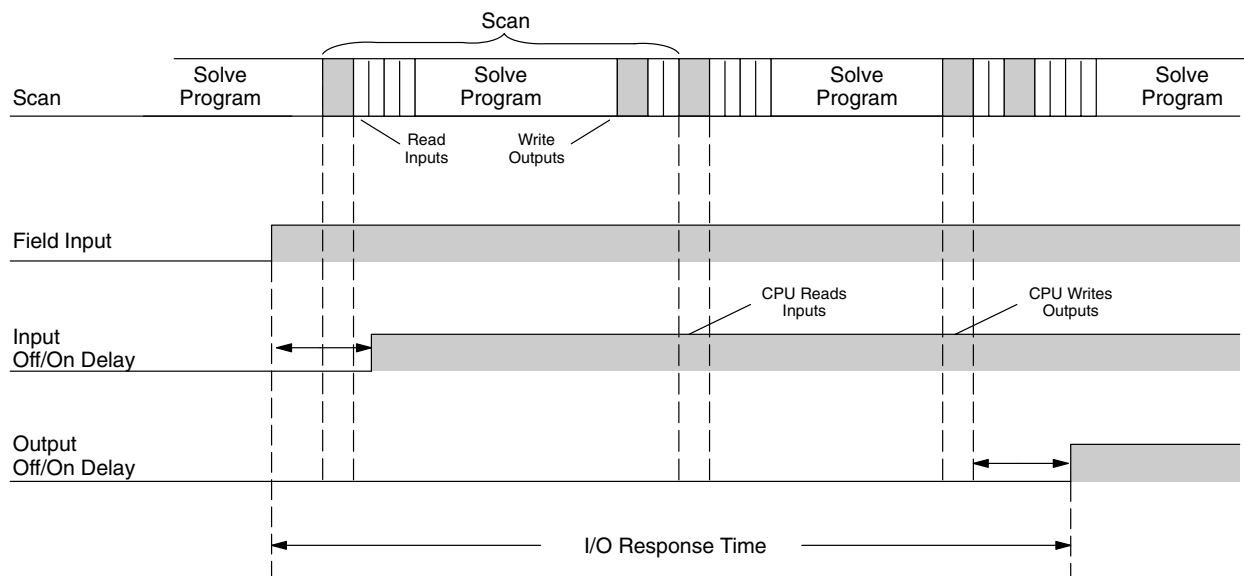


In this case, you can calculate the response time by simply adding the following items:

$$\text{Input Delay} + \text{Scan Time} + \text{Output Delay} = \text{Response Time}$$

### Normal Maximum I/O Response

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



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

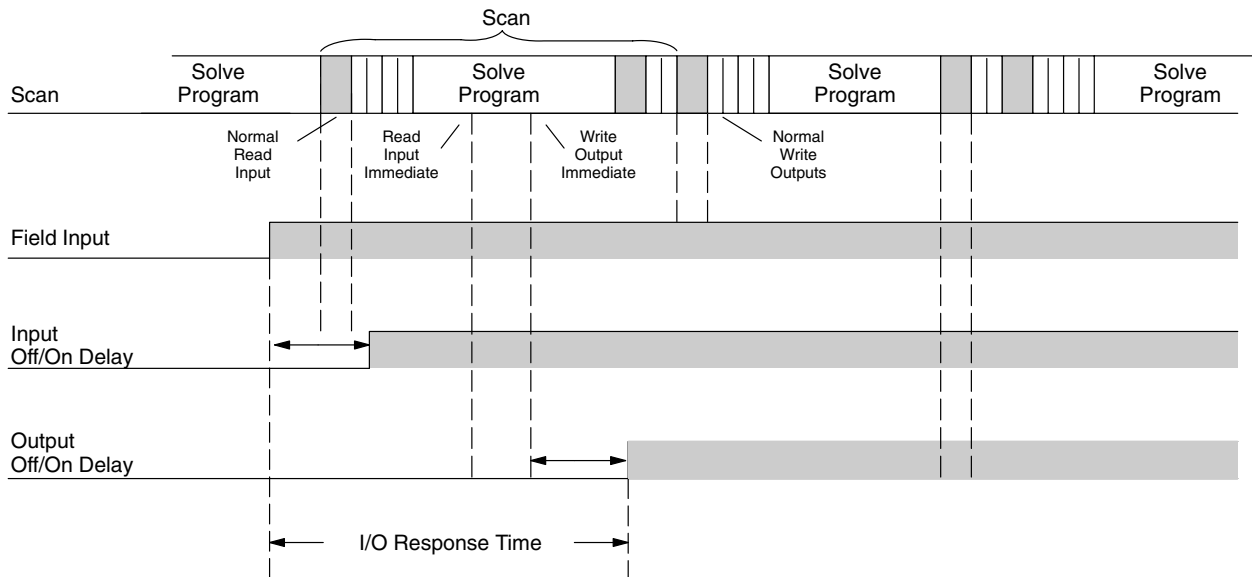
$$\text{Input Delay} + (2 \times \text{Scan Time}) + \text{Output Delay} = \text{Response Time}$$

## Improving Response Time

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

- You can choose instructions with faster execution times
- You can use immediate I/O instructions (which update the I/O points during the program execution)
- You can use the HSIO Mode 50 Pulse Catch features designed to operate in high-speed environments. See the Chapter 3 for details on using this feature.

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



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

$$\text{Input Delay} + \text{Instruction Execution Time} + \text{Output Delay} = \text{Response Time}$$

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



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

## CPU Scan Time Considerations

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

As we've shown previously there are several segments that make up the scan cycle. Each of these segments requires a certain amount of time to complete. Of all the segments, the following are the most important.

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

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

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

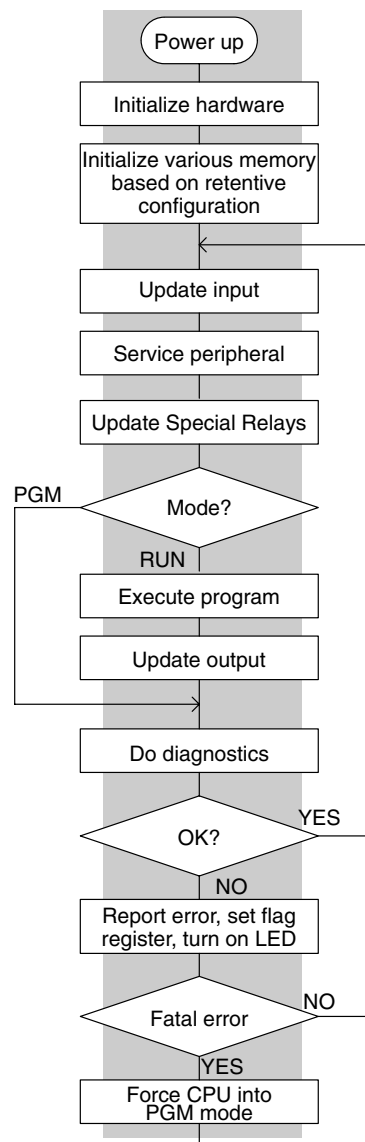
The following paragraphs provide some general information on how much time some of the segments can require.

### Reading Inputs

The time required during each scan to read the input status is 40  $\mu$ S. Don't confuse this with the I/O response time that was discussed earlier.

### Writing Outputs

The time required to write the output status is 629  $\mu$ S. Don't confuse this with the I/O response time that was discussed earlier.



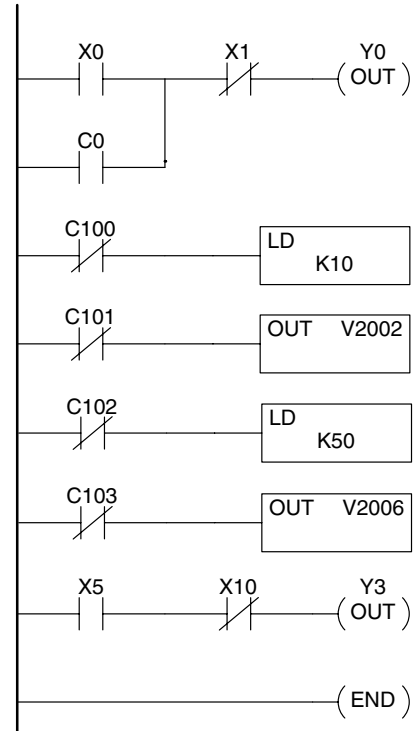
### Application Program Execution

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

Just add the execution times for all the instructions in your program to determine total execution time. Appendix C provides a complete list of the instruction execution times for the DL05 Micro PLC. For example, the execution time for running the program shown below is calculated as follows:

Instruction	Time
STR X0	2 $\mu$ s
OR C0	1.6 $\mu$ s
ANDN X1	1.6 $\mu$ s
OUT Y0	6.8 $\mu$ s
STRN C100	2.3 $\mu$ s
LD K10	42.7 $\mu$ s
STRN C101	2.3 $\mu$ s
OUT V2002	16.6 $\mu$ s
STRN C102	2.3 $\mu$ s
LD K50	42.7 $\mu$ s
STRN C103	2.3 $\mu$ s
OUT V2006	16.6 $\mu$ s
STR X5	2 $\mu$ s
ANDN X10	1.6 $\mu$ s
OUT Y3	6.8 $\mu$ s
END	24 $\mu$ s
<b>SUBTOTAL</b>	<b>174.2 <math>\mu</math>s</b>

Overhead	DL05
Minimum	0.66 mS
Maximum	2.5 ms



$$\text{TOTAL TIME} = (\text{Program execution time} + \text{Overhead}) \times 1.1$$

The program above takes only 174.2  $\mu$ s to execute during each scan. The DL05 spends 0.1ms, on internal timed interrupt management, for every 1ms of instruction time. The total scan time is calculated by adding the program execution time to the overhead (shown above) and multiplying the result (ms) by 1.1. "Overhead" includes all other housekeeping and diagnostic tasks. The scan time will vary slightly from one scan to the next, because of fluctuation in overhead tasks.

**Program Control Instructions** — the DL05 PLCs have an interrupt routine feature that changes the way a program executes. Since this instruction interrupts normal program flow, it will have an effect on the program execution time. For example, a timed interrupt routine with a 10 mS period interrupts the main program execution (before the END statement) every 10 mS, so the CPU can execute the interrupt routine. Chapter 5 provides detailed information on interrupts.

## PLC Numbering Systems

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

octal	1482	BCD	?	binary
?	3A9	?	3	0402 ?
	7	-961428	ASCII	
1001011011			hexadecimal	
	177	?	1011	
decimal	A	72B	?	
-300124				

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

### PLC Resources

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

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

Decimal	1	2	3	4	5	6	7	8
		●	●	●	●	●	●	●
Octal	1	2	3	4	5	6	7	10

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

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

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

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

X=	0	1	2	3	4	5	6	7
X	●	●	●	●	●	●	●	●
1 X	●	●	●	●	●	●	●	●
2 X	●	●	●	●	●	●	●	●

### V-Memory

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

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

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

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

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

### Binary-Coded Decimal Numbers

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

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

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

### Hexadecimal Numbers

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

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

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

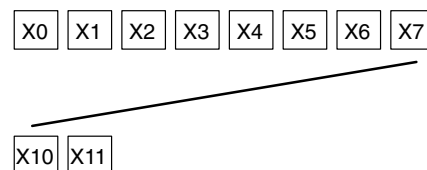
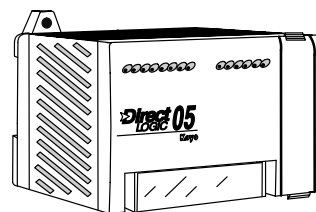
Hexadecimal number	A	7	F	4
V-memory storage	1 0 1 0	0 1 1 1	1 1 1 1	0 1 0 0

## Memory Map

With any PLC system, you generally have many different types of information to process. This includes input device status, output device status, various timing elements, parts counts, etc. It is important to understand how the system represents and stores the various types of data. For example, you need to know how the system identifies input points, output points, data words, etc. The following paragraphs discuss the various memory types used in DL05 Micro PLCs. A memory map overview for the CPU follows the memory descriptions.

### Octal Numbering System

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



### Discrete and Word Locations

As you examine the different memory types, you'll notice two types of memory in the DL05, discrete and word memory. Discrete memory is one bit that can be either a 1 or a 0. Word memory is referred to as V memory (variable) and is a 16-bit location normally used to manipulate data/numbers, store data/numbers, etc.

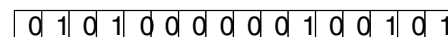
Some information is automatically stored in V memory. For example, the timer current values are stored in V memory.

Discrete – On or Off, 1 bit

X0



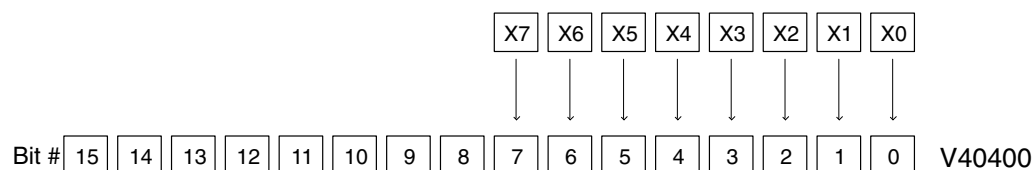
Word Locations – 16 bits



### V Memory Locations for Discrete Memory Areas

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

8 Discrete (X) Input Points

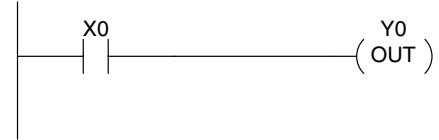


These discrete memory areas and their corresponding V memory ranges are listed in the memory area table for DL05 Micro PLCs on the following pages.

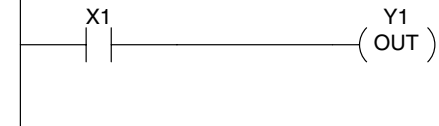


**Input Points  
(X Data Type)**

The discrete input points are noted by an X data type. There are 8 discrete input points and 256 discrete input addresses available with DL05 CPUs. In this example, the output point Y0 will be turned on when input X0 energizes.

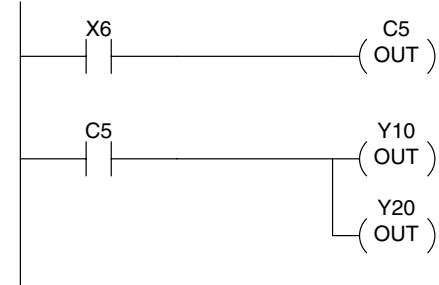
**Output Points  
(Y Data Type)**

The discrete output points are noted by a Y data type. There are 6 discrete outputs and 256 discrete output addresses available with DL05 CPUs. In this example, output point Y1 will be turned on when input X1 energizes.

**Control Relays  
(C Data Type)**

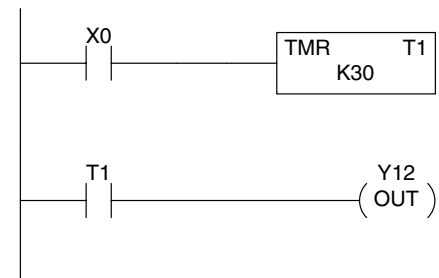
Control relays are discrete bits normally used to control the user program. The control relays do not represent a real world device, that is, they cannot be physically tied to switches, output coils, etc. They are internal to the CPU. Because of this, control relays can be programmed as discrete inputs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

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

**Timers and  
Timer Status Bits  
(T Data type)**

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

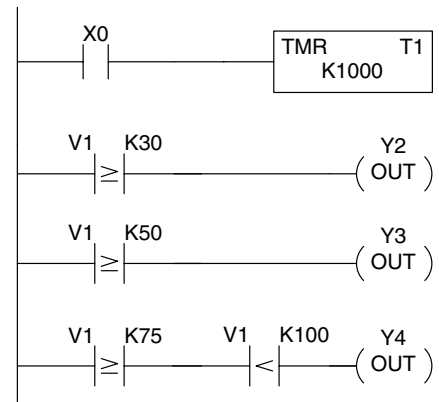
When input X0 turns on, timer T1 will start. When the timer reaches the preset of 3 seconds (K of 30) timer status contact T1 turns on. When T1 turns on, output Y12 turns on. Turning off X0 resets the timer.



### Timer Current Values (V Data Type)

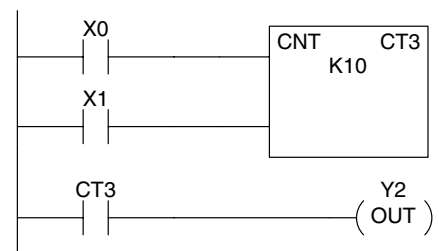
As mentioned earlier, some information is automatically stored in V memory. This is true for the current values associated with timers. For example, V0 holds the current value for Timer 0, V1 holds the current value for Timer 1, etc.

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



### Counters and Counter Status Bits (CT Data type)

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

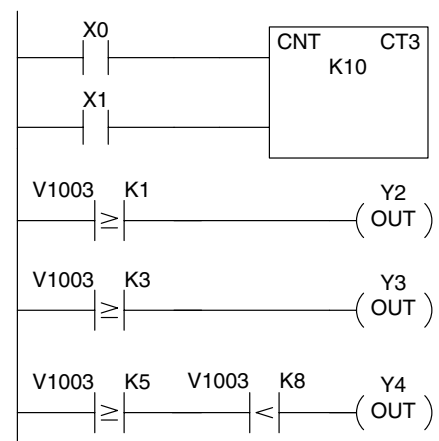


Each time contact X0 transitions from off to on, the counter increments by one. (If X1 comes on, the counter is reset to zero.) When the counter reaches the preset of 10 counts (K of 10) counter status contact CT3 turns on. When CT3 turns on, output Y2 turns on.

### Counter Current Values (V Data Type)

Just like the timers, the counter current values are also automatically stored in V memory. For example, V1000 holds the current value for Counter CT0, V1001 holds the current value for Counter CT1, etc.

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

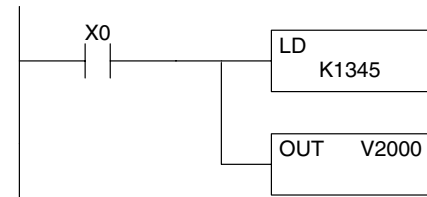


### Word Memory (V Data Type)

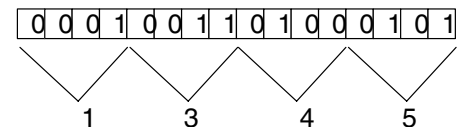
Word memory is referred to as V memory (variable) and is a 16-bit location normally used to manipulate data/numbers, store data/numbers, etc.

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

The example shows how a four-digit BCD constant is loaded into the accumulator and then stored in a V-memory location.



Word Locations – 16 bits

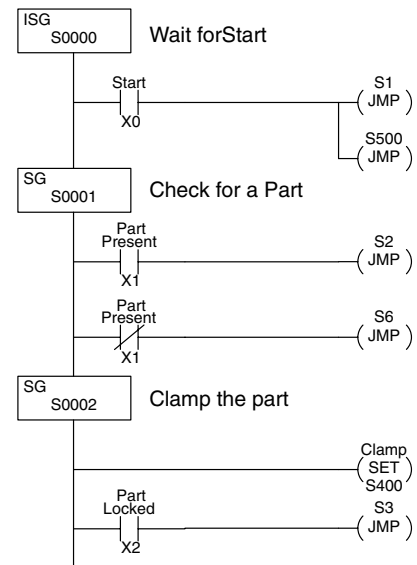


### Stages (S Data type)

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

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

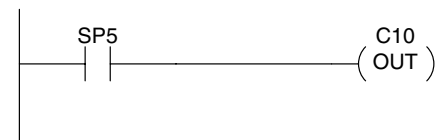
Ladder Representation



### Special Relays (SP Data Type)

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

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



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

## DL05 System V-memory

**System Parameters and Default Data Locations (V Data Type)** The DL05 PLCs reserve several V-memory locations for storing system parameters or certain types of system data. These memory locations store things like the error codes, High-Speed I/O data, and other types of system setup information.

System V-memory	Description of Contents	Default Values / Ranges
V2320–V2377	The default location for multiple preset values for the High-Speed Counter	N/A
V7620–V7627	Locations for DV–1000 operator interface parameters	
V7620	Sets the V-memory location that contains the value.	V0 – V2377
V7621	Sets the V-memory location that contains the message.	V0 – V2377
V7622	Sets the total number (1 – 16) of V-memory locations to be displayed.	1 – 16
V7623	Sets the V-memory location that contains the numbers to be displayed.	V0 – V2377
V7624	Sets the V-memory location that contains the character code to be displayed.	V0 – V2377
V7625	Contains the function number that can be assigned to each key.	V-memory location for X, Y, or C points used.
V7626	Powerup operational mode.	0, 1, 2, 12, 3
V7627	Change preset value.	0000 to 9999
V7630	Starting location for the multi-step presets for channel 1. The default value is 2320, which indicates the first value should be obtained from V2320. Since there are 24 presets available, the default range is V2320 – V2377. You can change the starting point if necessary.	Default: V2320 Range: V0 – V2320
V7631–V7632	Reserved	N/A
V7633	Sets the desired function code for the high speed counter, interrupt, pulse catch, pulse train, and input filter. Location can also be used to set the power-up in Run Mode option.	Default: 0060 Lower Byte Range: Range: 10 – Counter 20 – Quadrature 30 – Pulse Out 40 – Interrupt 50 – Pulse Catch 60 – Filtered discrete In.  Upper Byte Range: Bits 8–12, 14, 15: Unused Bit 13: Power-up in RUN, only if Mode Switch is in TERM position.
V7634	X0 Setup Register for High-Speed I/O functions	Default: 1006
V7635	X1 Setup Register for High-Speed I/O functions	Default: 1006
V7636	X2 Setup Register for High-Speed I/O functions	Default: 1006
V7637–V7646	Reserved	N/A
V7647	Timed Interrupt	Default: 0000 Range: 0003–03E7h (3–9999ms)
V7650–V7654	Reserved	N/A
V7655	Port 2: Setup for the protocol, time-out, and the response delay time.	Default: 00E0
V7656	Port 2: Setup for the station number, baud rate, STOP bit, and parity.	Default: 8501

System V-memory	Description of Contents	Default Values / Ranges
V7657	Port 2: Setup completion code used to notify the completion of the parameter setup.	Default: 0A00
V7660	Scan control setup: Keeps the scan control mode.	Default: 0000
V7661	Setup timer over counter: Counts the times the actual scan time exceeds the user setup time.	N/A
V7662–V7717	Reserved	N/A
V7720–V7722	Locations for DV-1000 operator interface parameters.	N/A
V7720	Titled Timer preset value pointer	N/A
V7721	Title Counter preset value pointer	N/A
V7722	HiByte-Titled Timer preset block size, LoByte-Titled Counter preset block size	N/A
V7723–V7750	Reserved	N/A
V7751	Fault Message Error Code — stores the 4-digit code used with the FAULT instruction when the instruction is executed.	N/A
V7752–V7754	Reserved	N/A
V7755	Error code — stores the fatal error code.	
V7756	Error code — stores the major error code.	
V7757	Error code — stores the minor error code.	
V7760–V7762	Reserved	N/A
V7763	Program address where syntax error exists	
V7764	Syntax error code	N/A
V7765	Scan — stores the total number of scan cycles that have occurred since the last Program Mode to Run Mode transition.	N/A
V7766–V7774	Reserved	N/A
V7775	Scan — stores the current scan time (milliseconds).	N/A
V7776	Scan — stores the minimum scan time that has occurred since the last Program Mode to Run Mode transition (milliseconds).	N/A
V7777	Scan — stores the maximum scan time that has occurred since the last Program Mode to Run Mode transition (milliseconds).	N/A

## DL05 Memory Map

Memory Type	Discrete Memory Reference (octal)	Word Memory Reference (octal)	Qty. Decimal	Symbol
Input Points (See note 1)	X0 – X377	V40400 - V40417	256	X0 
Output Points (See note 1)	Y0 – Y377	V40500 – V40517	256	Y0 
Control Relays	C0 – C777	V40600 - V40637	512	C0      C0 
Special Relays	SP0 – SP777	V41200 – V41237	512	SP0 
Timers	T0 – T177	V41100 – V41107	128	
Timer Current Values	None	V0 – V177	128	V0    K100 
Timer Status Bits	T0 – T177	V41100 – V41107	128	T0 
Counters	CT0 – CT177	V41140 – V41147	128	
Counter Current Values	None	V1000 – V1177	128	V1000    K100 
Counter Status Bits	CT0 – CT177	V41140 – V41147	128	CT0 
Data Words	None	V1200 – V7377	3968	None specific, used with many instructions
Data Words Non-volatile	None	V7400 – V7577	128	None specific, used with many instructions
Stages	S0 – S377	V41000 – V41017	256	S0 
System parameters	None	V7600 – V7777	128	None specific, used for various purposes

1 – The DL05 systems are limited to 8 discrete inputs and 6 discrete outputs with the present available hardware, but 256 point addresses exist.

## X Input Bit Map

This table provides a listing of individual Input points associated with each V-memory address bit for the DL05's eight physical inputs. Actual available references are X0 to X377 (V40400 – V40417).

DL05 Input (X) Points															MSB	LSB	Address
17	16	15	14	13	12	11	10	7	6	5	4	3	2	1	0		
–	–	–	–	–	–	–	–	007	006	005	004	003	002	001	000	V40400	

## Y Output Bit Map

This table provides a listing of individual output points associated with each V-memory address bit for the DL05's six physical outputs. Actual available references are Y0 to Y377 (V40500 – V40517).

DL05 Output (Y) Points															MSB	LSB	Address
17	16	15	14	13	12	11	10	7	6	5	4	3	2	1	0		
–	–	–	–	–	–	–	–	–	–	005	004	003	002	001	000	V40500	

## Stage Control / Status Bit Map

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

DL05 Stage (S) Control Bits															MSB	LSB	Address
17	16	15	14	13	12	11	10	7	6	5	4	3	2	1	0		
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V41000	
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V41001	
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V41002	
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V41003	
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V41004	
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V41005	
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V41006	
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V41007	
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200	V41010	
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220	V41011	
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240	V41012	
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260	V41013	
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300	V41014	
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320	V41015	
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340	V41016	
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360	V41017	

## Control Relay Bit Map

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

MSB		DL05 Control Relays (C)														LSB		Address
17	16	15	14	13	12	11	10	7	6	5	4	3	2	1	0			
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V40600		
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V40601		
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V40602		
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V40603		
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V40604		
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V40605		
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V40606		
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V40607		
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200	V40610		
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220	V40611		
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240	V40612		
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260	V40613		
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300	V40614		
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320	V40615		
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340	V40616		
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360	V40617		
417	416	415	414	413	412	411	410	407	406	405	404	403	402	401	400	V40620		
437	436	435	434	433	432	431	430	427	426	425	424	423	422	421	420	V40621		
457	456	455	454	453	452	451	450	447	446	445	444	443	442	441	440	V40622		
477	476	475	474	473	472	471	470	467	466	465	464	463	462	461	460	V40623		
517	516	515	514	513	512	511	510	507	506	505	504	503	502	501	500	V40624		
537	536	535	534	533	532	531	530	527	526	525	524	523	522	521	520	V40625		
557	556	555	554	553	552	551	550	547	546	545	544	543	542	541	540	V40626		
577	576	575	574	573	572	571	570	567	566	565	564	563	562	561	560	V40627		
617	616	615	614	613	612	611	610	607	606	605	604	603	602	601	600	V40630		
637	636	635	634	633	632	631	630	627	626	625	624	623	622	621	620	V40631		
657	656	655	654	653	652	651	650	647	646	645	644	643	642	641	640	V40632		
677	676	675	674	673	672	671	670	667	666	665	664	663	662	661	660	V40633		
717	716	715	714	713	712	711	710	707	706	705	704	703	702	701	700	V40634		
737	736	735	734	733	732	731	730	727	726	725	724	723	722	721	720	V40635		
757	756	755	754	753	752	751	750	747	746	745	744	743	742	741	740	V40636		
777	776	775	774	773	772	771	770	767	766	765	764	763	762	761	760	V40637		



## Timer Status Bit Map

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

DL05 Timer (T) Contacts															LSB	Address
17	16	15	14	13	12	11	10	7	6	5	4	3	2	1	0	
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V41100
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V41101
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V41102
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V41103
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V41104
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V41105
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V41106
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V41107

## Counter Status Bit Map

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

MSB		DL05 Counter (CT) Contacts														LSB		Address
17	16	15	14	13	12	11	10	7	6	5	4	3	2	1	0			
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V41140		
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V41141		
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V41142		
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V41143		
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V41144		
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V41145		
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V41146		
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V41147		

## Network Configuration and Connections

### Configuring the DL05's Comm Ports

This section describes how to configure the CPU's built-in networking ports for either MODBUS or **DirectNET**. This will allow you to connect the DL05 PLC system directly to MODBUS networks using the RTU protocol, or to other devices on a **DirectNET** network. MODBUS host systems must be capable of issuing the MODBUS commands to read or write the appropriate data. For details on the MODBUS protocol, check with your MODBUS supplier for the latest version of the Gould MODBUS Protocol reference Guide. For more details on **DirectNET**, order our **DirectNET** manual, part number DA-DNET-M.

#### Communication Port 1

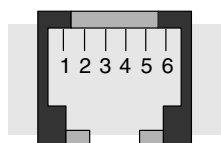
Com 1 Connects to HPP, **DirectSOFT**, operator interfaces, etc.  
 6-pin, RS232C  
 9600 Baud (Fixed)  
 Parity - odd (default)  
 Station address 1 (fixed)  
 8 data bits  
 1 start, 1 stop bit  
 Asynchronous, Half-duplex, DTE  
 Protocol: (Auto-Select)  
     K sequence (Slave only)  
     DirectNET (Slave only)  
     MODBUS (Slave only)

#### Communication Port 2

Com 2 Connects to HPP, **DirectSOFT**, operator interfaces, etc.  
 6-pin, RS232C  
 Communication speed (baud)  
     300, 600, 1200, 2400, 4800,  
     9600, 19200, 38400  
 Parity - odd (default), even, none  
 Station address 1 (default)  
 8 data bits  
 1 start, 1 stop bit  
 Asynchronous, Half-duplex, DTE  
 Protocol: (Auto-Select)  
     K sequence (Slave only)  
     DirectNET (Master/Slave)  
     MODBUS (Master/Slave)  
     Non-sequence/Print

You will need to make sure the network connection is a 3-wire RS-232 type. Normally, the RS-232 signals are used for communications between two devices with distances up to a maximum of 15 meters.

### Networking DL05 to DL05 RS-232C



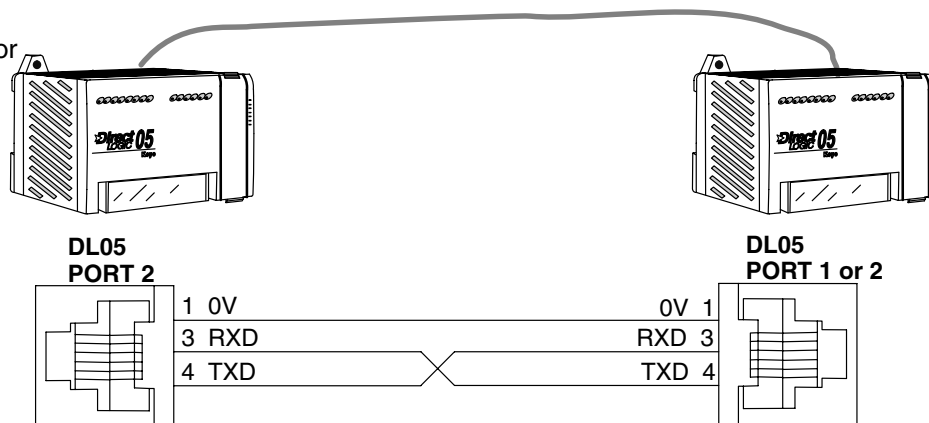
6-pin Female Modular Connector

#### Port 1 Pin Descriptions

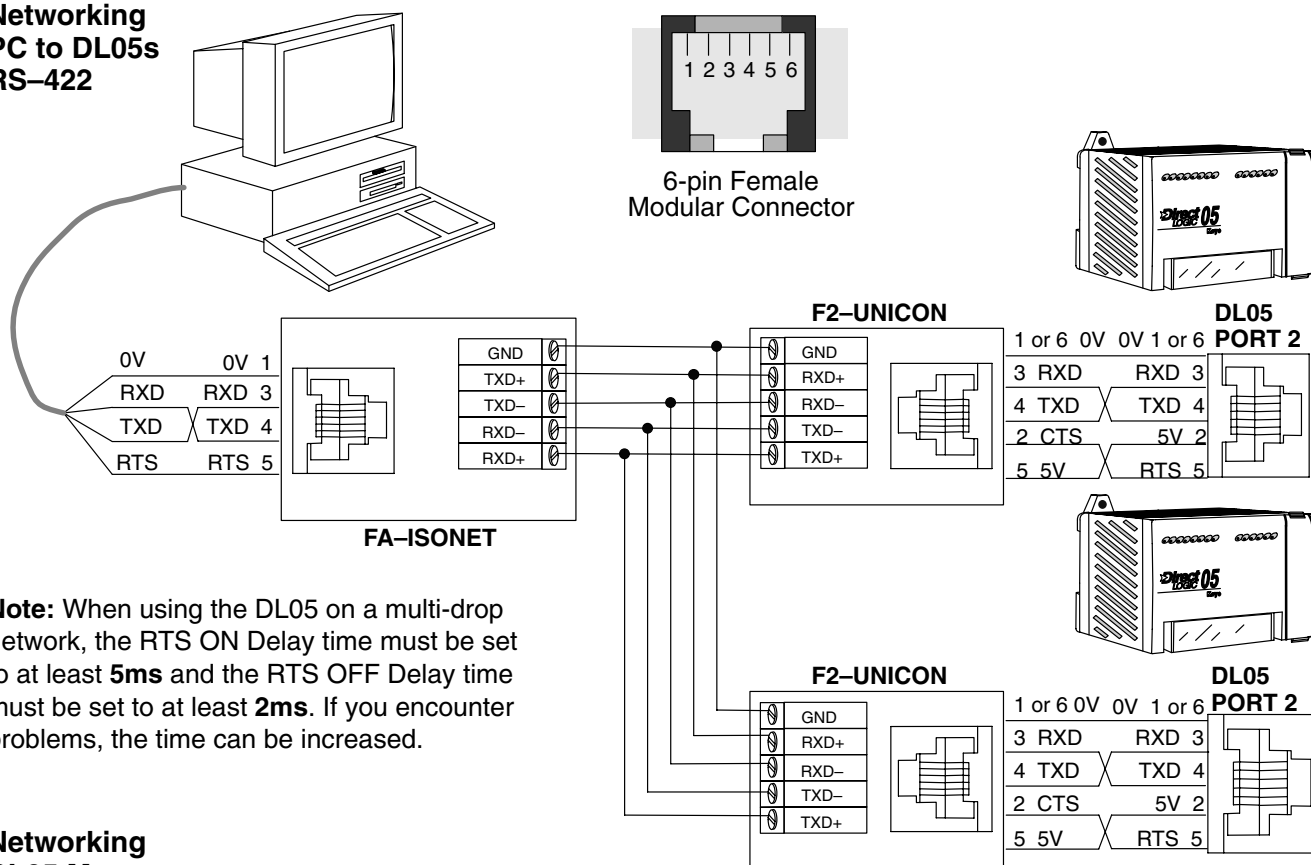
1	0V	Power (-) connection (GND)
2	5V	Power (+) connection
3	RXD	Receive Data (RS232C)
4	TXD	Transmit Data (RS232C)
5	5V	Power (+) connection
6	0V	Power (-) connection (GND)

#### Port 2 Pin Descriptions

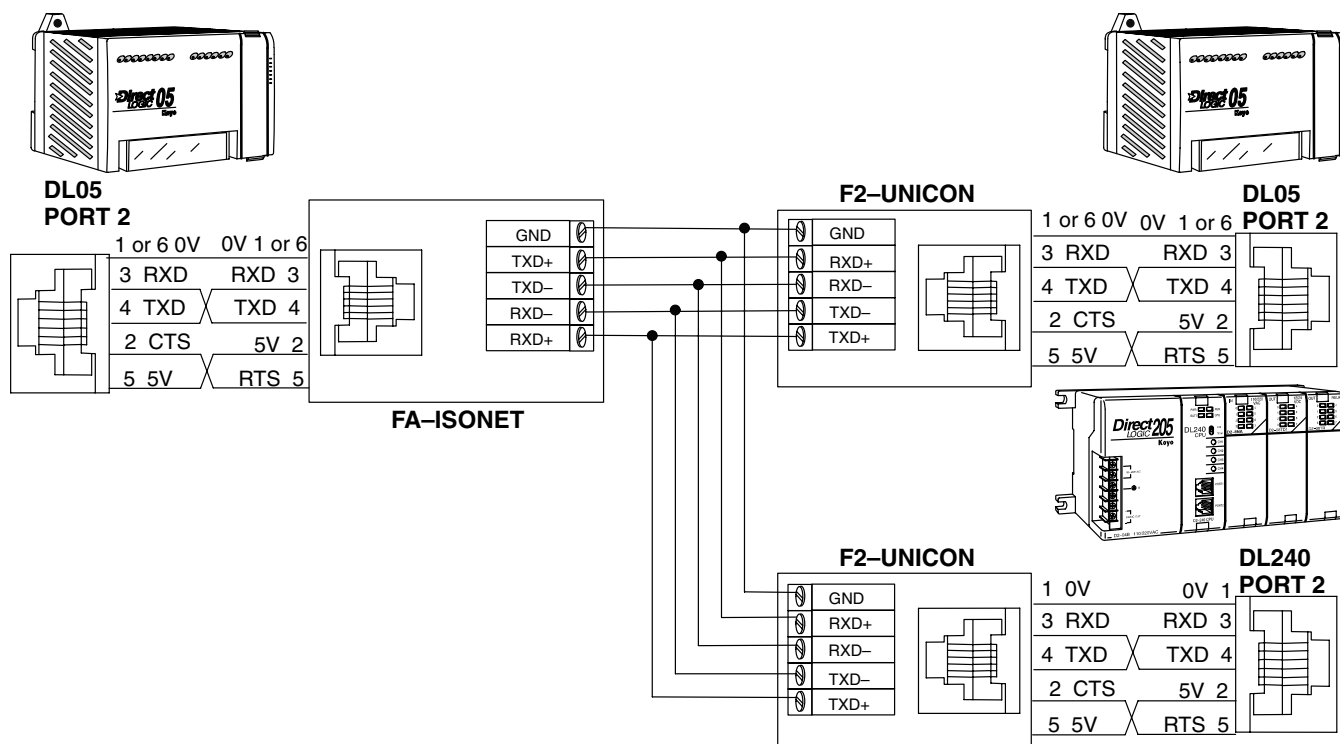
1	0V	Power (-) connection (GND)
2	5V	Power (+) connection
3	RXD	Receive Data (RS232C)
4	TXD	Transmit Data (RS232C)
5	RTS	Request to Send
6	0V	Power (-) connection (GND)



### Networking PC to DL05s RS-422



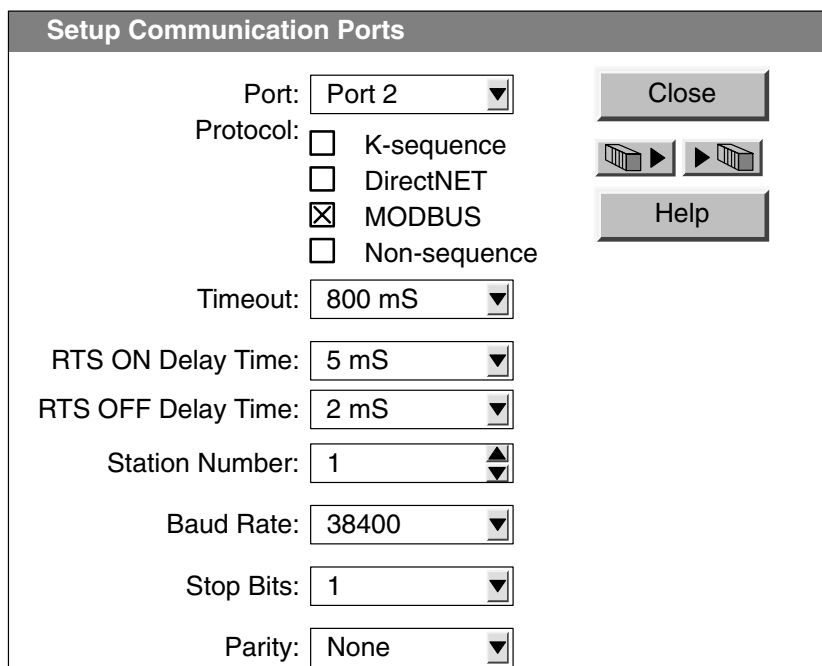
### Networking DL05 Master to Other PLCs



## MODBUS Port Configuration

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

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



The dialog box titled "Setup Communication Ports" contains the following settings:

- Port:** Port 2 (dropdown menu)
- Protocol:**
  - ☐ K-sequence
  - ☐ DirectNET
  - ☒ MODBUS
  - ☐ Non-sequence
- Timeout:** 800 mS (dropdown menu)
- RTS ON Delay Time:** 5 mS (dropdown menu)
- RTS OFF Delay Time:** 2 mS (dropdown menu)
- Station Number:** 1 (spin box)
- Baud Rate:** 38400 (dropdown menu)
- Stop Bits:** 1 (dropdown menu)
- Parity:** None (dropdown menu)

Buttons on the right: Close, a button with a right-pointing arrow and a box icon, and Help.

- **Timeout:** amount of time the port will wait after it sends a message to get a response before logging an error.
- **RTS ON / OFF Delay Time:** The RTS ON Delay Time specifies the time the DL05 waits to send the data after it has raised the RTS signal line. The RTS OFF Delay Time specifies the time the DL05 waits to release the RTS signal line after the data has been sent. *When using the DL05 on a multi-drop network, the RTS ON Delay time must be set to at least 5ms and the RTS OFF Delay time must be set to at least 2ms. If you encounter problems, the time can be increased.*
- **Station Number:** For making the CPU port a MODBUS® master, choose “1”. The possible range for MODBUS slave numbers is from 1 to 247, but the DL05 network instructions used in Master mode will access only slaves 1 to 99. Each slave must have a unique number. At powerup, the port is automatically a slave, unless and until the DL05 executes ladder logic network instructions which use the port as a master. Thereafter, the port reverts back to slave mode until ladder logic uses the port again.
- **Baud Rate:** The available baud rates include 300, 600, 1200, 2400, 4800, 9600, 19200, and 38400 baud. Choose a higher baud rate initially, reverting to lower baud rates if you experience data errors or noise problems on the network. Important: You must configure the baud rates of all devices on the network to the same value. Refer to the appropriate product manual for details.
- **Stop Bits:** Choose 1 or 2 stop bits for use in the protocol.
- **Parity:** Choose none, even, or odd parity for error checking.



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

### DirectNET Port Configuration

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

- **Port:** From the port number list box, choose “Port 2”.
- **Protocol:** Click the check box to the left of “DirectNET” (use AUX 56 on the HPP, then select “DNET”), and then you’ll see the dialog box below.

- **Timeout:** amount of time the port will wait after it sends a message to get a response before logging an error.
- **RTS ON / OFF Delay Time:** The RTS ON Delay Time specifies the time the DL05 waits to send the data after it has raised the RTS signal line. The RTS OFF Delay Time specifies the time the DL05 waits to release the RTS signal line after the data has been sent. *When using the DL05 on a multi-drop network, the RTS ON Delay time must be set to at least 5ms and the RTS OFF Delay time must be set to at least 2ms. If you encounter problems, the time can be increased.*
- **Station Number:** For making the CPU port a **DirectNET** master, choose “1”. The allowable range for **DirectNET** slaves is from 1 to 90 (each slave must have a unique number). At powerup, the port is automatically a slave, unless and until the DL05 executes ladder logic instructions which attempt to use the port as a master. Thereafter, the port reverts back to slave mode until ladder logic uses the port again.
- **Baud Rate:** The available baud rates include 300, 600, 1200, 2400, 4800, 9600, 19200, and 38400 baud. Choose a higher baud rate initially, reverting to lower baud rates if you experience data errors or noise problems on the network. Important: You must configure the baud rates of all devices on the network to the same value.
- **Stop Bits:** Choose 1 or 2 stop bits for use in the protocol.
- **Parity:** Choose none, even, or odd parity for error checking.
- **Format:** Choose between hex or ASCII formats.



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

## Network Slave Operation

This section describes how other devices on a network can communicate with a CPU port that you have configured as a **DirectNET** slave or MODBUS slave (DL05). A MODBUS host must use the MODBUS RTU protocol to communicate with the DL05 as a slave. The host software must send a MODBUS function code and MODBUS address to specify a PLC memory location the DL05 comprehends. The **DirectNET** host uses normal I/O addresses to access applicable DL05 CPU and system. No CPU ladder logic is required to support either MODBUS slave or **DirectNET** slave operation.

### MODBUS Function Codes Supported

The MODBUS function code determines whether the access is a read or a write, and whether to access a single data point or a group of them. The DL05 supports the MODBUS function codes described below.

MODBUS Function Code	Function	DL05 Data Types Available
01	Read a group of coils	Y, CR, T, CT
02	Read a group of inputs	X, SP
05	Set / Reset a single coil	Y, CR, T, CT
15	Set / Reset a group of coils	Y, CR, T, CT
03, 04	Read a value from one or more registers	V
06	Write a value into a single register	V
16	Write a value into a group of registers	V

### Determining the MODBUS Address

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

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

**If Your Host Software Requires the Data Type and Address...** Many host software packages allow you to specify the MODBUS data type and the MODBUS address that corresponds to the PLC memory location. This is the easiest method, but not all packages allow you to do it this way.

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

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

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

DL05 Memory Type	QTY (Dec.)	PLC Range (Octal)	MODBUS Address Range (Decimal)	MODBUS Data Type
<b>For Discrete Data Types .... Convert PLC Addr. to Dec. + Start of Range + Data Type</b>				
Inputs (X)	256	X0 – X377	2048 – 2303	Input
Special Relays (SP)	512	SP0 – SP777	3072 – 3583	Input
Outputs (Y)	256	Y0 – Y377	2048 – 2303	Coil
Control Relays (CR)	512	C0 – C777	3072 – 4583	Coil
Timer Contacts (T)	128	T0 – T177	6144 – 6271	Coil
Counter Contacts (CT)	128	CT0 – CT177	6400 – 6527	Coil
Stage Status Bits (S)	256	S0 – S377	5120 – 5375	Coil
<b>For Word Data Types .... Convert PLC Addr. to Dec. + Data Type</b>				
Timer Current Values (V)	128	V0 – V177	0 – 127	Input Register
Counter Current Values (V)	128	V1000 – V1177	512 – 639	Input Register
V Memory, user data (V)	3968	V1200 – V7377	640 – 3839	Holding Register
V Memory, non-volatile (V)	128	V7600 – V7777	3968 – 4095	Holding Register

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

**Example 1: V2100**

Find the MODBUS address for User V location V2100.

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

**PLC Address (Dec.) + Data Type**

V2100 = 1088 decimal

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

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

**Example 2: Y20**

Find the MODBUS address for output Y20.

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

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

Y20 = 16 decimal

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

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

**Example 3: T10 Current Value**

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

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

**PLC Address (Dec.) + Data Type**

T10 = 8 decimal

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

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

**Example 4: C54**

Find the MODBUS address for Control Relay C54.

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

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

C54 = 44 decimal

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

Control Relays (CR)	512	C0 – C777	3072 – 3583	Coil
---------------------	-----	-----------	-------------	------



### If Your MODBUS Host Software Requires an Address ONLY

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

- 484 Mode
- 584/984 Mode

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

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

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

DL05 Memory Type	QTY (Dec.)	PLC Range (Octal)	MODBUS Address Range (Decimal)	484 Mode Address	584/984 Mode Address	MODBUS Data Type
<b>For Discrete Data Types ... Convert PLC Addr. to Dec. + Start of Range + Appropriate Mode Address</b>						
Inputs (X)	256	X0 – X377	2048 – 2303	1001	10001	Input
Special Relays (SP)	512	SP0 – SP777	3072 – 3583	1001	10001	Input
Outputs (Y)	256	Y0 – Y377	2048 – 2303	1	1	Coil
Control Relays (CR)	512	C0 – C777	3072 – 3583	1	1	Coil
Timer Contacts (T)	128	T0 – T177	6144 – 6271	1	1	Coil
Counter Contacts (CT)	128	CT0 – CT177	6400 – 6527	1	1	Coil
Stage Status Bits (S)	256	S0 – S377	5120 – 5375	1	1	Coil
<b>For Word Data Types .... Convert PLC Addr. to Dec. + Appropriate Mode Address</b>						
Timer Current Values (V)	128	V0 – V377	0 – 127	3001	30001	Input Reg.
Counter Current Values (V)	128	V1000 – V1177	512 – 639	3001	30001	Input Reg.
V Memory, user data (V)	3200	V1400 – V7377	768 – 3839	4001	40001	Hold Reg.
V Memory, non-volatile (V)	128	V7400 – V7577	3840 – 3967	4001	40001	Hold Reg.
V Memory, system (V)	256	V7600 – V7777	3968 – 4095	4001	40001	Hold Reg.

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

**Example 1: V2100  
584/984 Mode**

Find the MODBUS address for User V location V2100.

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

**PLC Address (Dec.) + Mode Address**

$$\begin{aligned} \text{V2100} &= 1088 \text{ decimal} \\ 1088 + 40001 &= \boxed{41089} \end{aligned}$$

V Memory, system (V)	128	V1200 – V7377	3480 – 3735	4001	40001	Hold Reg.
----------------------	-----	---------------	-------------	------	-------	-----------

**Example 2: Y20  
584/984 Mode**

Find the MODBUS address for output Y20.

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

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

$$\begin{aligned} \text{Y20} &= 16 \text{ decimal} \\ 16 + 2048 + 1 &= \boxed{2065} \end{aligned}$$

Outputs (Y)	256	Y0 – Y377	2048 – 2303	1	1	Coil
-------------	-----	-----------	-------------	---	---	------

**Example 3: T10 Current Value  
484 Mode**

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

1. Find Timer Current Values in the table.
2. Convert T10 into decimal (8).
3. Add the MODBUS starting address for the mode (3001).

**PLC Address (Dec.) + Mode Address**

$$\begin{aligned} \text{T10} &= 8 \text{ decimal} \\ 8 + 3001 &= \boxed{3009} \end{aligned}$$

Timer Current Values (V)	128	V0 – V177	0 – 127	3001	30001	Input Reg.
--------------------------	-----	-----------	---------	------	-------	------------

**Example 4: C54  
584/984 Mode**

Find the MODBUS address for Control Relay C54.

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

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

$$\begin{aligned} \text{C54} &= 44 \text{ decimal} \\ 44 + 3072 + 1 &= \boxed{3117} \end{aligned}$$

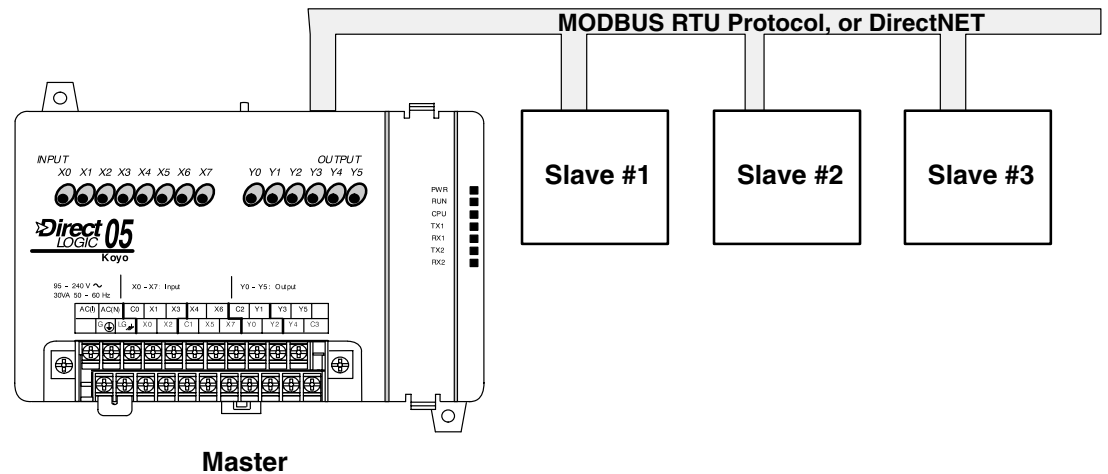
Control Relays (CR)	512	C0 – C777	3072 – 3583	1	1	Coil
---------------------	-----	-----------	-------------	---	---	------

**Determining the  
DirectNET Address**

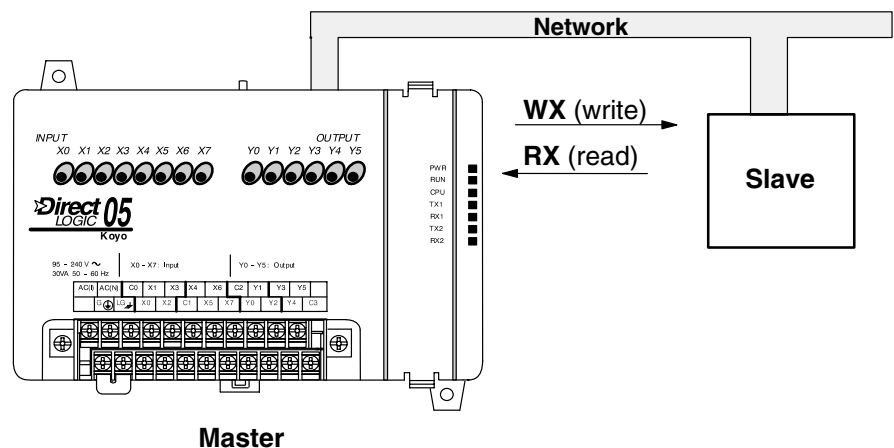
Addressing the memory types for **DirectNET** slaves is very easy. Use the ordinary native address of the slave device itself. To access a slave PLC's memory address V2000 via **DirectNET**, for example, the network master will request V2000 from the slave.

## Network Master Operation

This section describes how the DL05 can communicate on a MODBUS or **DirectNET** network as a master. For MODBUS networks, it uses the MODBUS RTU protocol, which must be interpreted by all the slaves on the network. Both MODBUS and **DirectNet** are single master/multiple slave networks. The master is the only member of the network that can initiate requests on the network. This section teaches you how to design the required ladder logic for network master operation.



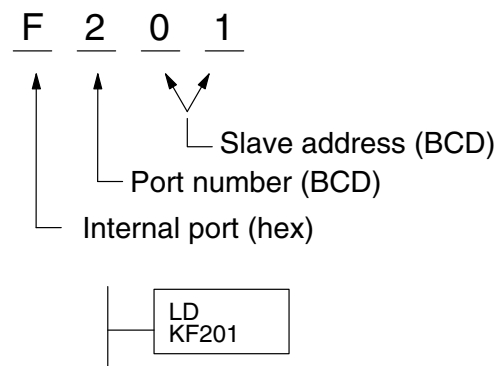
When using the DL05 PLC as the master station, simple RLL instructions are used to initiate the requests. The WX instruction initiates network write operations, and the RX instruction initiates network read operations. Before executing either the WX or RX commands, we will need to load data related to the read or write operation onto the CPU's accumulator stack. When the WX or RX instruction executes, it uses the information on the stack combined with data in the instruction box to completely define the task, which goes to the port.



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

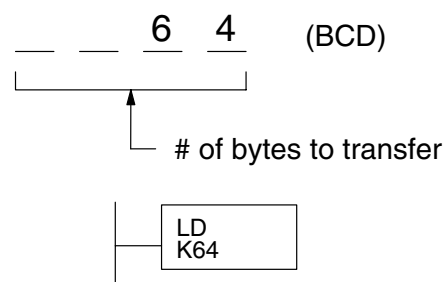
**Step 1:  
Identify Master  
Port # and Slave #**

The first Load (LD) instruction identifies the communications port number on the network master (DL05) and the address of the slave station. This instruction can address up to 99 MODBUS slaves, or 90 **Direct**NET slaves. The format of the word is shown to the right. The “F2” in the upper byte indicates the use of the right port of the DL05 PLC, port number 2. The lower byte contains the slave address number in BCD (01 to 99).



**Step 2:  
Load Number of  
Bytes to Transfer**

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



The number of bytes specified also depends on the type of data you want to obtain. For example, the DL05 Input points can be accessed by V-memory locations or as X input locations. However, if you only want X0 – X27, you’ll have to use the X input data type because the V-memory locations can only be accessed in 2-byte increments. The following table shows the byte ranges for the various types of **Direct**LOGIC™ products.

DL 05 / 205 / 350 / 405 Memory	Bits per unit	Bytes
V memory	16	2
T / C current value	16	2
Inputs (X, SP)	8	1
Outputs (Y, C, Stage, T/C bits)	8	1
Scratch Pad Memory	8	1
Diagnostic Status	8	1

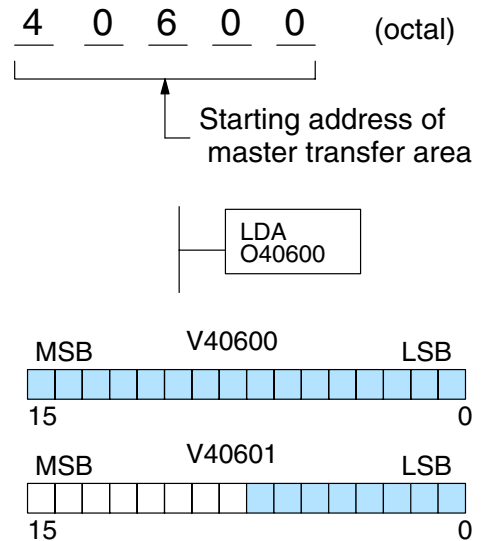
DL330 / 340 Memory	Bits per unit	Bytes
Data registers	8	1
T / C accumulator	16	2
I/O, internal relays, shift register bits, T/C bits, stage bits	1	1
Scratch Pad Memory	8	1
Diagnostic Status(5 word R/W)	16	10

**Step 3:  
Specify Master  
Memory Area**

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

For a WX instruction, the DL05 CPU sends the number of bytes previously specified from its memory area beginning at the LDA address specified.

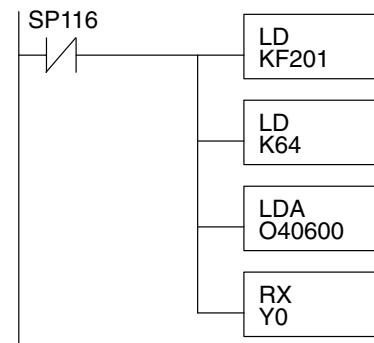
For an RX instruction, the DL05 CPU reads the number of bytes previously specified from the slave, placing the received data into its memory area beginning at the LDA address specified.



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

**Step 4:  
Specify Slave  
Memory Area**

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



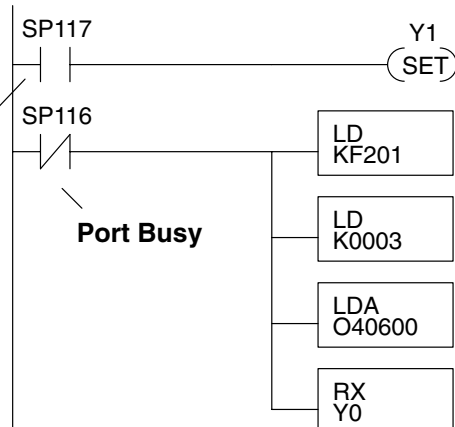
- **DirectNET** slaves – specify the same address in the WX and RX instruction as the slave's native I/O address
- MODBUS DL405, DL205, or DL05 slaves – specify the same address in the WX and RX instruction as the slave's native I/O address
- MODBUS 305 slaves – use the following table to convert DL305 addresses to MODBUS addresses

DL305 Series CPU Memory Type-to-MODBUS Cross Reference (excluding 350 CPU)					
PLC Memory type	PLC base address	MODBUS base addr.	PLC Memory Type	PLC base address	MODBUS base addr.
TMR/CNT Current Values	R600	V0	TMR/CNT Status Bits	CT600	GY600
I/O Points	IO 000	GY0	Control Relays	CR160	GY160
Data Registers	R401, R400	V100	Shift Registers	SR400	GY400
Stage Status Bits (D3-330P only)	S0	GY200			

### Communications from a Ladder Program

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

#### Port Communication Error



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

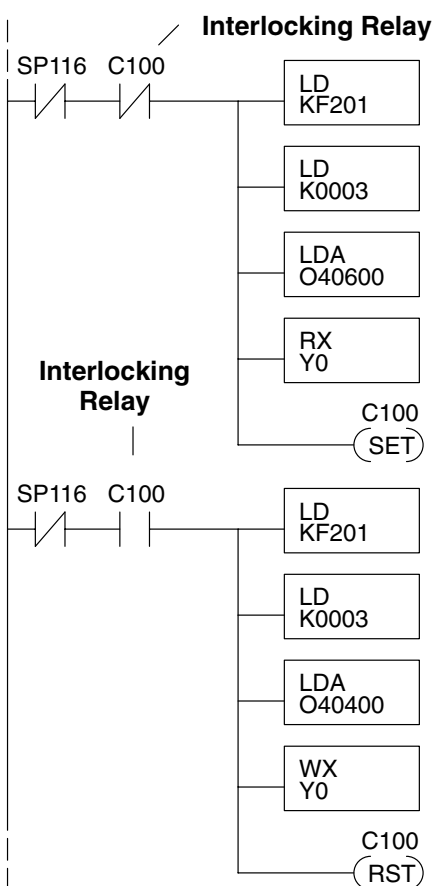
The “Port Communication Error” bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an RX or WX instruction is executed.

### Multiple Read and Write Interlocks

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

In the example to the right, after the RX instruction is executed, C0 is set. When the port has finished the communication task, the second routine is executed and C0 is reset.

If you're using RLL<sup>PLUS</sup> Stage Programming, you can put each routine in a separate program stage to ensure proper execution and switch from stage to stage allowing only one of them to be active at a time.



# Standard RLL Instructions

---

In This Chapter. . . .

- Boolean Instructions
  - Comparative Boolean
  - Immediate Instructions
  - Timer, Counter and Shift Register Instructions
  - Accumulator / Stack Load and Output Data Instructions
  - Logical Instructions (Accumulator)
  - Math Instructions
  - Bit Operation Instructions (Accumulator)
  - Number Conversion Instructions (Accumulator)
  - Table Instructions
  - CPU Control Instructions
  - Program Control Instructions
  - Interrupt Instructions
  - Message Instructions
-

## Introduction

DL05 Micro PLCs offer a wide variety of instructions to perform many different types of operations. This chapter shows you how to use each standard Relay Ladder Logic (RLL) instruction. In addition to these instructions, you may also need to refer to the Drum instruction in Chapter 6, or the Stage programming instructions in Chapter 7.

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

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

Instruction	Page
ACON	5-107
ADDB	5-73
ADD	5-63
ADDD	5-64
AND	5-11, 5-55
AND STR	5-12
ANDD	5-56
ANDE	5-22
ANDI	5-27
ANDN	5-11, 5-25
ANDND	5-17
ANDNE	5-22
ANDNI	5-27
ANDPD	5-17
ATH	5-85
BCD	5-83
BIN	5-82
CMP	5-61
CMPD	5-62
CNT	5-36
CV	7-23
CVJMP	7-23
DEC	5-71
DECB	5-72
DECO	5-81
DISI	5-104
DIV	5-69
DIVB	5-76
DIVD	5-70
DLBL	5-107
DRUM	6-12
EDRUM	6-2, NO TAG

Instruction	Page
ENCO	5-80
END	5-94
ENI	5-103
FAULT	5-106
FOR	5-96
GRAY	5-88
GTS	5-98
HTA	5-86
INC	5-71
INCB	5-72
INT	5-103
INV	5-84
IRT	5-103
IRTC	5-103
ISG	7-22
JMP	7-22
LD	5-48
LDA	5-51
LDD	5-49
LDF	5-50
LDLBL	5-92
MLR	5-101
MLS	5-101
MOV	5-91
MOVMC	5-92
MUL	5-67
MULB	5-75
MULD	5-68
NCON	5-107
NEXT	5-96
NJMP	7-22
NOP	5-94
NOT	5-14

Instruction	Page
OR	5-10, 5-57
OR OUT	5-13
ORE	5-21
ORI	5-26
ORN	5-10, 5-24
ORND	5-16
ORNE	5-21
OR OUTI	5-28
OR STR	5-12
ORD	5-58
ORNI	5-26
OROUTI	5-28
ORPD	5-16
OUT	5-13, 5-52
OUTD	5-52
OUTF	5-53
OUTI	5-28
PAUSE	5-19
PD	5-14
POP	5-53
PRINT	5-109
RST	5-18
RSTI	5-29
RSTWT	5-95
RT	5-98
RTC	5-98
RX	5-113
SBR	5-98
SET	5-18
SETI	5-29
SFLDGT	5-89
SG	7-21
SGCNT	5-38



Instruction	Page
SHFL	5-77
SHFR	5-79
SR	5-42
STOP	5-94
STR	5-9, 5-23
STRE	5-20
STRI	5-26
STRN	5-9, 5-23

Instruction	Page
STRND	5-15
STRNE	5-20
STRNI	5-26
STRPD	5-15
SUB	5-65
SUBB	5-74
SUBD	5-66
SUM	5-77

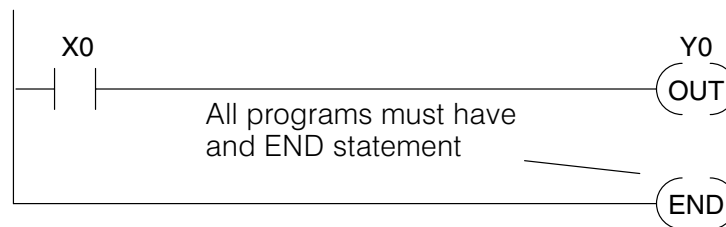
Instruction	Page
TMR	5-31
TMRA	5-33
TMRAF	5-33
TMRF	5-31
UDC	5-40
WX	5-115
XOR	5-59
XORD	5-60

## Using Boolean Instructions

Do you ever wonder why so many PLC manufacturers always quote the scan time for a 1K boolean program? Simple. Most all programs utilize many boolean instructions. These are typically very simple instructions designed to join input and output contacts in various series and parallel combinations. Since the **DirectSOFT™** package allows you to use graphic symbols to build the program, you don't absolutely *have* to know the mnemonics of the instructions. However, it may be helpful at some point, especially if you ever have to troubleshoot the program with a Handheld Programmer. The following paragraphs show how these instructions are used to build simple ladder programs.

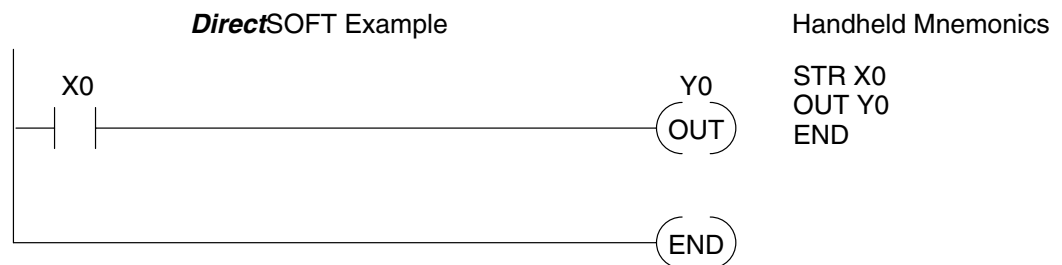
### END Statement

All DL05 programs require an END statement as the last instruction. This tells the CPU that this is the end of the program. Normally, any instructions placed after the END statement will not be executed. There are exceptions to this such as interrupt routines, etc. Chapter 5 discusses the instruction set in detail.



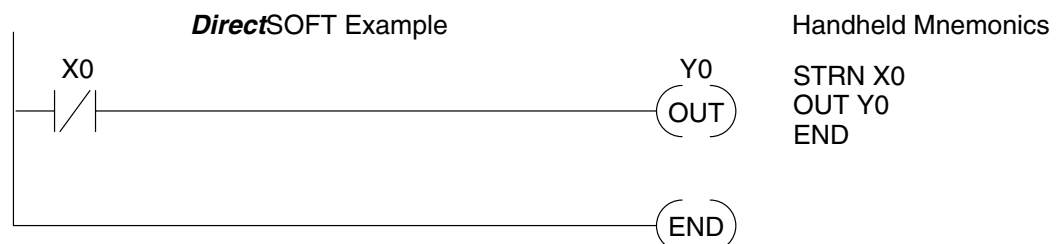
### Simple Rungs

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

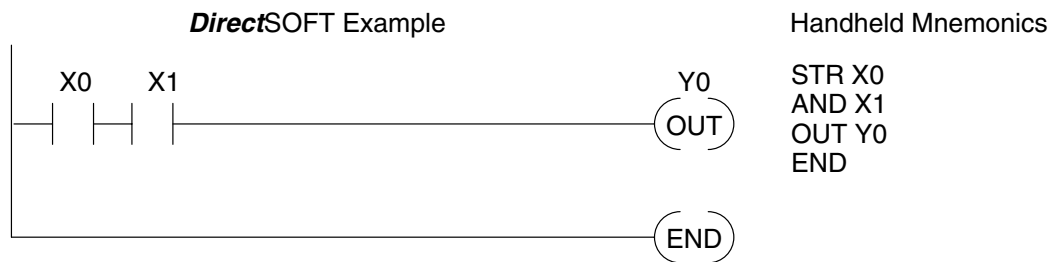


### Normally Closed Contact

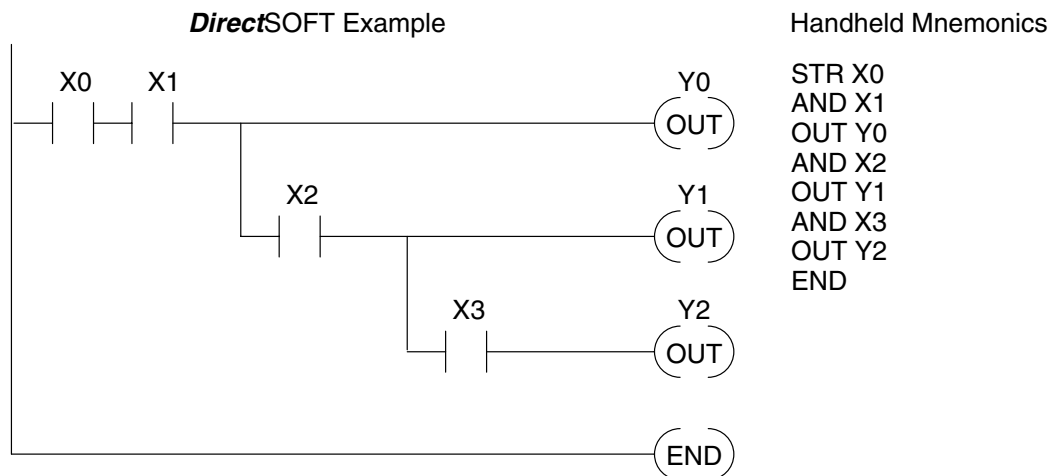
Normally closed contacts are also very common. This is accomplished with the Store Not or, STRN instruction. The following example shows a simple rung with a normally closed contact.



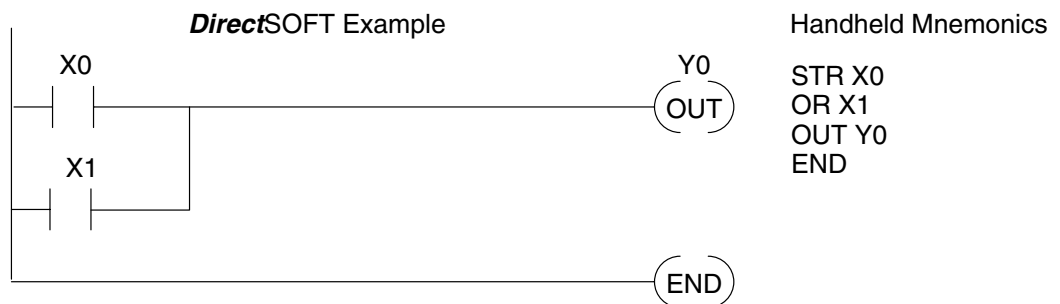
**Contacts in Series** Use the AND instruction to join two or more contacts in series. The following example shows two contacts in series and a single output coil. The instructions used would be STR X0, AND X1, followed by OUT Y0.



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

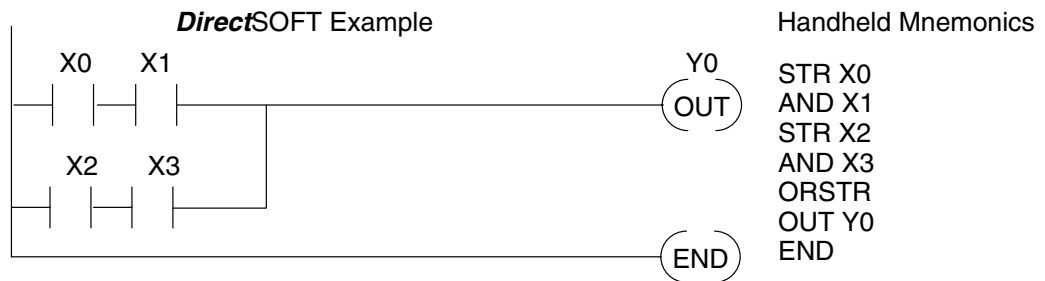


**Parallel Elements** You also have to join contacts in parallel. The OR instruction allows you to do this. The following example shows two contacts in parallel and a single output coil. The instructions would be STR X0, OR X1, followed by OUT Y0.

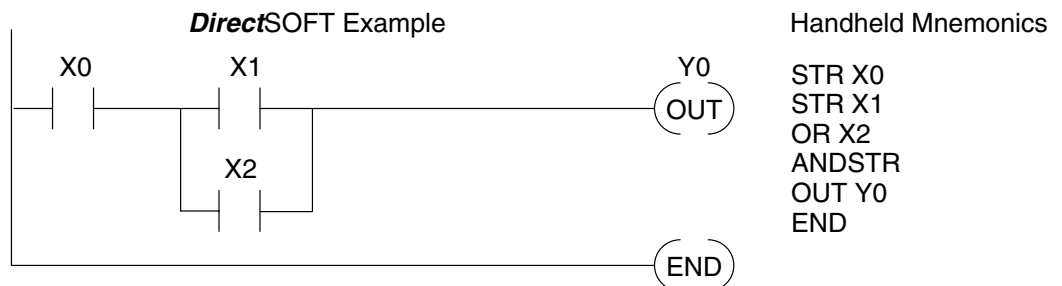


**Joining Series Branches in Parallel**

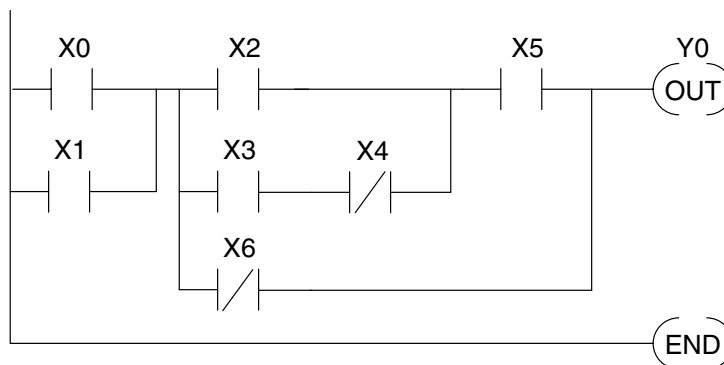
Quite often it is necessary to join several groups of series elements in parallel. The Or Store (ORSTR) instruction allows this operation. The following example shows a simple network consisting of series elements joined in parallel.

**Joining Parallel Branches in Series**

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

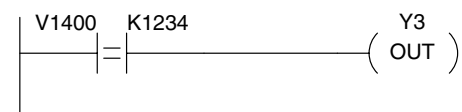
**Combination Networks**

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

**Comparative Boolean**

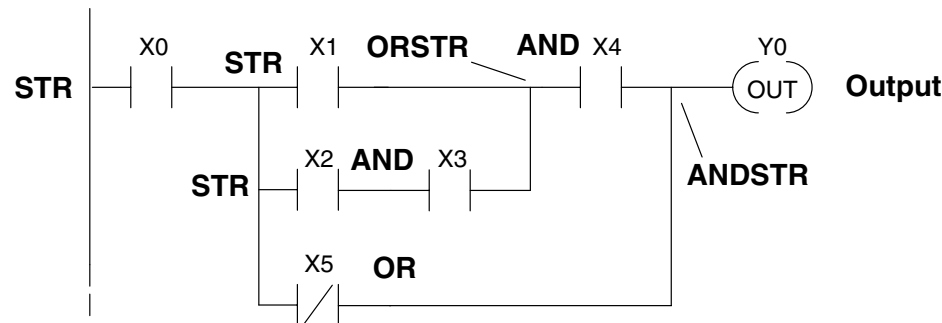
Some PLC manufacturers make it really difficult to do a simple comparison of two numbers. Some of them require you to move the data all over the place before you can actually perform the comparison. The DL05 Micro PLCs provide Comparative Boolean instructions that allow you to quickly and easily solve this problem. The Comparative Boolean provides evaluation of two 4-digit values using boolean contacts. The valid evaluations are: equal to, not equal to, equal to or greater than, and less than.

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



Boolean Stack

There are limits to how many elements you can include in a rung. This is because the DL05 PLCs use an 8-level boolean stack to evaluate the various logic elements. The boolean stack is a temporary storage area that solves the logic for the rung. Each time the program encounters a STR instruction, the instruction is placed on the top of the stack. Any other STR instructions already on the boolean stack are pushed down a level. The ANDSTR, and ORSTR instructions combine levels of the boolean stack when they are encountered. An error will occur during program compilation if the CPU encounters a rung that uses more than the eight levels of the boolean stack. The following example shows how the boolean stack is used to solve boolean logic.



**STR X0**

1	STR X0
2	
3	
4	
5	
6	
7	
8	

**STR X1**

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

**STR X2**

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

**AND X3**

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

**ORSTR**

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

**AND X4**

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

**ORNOT X5**

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

:

8	
---	--

:

8	
---	--

:

8	
---	--

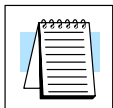
**ANDSTR**

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

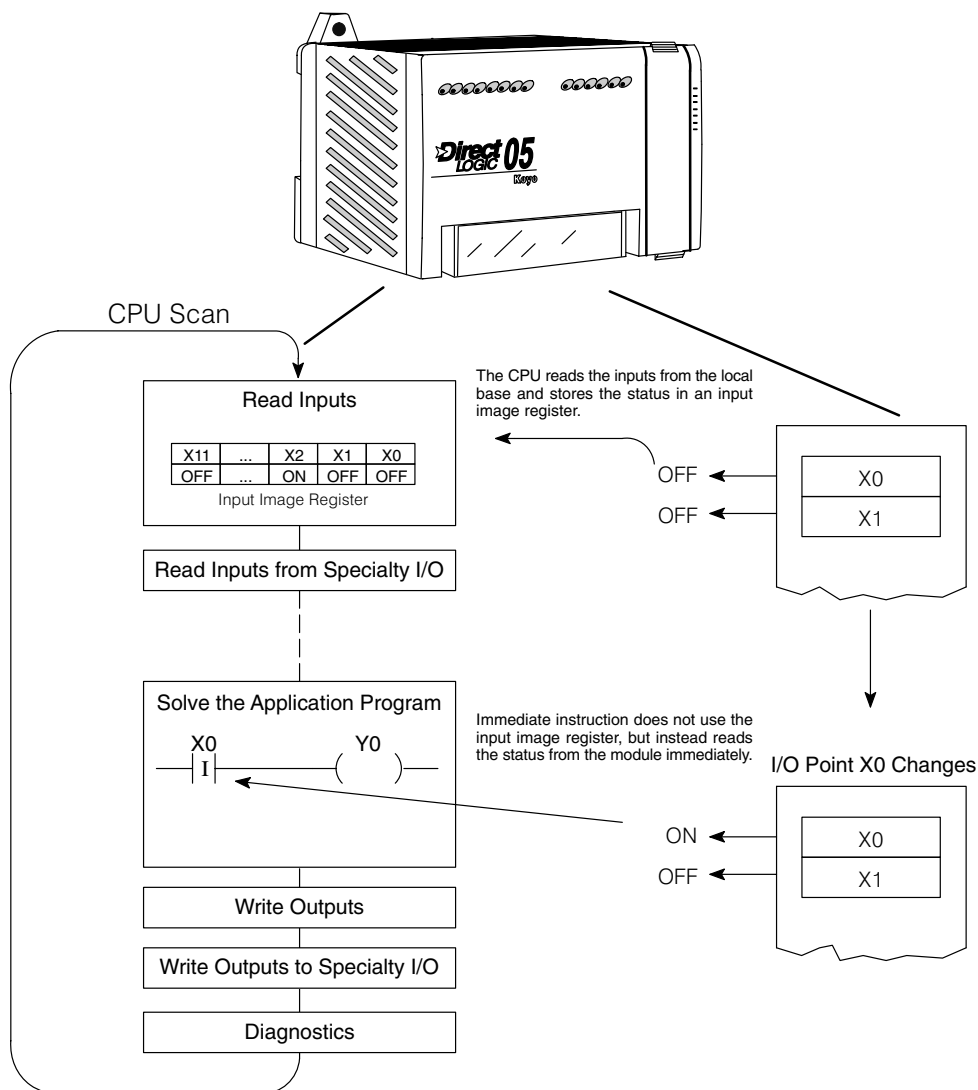
:

8	
---	--

**Immediate Boolean** The DL05 Micro PLCs can usually complete an operation cycle in a matter of milliseconds. However, in some applications you may not be able to wait a few milliseconds until the next I/O update occurs. The DL05 PLCs offer Immediate input and outputs which are special boolean instructions that allow reading directly from inputs and writing directly to outputs during the program execution portion of the CPU cycle. You may recall that this is normally done during the input or output update portion of the CPU cycle. The immediate instructions take longer to execute because the program execution is interrupted while the CPU reads or writes the I/O point. This function is not normally done until the read inputs or the write outputs portion of the CPU cycle.



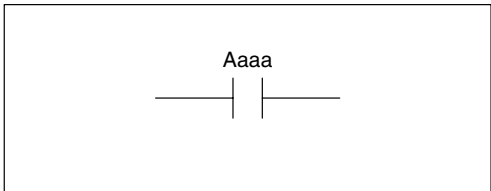
**NOTE:** Even though the immediate input instruction reads the most current status from the input point, it only uses the results to solve that one instruction. It does not use the new status to update the image register. Therefore, any regular instructions that follow will still use the image register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



Boolean Instructions

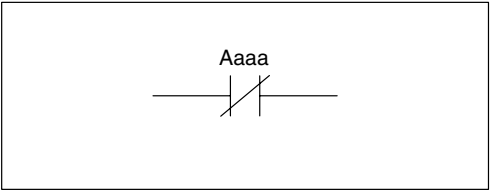
Store (STR)

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



Store Not (STRN)

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



Operand Data Type		DL05 Range
	A	aaa
Inputs	X	0-377
Outputs	Y	0-377
Control Relays	C	0-777
Stage	S	0-377
Timer	T	0-177
Counter	CT	0-177
Special Relay	SP	0-777

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

DirectSOFT



Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT
GX OUT	→	C 2	ENT

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

DirectSOFT

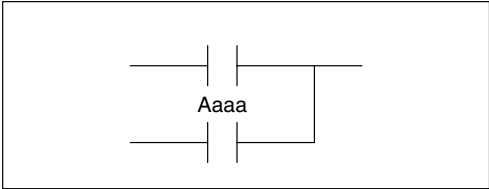


Handheld Programmer Keystrokes

SP STRN	→	B 1	ENT
GX OUT	→	C 2	ENT

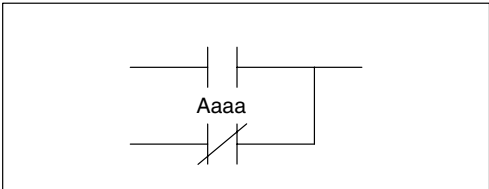
Or  
(OR)

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



Or Not  
(ORN)

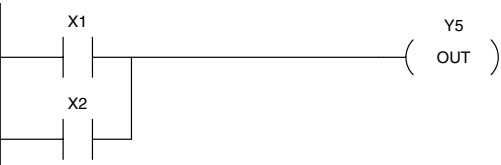
The Or Not instruction logically ors a normally closed contact in parallel with another contact in a rung. The status of the contact will be opposite the state of the associated image register point or memory location.



Operand Data Type		DL05 Range
A		aaa
Inputs	X	0-377
Outputs	Y	0-377
Control Relays	C	0-777
Stage	S	0-377
Timer	T	0-177
Counter	CT	0-177
Special Relay	SP	0-777

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

DirectSOFT



Handheld Programmer Keystrokes

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

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

DirectSOFT



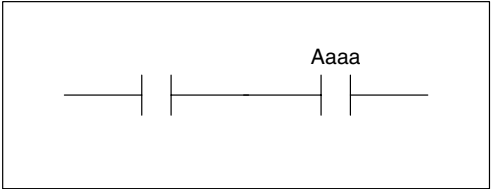
Handheld Programmer Keystrokes

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



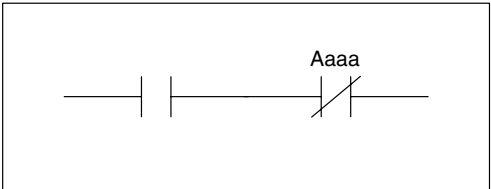
And  
(AND)

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



And Not  
(ANDN)

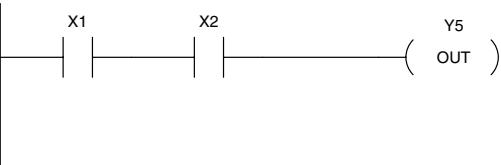
The And Not instruction logically ands a normally closed contact in series with another contact in a rung. The status of the contact will be opposite the state of the associated image register point or memory location.



Operand Data Type		DL05 Range
	A	aaa
Inputs	X	0-377
Outputs	Y	0-377
Control Relays	C	0-777
Stage	S	0-377
Timer	T	0-177
Counter	CT	0-177
Special Relay	SP	0-777

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

DirectSOFT

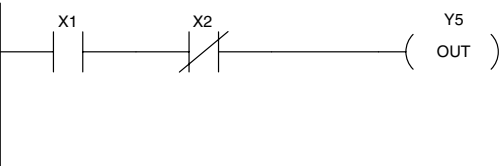


Handheld Programmer Keystrokes

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

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

DirectSOFT

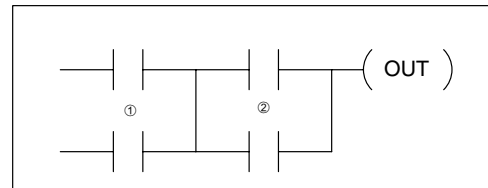


Handheld Programmer Keystrokes

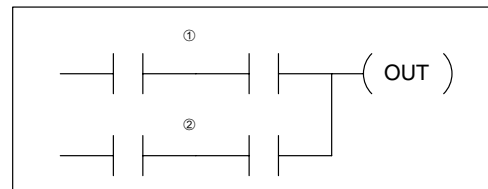
\$ STR	→	B 1	ENT
W ANDN	→	C 2	ENT
GX OUT	→	F 5	ENT

**And Store  
(AND STR)**

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

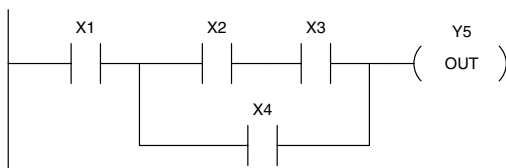
**Or Store  
(OR STR)**

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



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

DirectSOFT

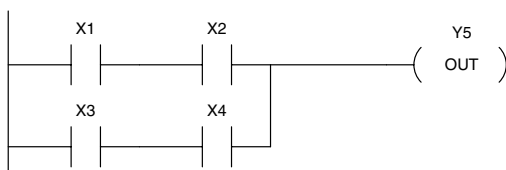


Handheld Programmer Keystrokes

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

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

DirectSOFT



Handheld Programmer Keystrokes

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

## Out (OUT)

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

—( Aaaa  
OUT )

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

Operand Data Type		DL05 Range
	<b>A</b>	<b>aaa</b>
Inputs	X	0-377
Outputs	Y	0-377
Control Relays	C	0-777

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

DirectSOFT



Handheld Programmer Keystrokes

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

## Or Out (OR OUT)

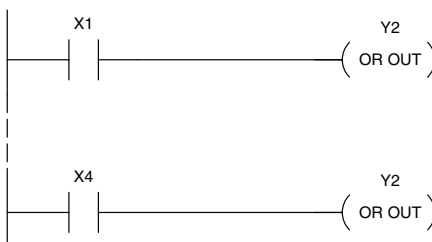
The Or Out instruction allows more than one rung of discrete logic to control a single output. Multiple Or Out instructions referencing the same output coil may be used, since *all* contacts controlling the output are logically ORED together. If the status of *any* rung is on, the output will also be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

—( A aaa  
OR OUT )

Operand Data Type		DL05 Range
	<b>A</b>	<b>aaa</b>
Inputs	X	0-177
Outputs	Y	0-177
Control Relays	C	0-777

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

DirectSOFT

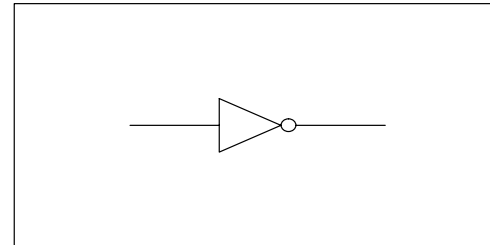


Handheld Programmer Keystrokes

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

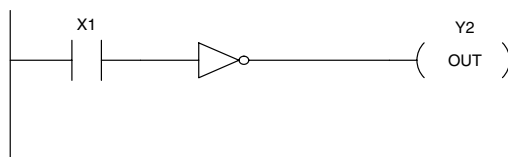
**Not  
(NOT)**

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

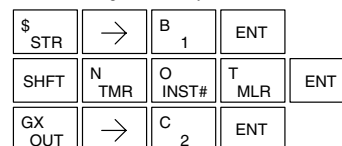


In the following example when X1 is off, Y2 will energize. This is because the Not instruction inverts the status of the rung at the point of the instruction.

DirectSOFT



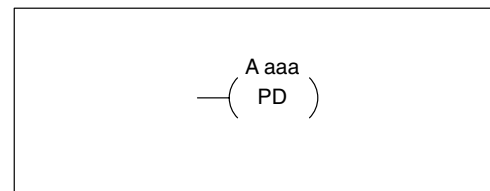
Handheld Programmer Keystrokes



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

**Positive  
Differential  
(PD)**

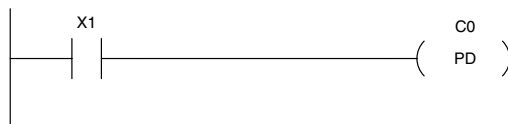
The Positive Differential instruction is typically known as a one shot. When the input logic produces an off to on transition, the output will energize for one CPU scan.



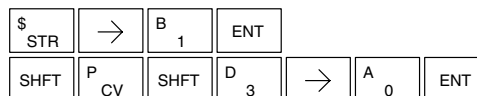
Operand Data Type		DL05 Range
	A	aaa
Inputs	X	0-377
Outputs	Y	0-377
Control Relays	C	0-777

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

DirectSOFT

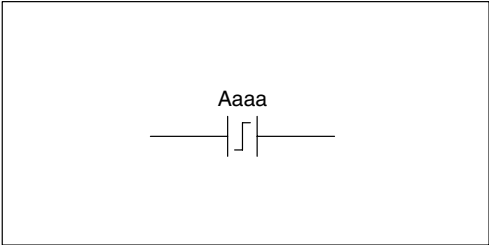


Handheld Programmer Keystrokes



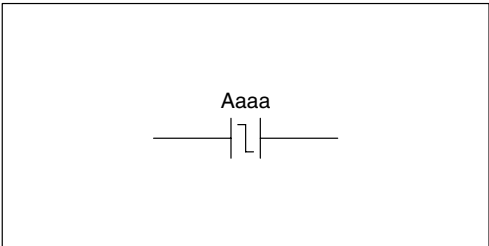
Store Positive  
Differential  
(STRPD)

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



Store Negative  
Differential  
(STRND)

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



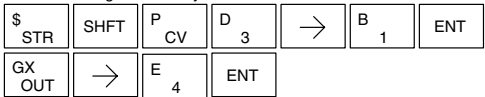
Operand Data Type		DL05 Range
A		aaa
Inputs	X	0-377
Outputs	Y	0-377
Control Relays	C	0-777
Stage	S	0-377
Timer	T	0-177
Counter	CT	0-177

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

DirectSOFT

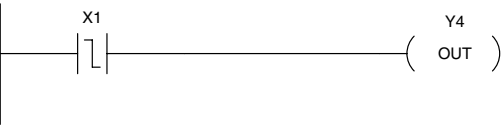


Handheld Programmer Keystrokes

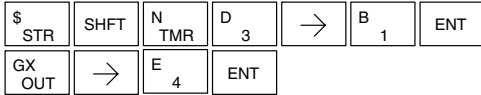


In the following example, each time X1 is makes an On-to-Off transition, Y4 will energize for one scan.

DirectSOFT

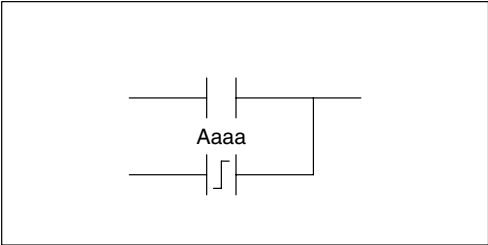


Handheld Programmer Keystrokes



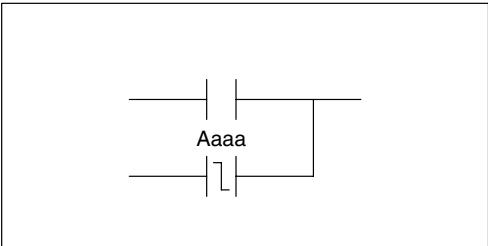
Or Positive Differential (ORPD)

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



Or Negative Differential (ORND)

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



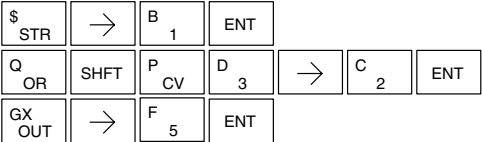
Operand Data Type	DL05 Range	
	<b>A</b>	<b>aaa</b>
Inputs	X	0-377
Outputs	Y	0-377
Control Relays	C	0-777
Stage	S	0-377
Timer	T	0-177
Counter	CT	0-177

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

DirectSOFT



Handheld Programmer Keystrokes

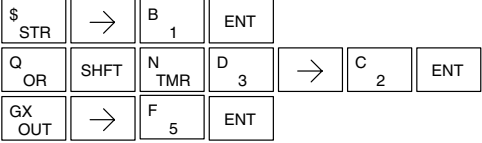


In the following example, Y 5 will energize whenever X1 is on, or for one CPU scan when X2 transitions from On to Off.

DirectSOFT

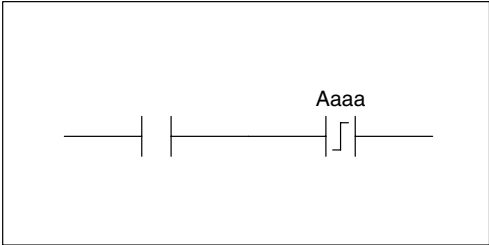


Handheld Programmer Keystrokes



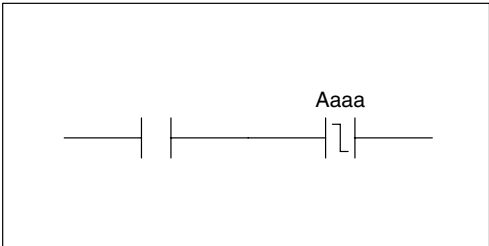
And Positive  
Differential  
(ANDPD)

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



And Negative  
Differential  
(ANDND)

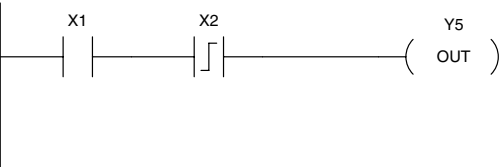
The And Negative Differential instruction logically ands a normally open contact in parallel with another contact in a rung. The status of the contact will be open until the associated image register point makes an On-to-Off transition, closing it for one CPU scan. Thereafter, it remains open until another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



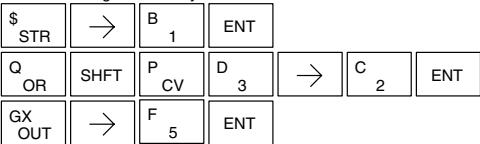
Operand Data Type	DL05 Range	
	A	aaa
Inputs	X	0-377
Outputs	Y	0-377
Control Relays	C	0-777
Stage	S	0-377
Timer	T	0-177
Counter	CT	0-177

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

DirectSOFT

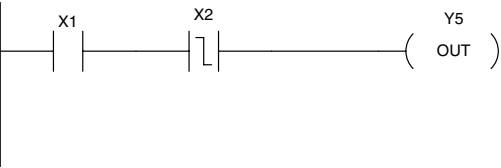


Handheld Programmer Keystrokes

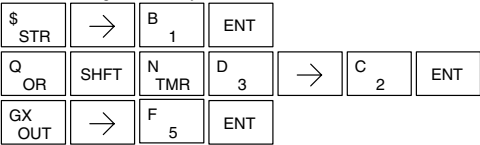


In the following example, Y5 will energize for one CPU scan whenever X1 is on and X2 transitions from On to Off.

DirectSOFT



Handheld Programmer Keystrokes



**Set  
(SET)**

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

Optional  
memory range

A aaa      aaa

— ( SET )

**Reset  
(RST)**

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

Optional  
memory range

A aaa      aaa

— ( RST )

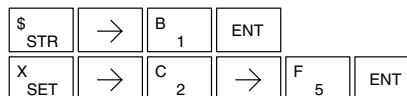
Operand Data Type		DL05 Range
	<b>A</b>	<b>aaa</b>
Inputs	X	0–377
Outputs	Y	0–377
Control Relays	C	0–777
Stage	S	0–377
Timer	T	0–177
Counter	CT	0–177

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

DirectSOFT



Handheld Programmer Keystrokes

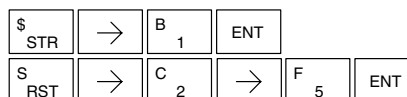


In the following example when X1 is on, Y2 through Y5 will be reset or de-energized.

DirectSOFT



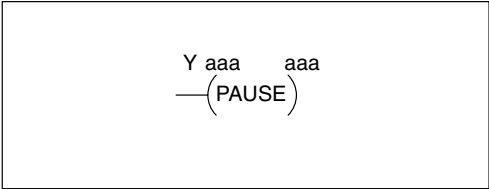
Handheld Programmer Keystrokes





Pause  
(PAUSE)

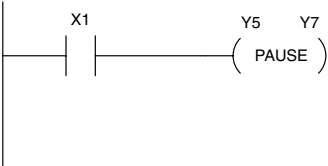
The Pause instruction disables the output update on a range of outputs. The ladder program will continue to run and update the image register. However, the outputs in the range specified in the Pause instruction will be turned off at the output points.



Operand Data Type	DL05 Range
	aaa
Outputs Y	0-377

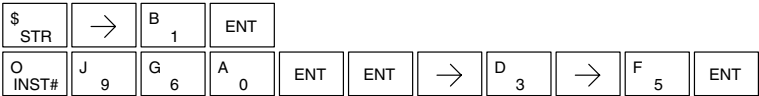
In the following example, when X1 is ON, Y3–Y5 will be turned OFF. The execution of the ladder program will not be affected.

DirectSOFT



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

Handheld Programmer Keystrokes

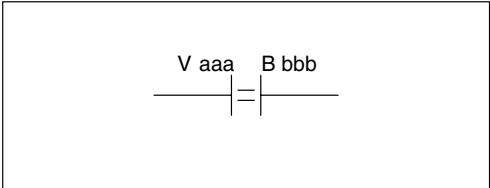


In some cases, you may want certain output points in the specified pause range to operate normally. In that case, use Aux 58 to over-ride the Pause instruction.

# Comparative Boolean

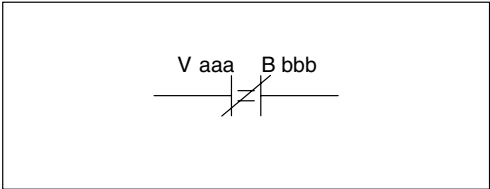
## Store If Equal (STRE)

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



## Store If Not Equal (STRNE)

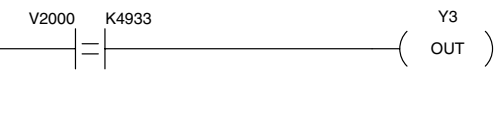
The Store If Not Equal instruction begins a new rung or additional branch in a rung with a normally closed comparative contact. The contact will be on when  $Vaaa \neq Bbbb$ .



Operand Data Type		DL05 Range	
	B	aaa	bbb
V memory	V	All (See page 4–28)	All (See page 4–28)
Pointer	P	All (See page 4–28)	All (See page 4–28)
Constant	K	—	0–9999

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

DirectSOFT

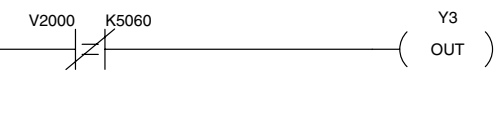


Handheld Programmer Keystrokes

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

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

DirectSOFT

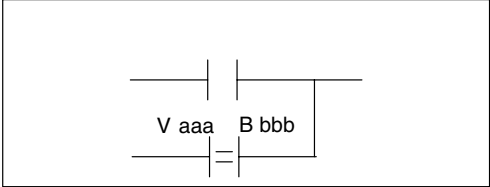


Handheld Programmer Keystrokes

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

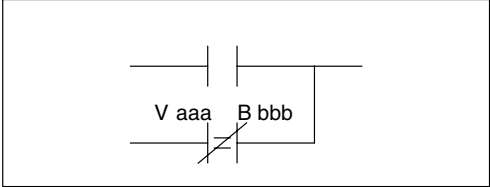
### Or If Equal (ORE)

The Or If Equal instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when  $Vaaa = Bbbb$ .



### Or If Not Equal (ORNE)

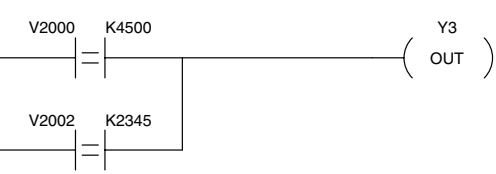
The Or If Not Equal instruction connects a normally closed comparative contact in parallel with another contact. The contact will be on when  $Vaaa \neq Bbbb$ .



Operand Data Type		DL05 Range	
	B	aaa	bbb
V memory	V	All (See page 4-28)	All (See page 4-28)
Pointer	P	All (See page 4-28)	All (See page 4-28)
Constant	K	—	0-9999

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

DirectSOFT

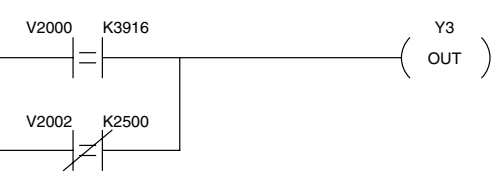


Handheld Programmer Keystrokes

\$	STR	SHFT	E	→	C	A	A	A	→
4			4		2	0	0	0	
E	4	F	A		ENT				
		5	0						
Q	OR	SHFT	E	→	C	A	A	C	→
			4		2	0	0	2	
C	2	D	E		ENT				
		3	4						
GX	OUT	→	D		ENT				
			3						

In the following example, when the value in V memory location V2000 = 3916 or V2002  $\neq$  050, Y3 will energize.

DirectSOFT

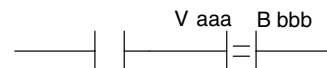


Handheld Programmer Keystrokes

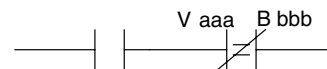
\$	STR	SHFT	E	→	C	A	A	A	→
			4		2	0	0	0	
D	3	J	B		ENT				
		9	1						
R	ORN	SHFT	E	→	C	A	A	C	→
			4		2	0	0	2	
C	2	F	A		ENT				
		5	0						
GX	OUT	→	D		ENT				
			3						

**And If Equal  
(ANDE)**

The And If Equal instruction connects a normally open comparative contact in series with another contact. The contact will be on when  $Vaaa = Bbbb$ .

**And If Not Equal  
(ANDNE)**

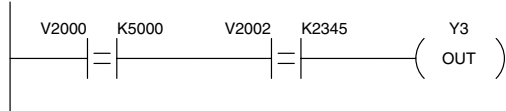
The And If Not Equal instruction connects a normally closed comparative contact in series with another contact. The contact will be on when  $Vaaa \neq Bbbb$ .



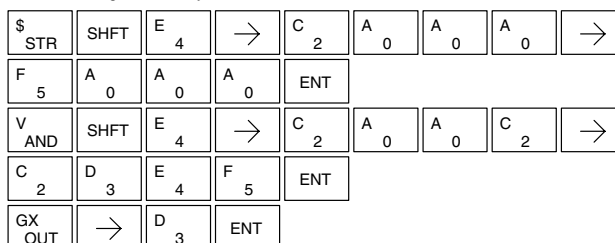
Operand Data Type		DL05 Range	
	A/B	aaa	bbb
V memory	V	All (See page 4-28)	All (See page 4-28)
Pointer	P	All (See page 4-28)	All (See page 4-28)
Constant	K	—	0-9999

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

DirectSOFT

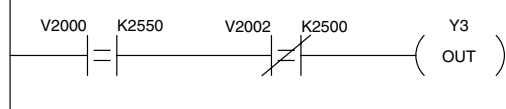


Handheld Programmer Keystrokes

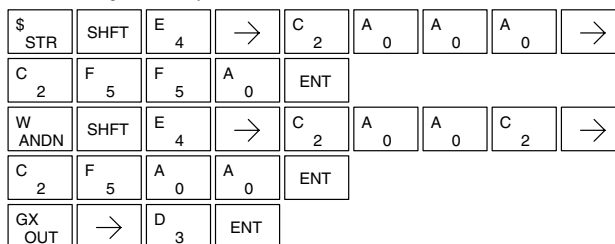


In the following example, when the value in V memory location V2000 = 2550 and V2002  $\neq$  050, Y3 will energize.

DirectSOFT

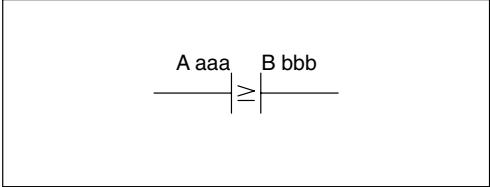


Handheld Programmer Keystrokes



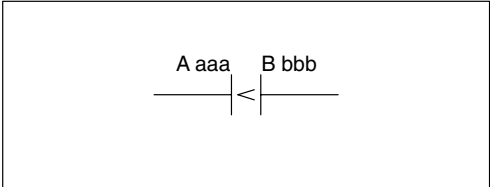
**Store  
(STR)**

The Comparative Store instruction begins a new rung or additional branch in a rung with a normally open comparative contact. The contact will be on when  $Aaaa \geq Bbbb$ .



**Store Not  
(STRN)**

The Comparative Store Not instruction begins a new rung or additional branch in a rung with a normally closed comparative contact. The contact will be on when  $Aaaa < Bbbb$ .



Operand Data Type		DL05 Range	
	A/B	aaa	bbb
V memory	V	All (See page 4-28)	All (See page 4-28)
Pointer	P	All (See page 4-28)	All (See page 4-28)
Constant	K	—	0-9999
Timer	T	0-177	
Counter	CT	0-177	

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

DirectSOFT



Handheld Programmer Keystrokes

\$ STR	→	SHFT	V AND	C 2	A 0	A 0	A 0
→	B 1	A 0	A 0	A 0	ENT		
GX OUT	→	D 3	ENT				

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

DirectSOFT

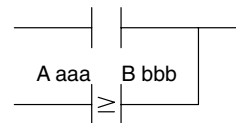


Handheld Programmer Keystrokes

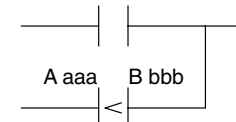
SP STRN	→	SHFT	V AND	C 2	A 0	A 0	A 0
→	E 4	A 0	F 5	A 0	ENT		
GX OUT	→	D 3	ENT				

**Or  
(OR)**

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

**Or Not  
(ORN)**

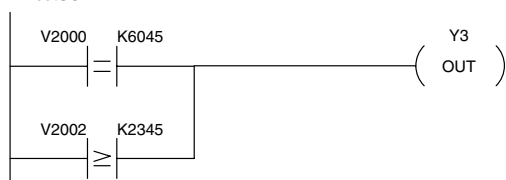
The Comparative Or Not instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when  $Aaaa < Bbbb$ .



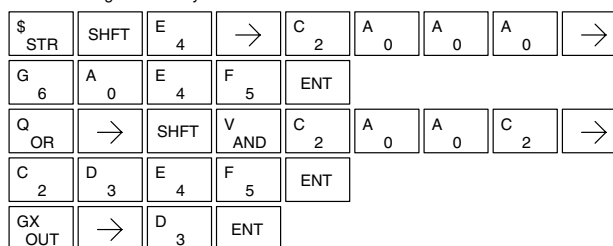
Operand Data Type		DL05 Range	
	A/B	aaa	bbb
V memory	V	All (See page 4-28)	All (See page 4-28)
Pointer	P	All (See page 4-28)	All (See page 4-28)
Constant	K	—	0-9999
Timer	T	0-177	
Counter	CT	0-177	

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

DirectSOFT

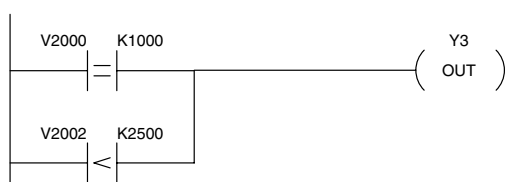


Handheld Programmer Keystrokes

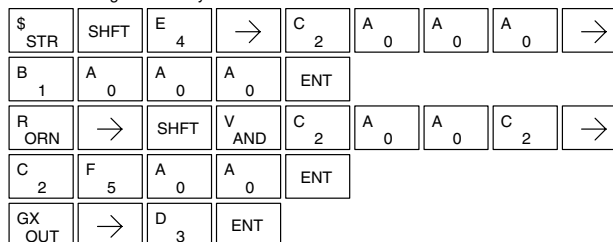


In the following example when the value in V memory location V2000 = 1000 or V2002 < 050, Y3 will energize.

DirectSOFT

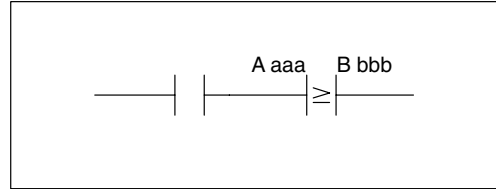


Handheld Programmer Keystrokes



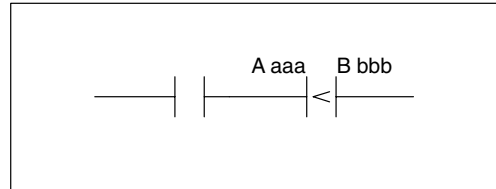
## And (AND)

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



## And Not (ANDN)

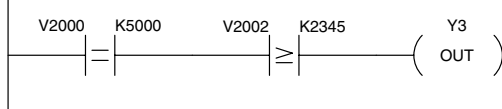
The Comparative And Not instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when  $Aaaa < Bbbb$ .



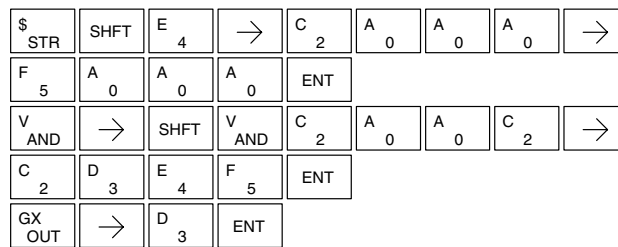
Operand Data Type		DL05 Range	
A/B		aaa	bbb
V memory	V	All (See page 4-28)	All (See page 4-28)
Pointer	P	All (See page 4-28)	All (See page 4-28)
Constant	K	—	0-9999
Timer	T	0-177	
Counter	CT	0-177	

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

DirectSOFT

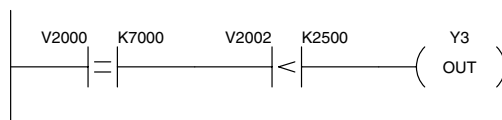


Handheld Programmer Keystrokes

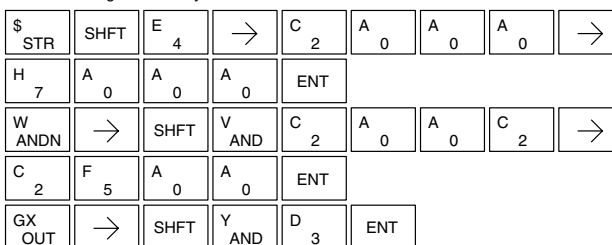


In the following example, when the value in V memory location V2000 = 7000 and  $V2002 < 050$ , Y3 will energize.

DirectSOFT



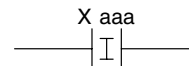
Handheld Programmer Keystrokes



## Immediate Instructions

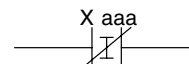
### Store Immediate (STRI)

The Store Immediate instruction begins a new rung or additional branch in a rung. The status of the contact will be the same as the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



### Store Not Immediate (STRNI)

The Store Not Immediate instruction begins a new rung or additional branch in a rung. The status of the contact will be opposite the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



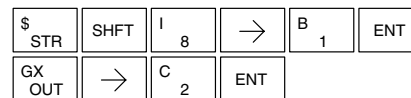
Operand Data Type	DL05 Range
	aaa
Inputs X	0-377

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

DirectSOFT



Handheld Programmer Keystrokes

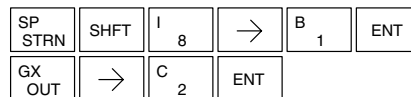


In the following example when X1 is off, Y2 will energize.

DirectSOFT

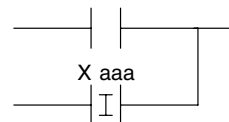


Handheld Programmer Keystrokes



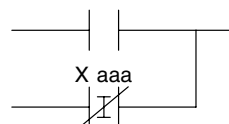
### Or Immediate (ORI)

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



### Or Not Immediate (ORNI)

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



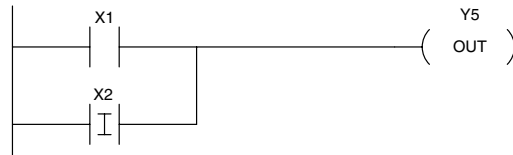


## OR Immediate Instructions Cont'd

Operand Data Type		DL05 Range
		aaa
Inputs	X	0-377

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

DirectSOFT

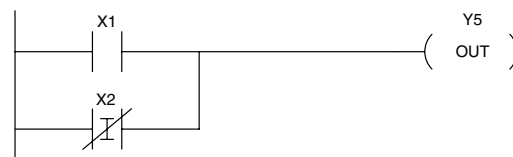


Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT		
Q OR	SHFT	I 8	→	C 2	ENT
GX OUT	→	F 5	ENT		

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

DirectSOFT

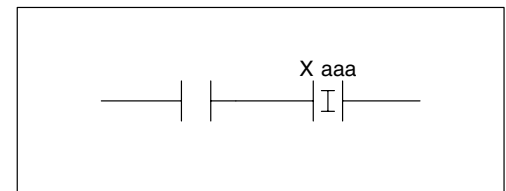


Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT		
R ORN	SHFT	I 8	→	C 2	ENT
GX OUT	→	F 5	ENT		

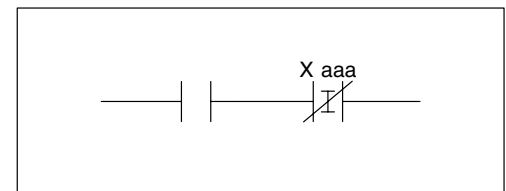
## And Immediate (ANDI)

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



## And Not Immediate (ANDNI)

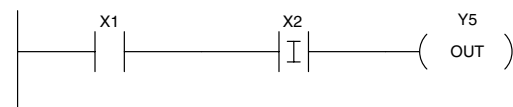
The And Not Immediate connects two contacts in series. The status of the contact will be opposite the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



Operand Data Type		DL05 Range
		aaa
Inputs	X	0-377

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

DirectSOFT

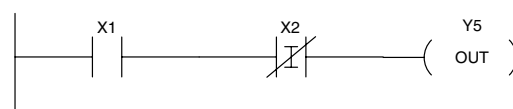


Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT		
V AND	SHFT	I 8	→	C 2	ENT
GX OUT	→	F 5	ENT		

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

DirectSOFT



Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT		
W ANDN	SHFT	I 8	→	C 2	ENT
GX OUT	→	F 5	ENT		

**Out Immediate  
(OUTI)**

The Out Immediate instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) status to the specified module output point and the image register *at the time the instruction is executed*. If multiple Out Immediate instructions referencing the same discrete point are used it is possible for the module output status to change multiple times in a CPU scan. See Or Out Immediate.

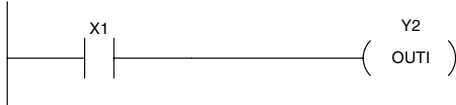
**Or Out Immediate  
(OROUTI)**

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

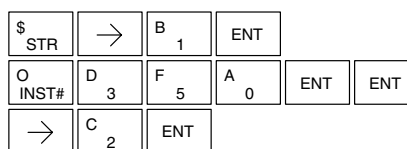
Operand Data Type	DL05 Range
	aaa
Outputs Y	0-377

In the following example, when X1 is on, output point Y2 on the output module will turn on. For instruction entry on the Handheld Programmer, you can use the instruction number (#350) as shown, or type each letter of the command.

DirectSOFT

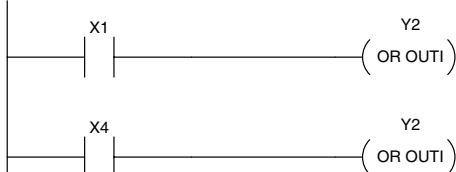


Handheld Programmer Keystrokes

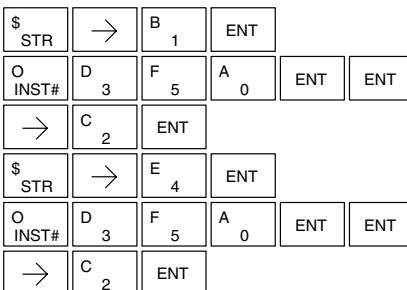


In the following example, when X1 or X4 is on, Y2 will energize.

DirectSOFT

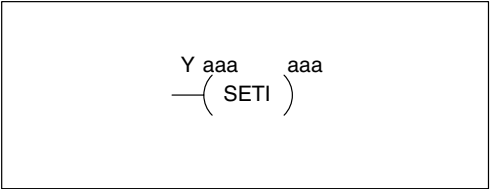


Handheld Programmer Keystrokes



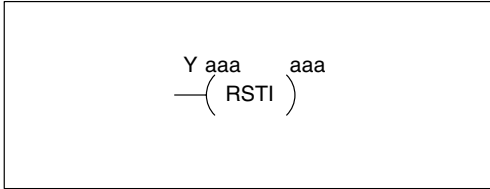
Set Immediate  
(SETI)

The Set Immediate instruction immediately sets, or turns on an output or a range of outputs in the image register and the corresponding output point(s) *at the time the instruction is executed*. Once the outputs are set it is not necessary for the input to remain on. The Reset Immediate instruction can be used to reset the outputs.



Reset  
Immediate  
(RSTI)

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



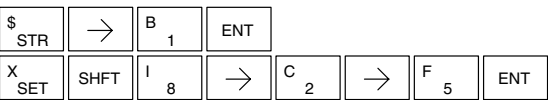
Operand Data Type	DL05 Range
	aaa
Outputs Y	0-377

In the following example, when X1 is on, Y2 through Y5 will be set on in the image register and on the corresponding output points.

DirectSOFT



Handheld Programmer Keystrokes

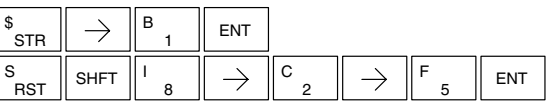


In the following example, when X1 is on, Y5 through Y22 will be reset (off) in the image register and on the corresponding output module(s).

DirectSOFT



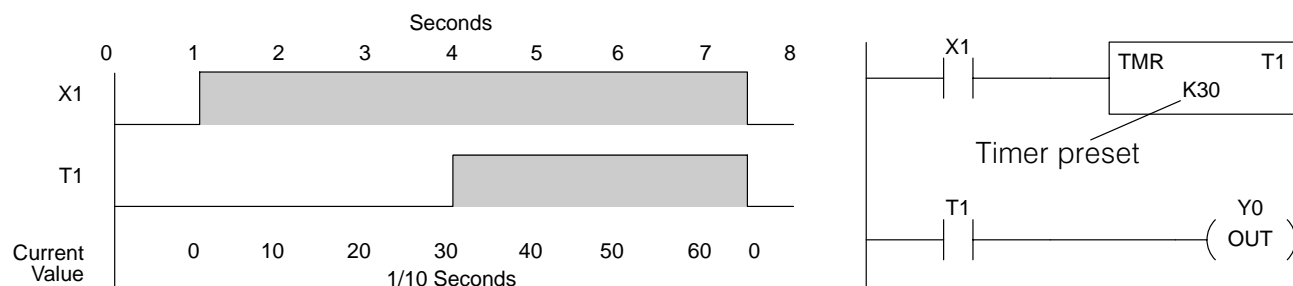
Handheld Programmer Keystrokes



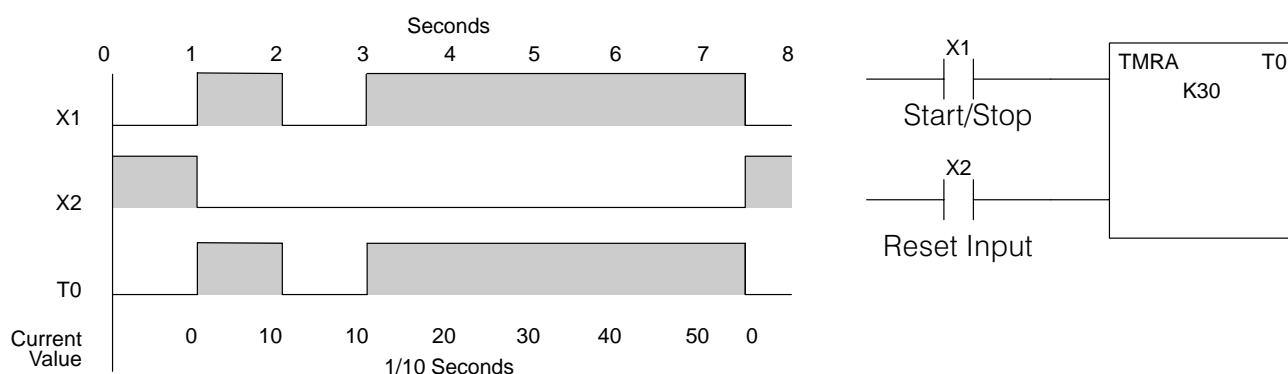
## Timer, Counter and Shift Register Instructions

### Using Timers

Timers are used to time an event for a desired length of time. The single input timer will time as long as the input is on. When the input changes from on to off the timer current value is reset to 0. There is a tenth of a second and a hundredth of a second timer available with a maximum time of 999.9 and 99.99 seconds respectively. There is a discrete bit associated with each timer to indicate that the current value is equal to or greater than the preset value. The timing diagram below shows the relationship between the timer input, associated discrete bit, current value, and timer preset.



There are those applications that need an accumulating timer, meaning it has the ability to time, stop, and then resume from where it previously stopped. The accumulating timer works similarly to the regular timer, but two inputs are required. The start/stop input starts and stops the timer. When the timer stops, the elapsed time is maintained. When the timer starts again, the timing continues from the elapsed time. When the reset input is turned on, the elapsed time is cleared and the timer will start at 0 when it is restarted. There is a tenth of a second and a hundredth of a second timer available with a maximum time of 9999999.9 and 999999.99 seconds respectively. The timing diagram below shows the relationship between the timer input, timer reset, associated discrete bit, current value, and timer preset.



**Timer (TMR) and  
Timer Fast (TMRF)**

The Timer instruction is a 0.1 second single input timer that times to a maximum of 999.9 seconds. The Timer Fast instruction is a 0.01 second single input timer that times up to a maximum of 99.99 seconds. These timers will be enabled if the input logic is true (on) and will be reset to 0 if the input logic is false (off).

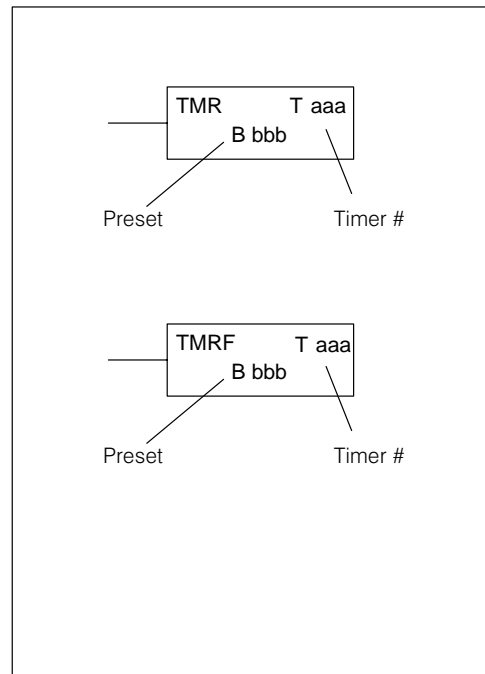
**Instruction Specifications**

**Timer Reference (Taaa):** Specifies the timer number.

**Preset Value (Bbbb):** Constant value (K) or a V memory location.

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

**Discrete Status Bit:** The discrete status bit is referenced by the associated T memory location. Operating as a “timer done bit”, it will be on if the current value is equal to or greater than the preset value. For example, the discrete status bit for Timer 2 is TA2.

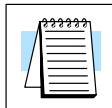


The timer discrete status bit and the current value are not specified in the timer instruction.



**NOTE:** Timer preset constants (K) may be changed by using a handheld programmer, even when the CPU is in Run Mode. Therefore, a V-memory preset is required only if the ladder program must change the preset.

Operand Data Type		DL05 Range	
A/B		aaa	bbb
Timers	T	0-177	—
V memory for preset values	V	—	1200-7377 7400-7577
Pointers (preset only)	P	—	1200-7377 7400-7577
Constants (preset only)	K	—	0-9999
Timer discrete status bits	T/V	0-177 or V41100-41107	
Timer current values	V / T*	0-177	



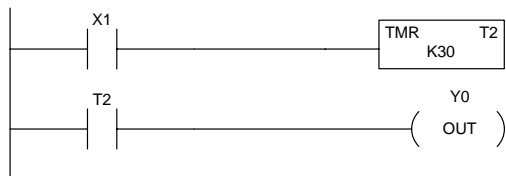
**NOTE:** \* With the HPP, both the Timer discrete status bits and current value are accessed with the same data reference. **DirectSOFT** uses separate references, such as “T2” for discrete status bit for Timer T2, and “TA2” for the current value of Timer T2.

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

### Timer Example Using Discrete Status Bits

In the following example, a single input timer is used with a preset of 3 seconds. The timer discrete status bit (T2) will turn on when the timer has timed for 3 seconds. The timer is reset when X1 turns off, turning the discrete status bit off and resetting the timer current value to 0.

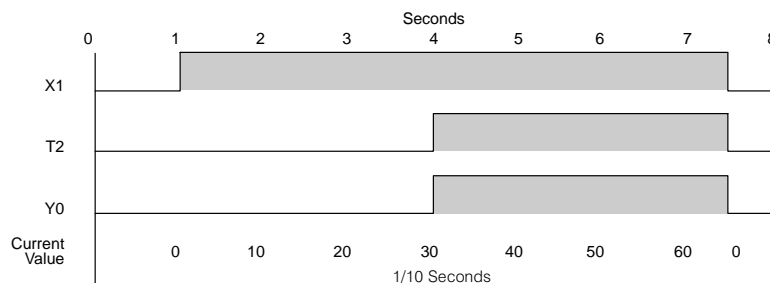
DirectSOFT



Handheld Programmer Keystrokes

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

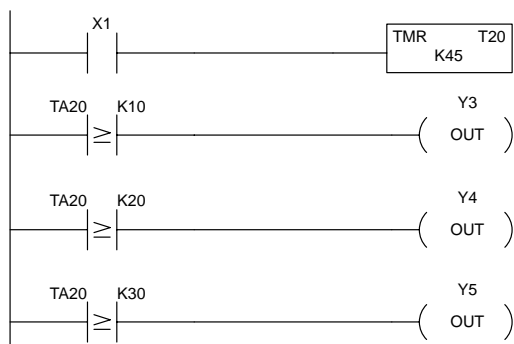
Timing Diagram



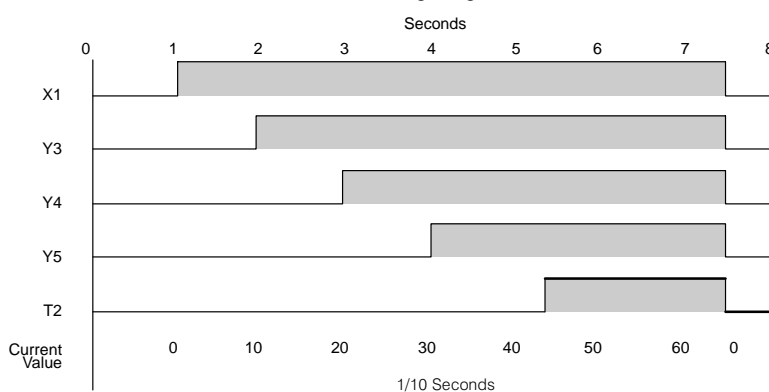
### Timer Example Using Comparative Contacts

In the following example, a single input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energize Y3, Y4, and Y5 at one second intervals respectively. When X1 is turned off the timer will be reset to 0 and the comparative contacts will turn off Y3, Y4, and Y5.

DirectSOFT



Timing Diagram



Handheld Programmer Keystrokes

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

**Accumulating  
Timer (TMRA)****Accumulating Fast  
Timer (TMRAF)**

The Accumulating Timer is a 0.1 second two input timer that will time to a maximum of 9999999.9. The Accumulating Fast Timer is a 0.01 second two-input timer that will time to a maximum of 99999.99. *Each one uses two timer registers in V-memory.* These timers have two inputs, an enable and a reset. The timer starts timing when the enable is on and stops when the enable is off (without resetting the count). The reset will reset the timer when on and allow the timer to time when off.

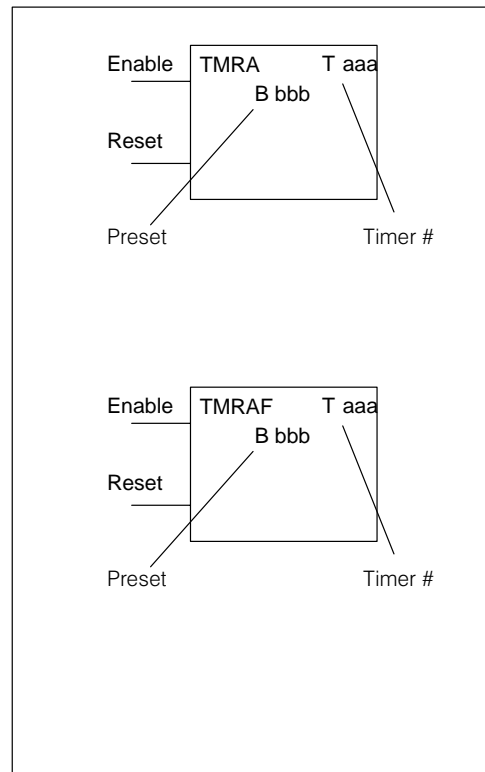
**Instruction Specifications**

**Timer Reference (Taaa):** Specifies the timer number.

**Preset Value (Bbbb):** Constant value (K) or a V memory location.

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

**Discrete Status Bit:** The discrete status bit is accessed by referencing the associated T memory location. Operating as a “timer done bit”, it will be on if the current value is equal to or greater than the preset value. For example the discrete status bit for timer 2 would be T2.



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



**NOTE:** The accumulating type timer uses **two consecutive V-memory locations** for the 8-digit value, and therefore two consecutive timer locations. For example, if TMR 1 is used, the next available timer number is TMR 3.

Operand Data Type	DL05 Range		
	A/B	aaa	bbb
Timers	T	0-176	—
V memory for preset values	V	—	1200-7377 7400-7577
Pointers (preset only)	P	—	1200-7377 7400-7577
Constants (preset only)	K	—	0-99999999
Timer discrete status bits	T/V	0-176 or V41100-41107	
Timer current values	V / T*	0-176	



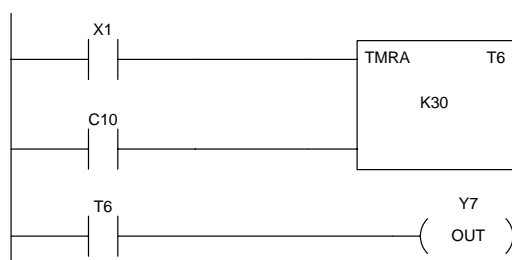
**NOTE:** \* With the HPP, both the Timer discrete status bits and current value are accessed with the same data reference. **DirectSOFT** uses separate references, such as “T2” for discrete status bit for Timer T2, and “TA2” for the current value of Timer T2.

The following examples show two methods of programming timers. One performs functions when the timer reaches the preset value using the discrete status bit, or use comparative contacts to perform functions at different time intervals.

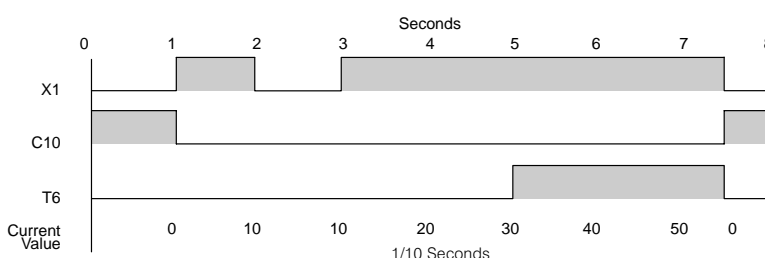
### Accumulating Timer Example using Discrete Status Bits

In the following example, a two input timer (accumulating timer) is used with a preset of 3 seconds. The timer discrete status bit (T6) will turn on when the timer has timed for 3 seconds. Notice in this example that the timer times for 1 second, stops for one second, then resumes timing. The timer will reset when C10 turns on, turning the discrete status bit off and resetting the timer current value to 0.

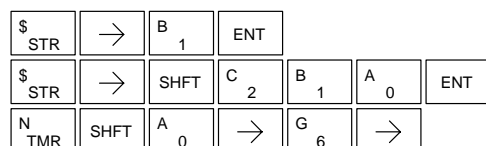
DirectSOFT



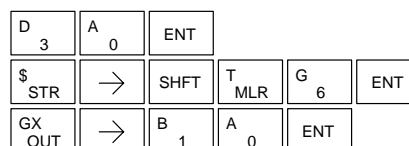
Timing Diagram



Handheld Programmer Keystrokes



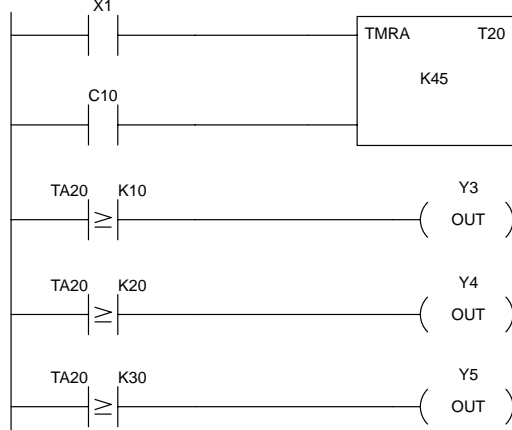
Handheld Programmer Keystrokes (cont)



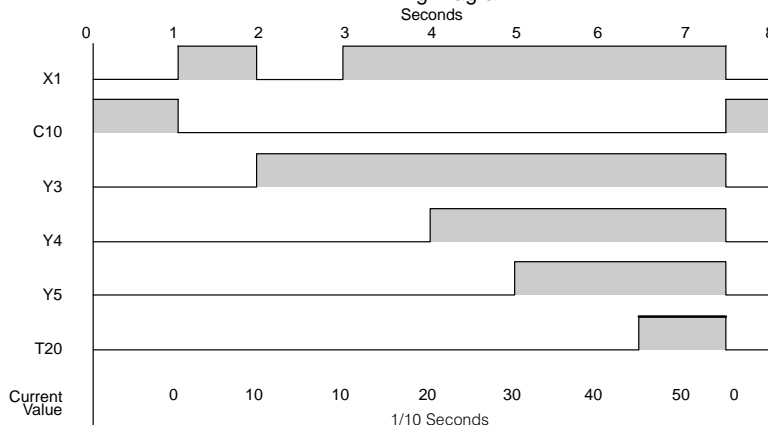
### Accumulator Timer Example Using Comparative Contacts

In the following example, a single input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energized Y3, Y4, and Y5 at one second intervals respectively. The comparative contacts will turn off when the timer is reset.

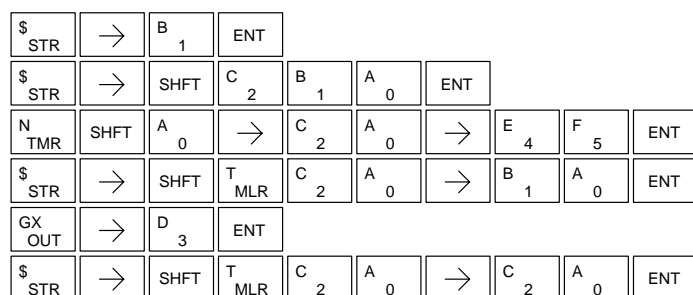
DirectSOFT



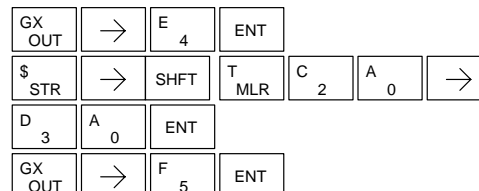
Timing Diagram



Handheld Programmer Keystrokes



Handheld Programmer Keystrokes (cont)

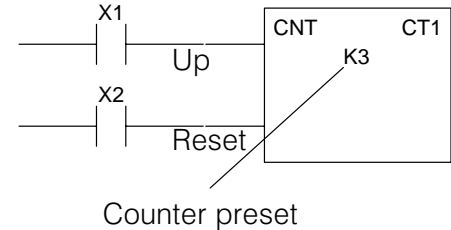
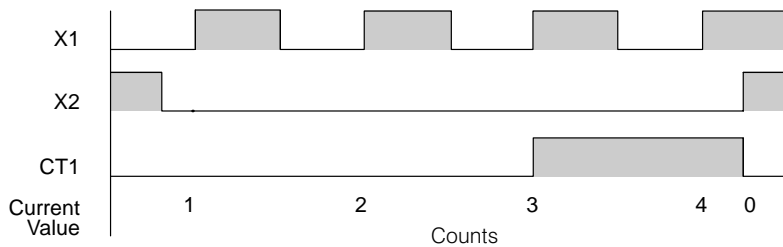




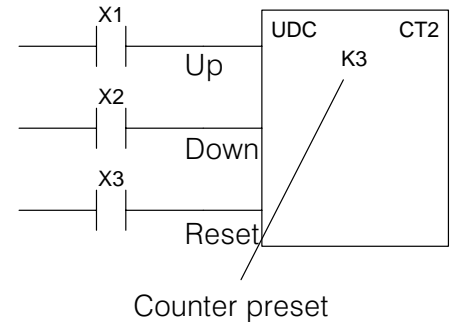
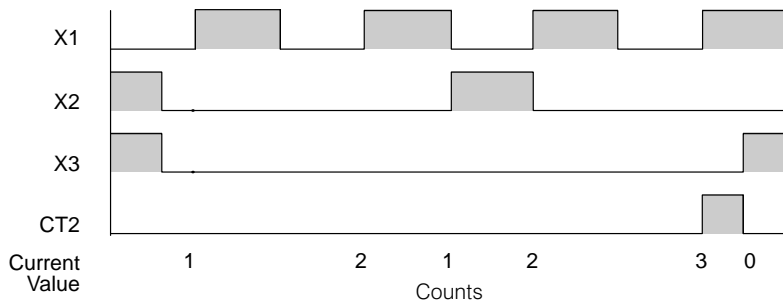
## Using Counters

Counters are used to count events. The counters available are up counters, up/down counters, and stage counters (used with RLL<sup>PLUS</sup> programming).

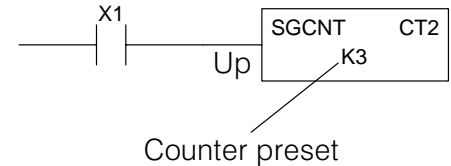
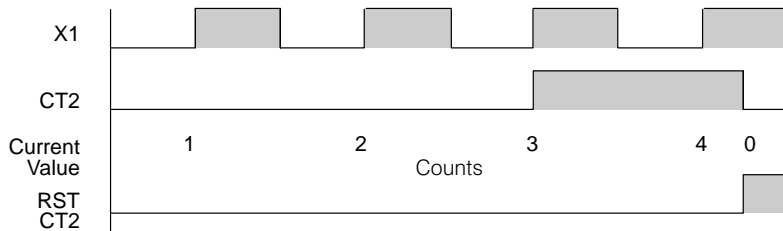
The up counter has two inputs, a count input and a reset input. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, counter reset, associated discrete bit, current value, and counter preset.



The up down counter has three inputs, a count up input, count down input and reset input. The maximum count value is 99999999. The timing diagram below shows the relationship between the counter input, counter reset, associated discrete bit, current value, and counter preset.



The stage counter has a count input and is reset by the RST instruction. This instruction is useful when programming using the RLL<sup>PLUS</sup> structured programming. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, associated discrete bit, current value, counter preset and reset instruction.



**Counter  
(CNT)**

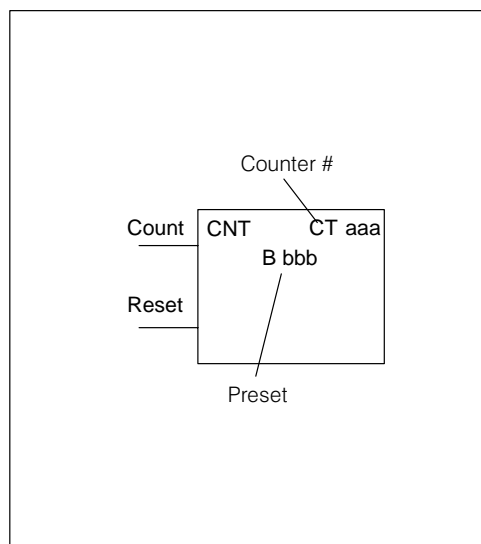
The Counter is a two input counter that increments when the count input logic transitions from off to on. When the counter reset input is on the counter resets to 0. When the current value equals the preset value, the counter status bit comes on and the counter continues to count up to a maximum count of 9999. The maximum value will be held until the counter is reset.

**Instruction Specifications**

**Counter Reference (CTaaa):** Specifies the counter number.

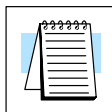
**Preset Value (Bbbb):** Constant value (K) or a V memory location.

**Current Values:** Counter current values are accessed by referencing the associated V or CT memory locations\*. The V-memory location is the counter location + 1000. For example, the counter current value for CT3 resides in V memory location V1003.



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

**Discrete Status Bit:** The discrete status bit is accessed by referencing the associated CT memory location. It will be on if the value is equal to or greater than the preset value. For example the discrete status bit for counter 2 would be CT2.



**NOTE:** Counter preset constants (K) may be changed by using a programming device, even when the CPU is in Run Mode. Therefore, a V-memory preset is required only if the ladder program must change the preset.

Operand Data Type		DL05 Range	
A/B		aaa	bbb
Counters	CT	0-177	—
V memory (preset only)	V	—	1200-7377 7400-7577
Pointers (preset only)	P	—	1200-7377 7400-7577
Constants (preset only)	K	—	0-9999
Counter discrete status bits	CT/V	0-177 or V41140-41147	
Counter current values	V/CT*	0-177	

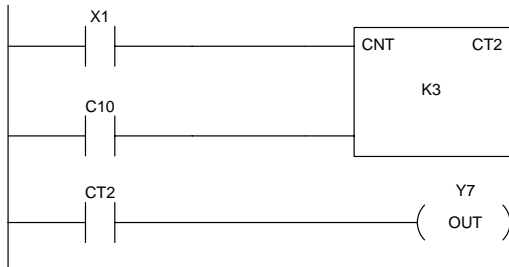


**NOTE:** \* With the HPP, both the Counter discrete status bits and current value are accessed with the same data reference. **DirectSOFT** uses separate references, such as “CT2” for discrete status bit for Counter CT2, and “CTA2” for the current value of Counter CT2.

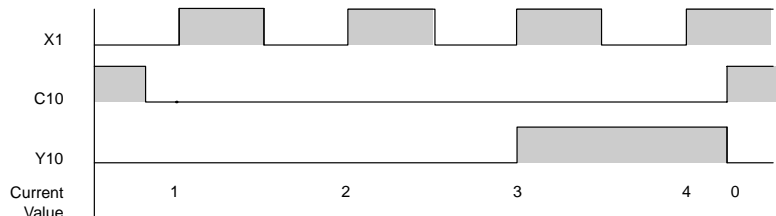
### Counter Example Using Discrete Status Bits

In the following example, when X1 makes an off to on transition, counter CT2 will increment by one. When the current value reaches the preset value of 3, the counter status bit CT2 will turn on and energize Y7. When the reset C10 turns on, the counter status bit will turn off and the current value will be 0. The current value for counter CT2 will be held in V memory location V1002.

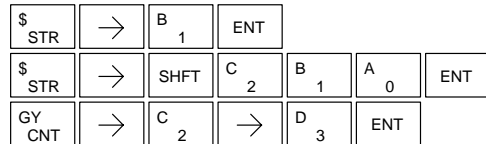
DirectSOFT



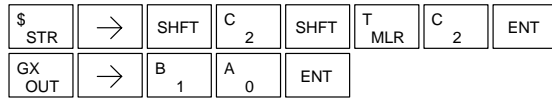
Counting diagram



Handheld Programmer Keystrokes



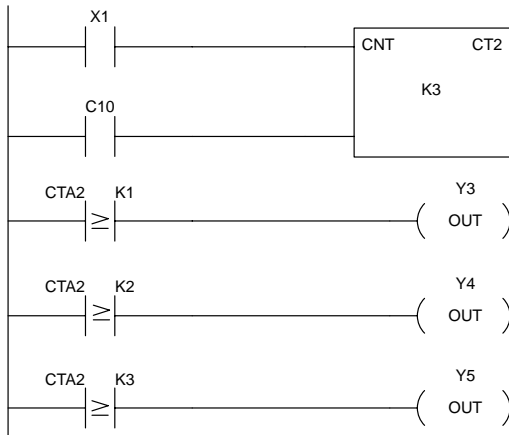
Handheld Programmer Keystrokes (cont)



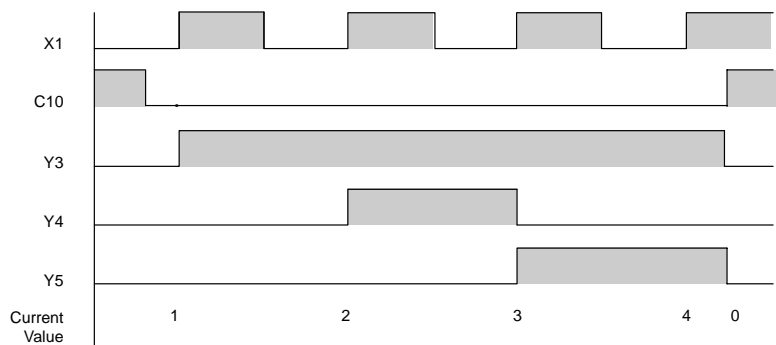
### Counter Example Using Comparative Contacts

In the following example, when X1 makes an off to on transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3, Y4, and Y5 at different counts. When the reset C10 turns on, the counter status bit will turn off and the counter current value will be 0, and the comparative contacts will turn off.

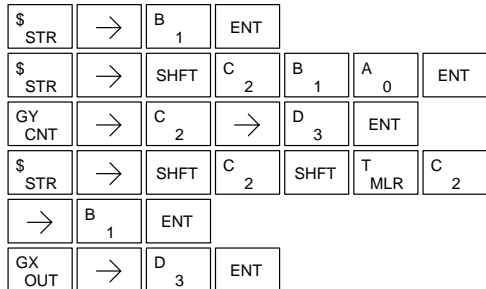
DirectSOFT



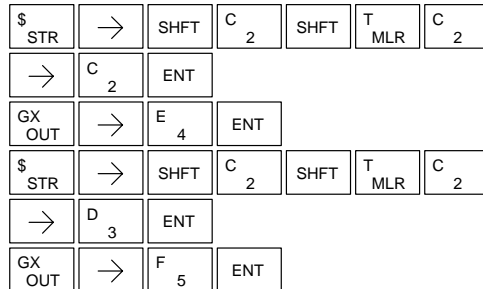
Counting diagram



Handheld Programmer Keystrokes

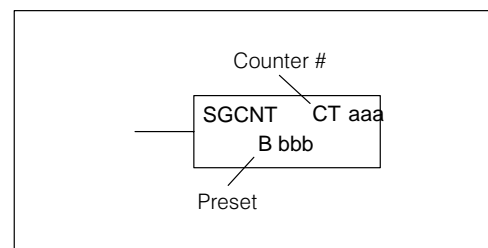


Handheld Programmer Keystrokes (cont)



**Stage Counter  
(SGCNT)**

The Stage Counter is a single input counter that increments when the input logic transitions from off to on. This counter differs from other counters since it will hold its current value until reset using the RST instruction. The Stage Counter is designed for use in RLL *PLUS* programs but can be used in relay ladder logic programs. When the current value equals the preset value, the counter status bit turns on and the counter continues to count up to a maximum count of 9999. The maximum value will be held until the counter is reset.



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

**Instruction Specifications**

**Counter Reference (CTaaa):** Specifies the counter number.

**Preset Value (Bbbb):** Constant value (K) or a V memory location.

**Current Values:** Counter current values are accessed by referencing the associated V or CT memory locations\*. The V-memory location is the counter location + 1000. For example, the counter current value for CT3 resides in V memory location V1003.

**Discrete Status Bit:** The discrete status bit is accessed by referencing the associated CT memory location. It will be on if the value is equal to or greater than the preset value. For example the discrete status bit for counter 2 would be CT2.

Operand Data Type		DL05 Range	
A/B		aaa	bbb
Counters	CT	0-177	—
V memory (preset only)	V	—	1200-7377 7400-7577
Pointers (preset only)	P	—	1200-7377 7400-7577
Constants (preset only)	K	—	0-9999
Counter discrete status bits	CT/V	0-177 or V41140-41147	
Counter current values	V/CT*	1000-1177	

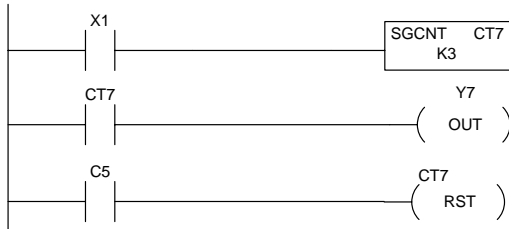
**NOTE:** \* With the HPP, both the Counter discrete status bits and current value are accessed with the same data reference. **DirectSOFT** uses separate references, such as “CT2” for discrete status bit for Counter CT2, and “CTA2” for the current value of Counter CT2.



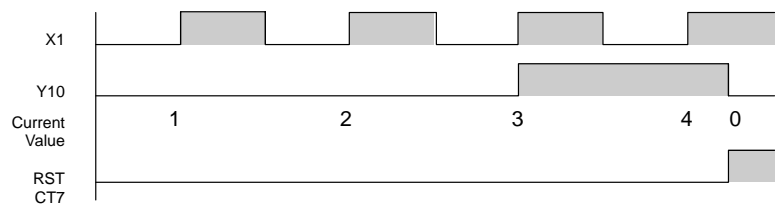
### Stage Counter Example Using Discrete Status Bits

In the following example, when X1 makes an off to on transition, stage counter CT7 will increment by one. When the current value reaches 3, the counter status bit CT7 will turn on and energize Y7. The counter status bit CT7 will remain on until the counter is reset using the RST instruction. When the counter is reset, the counter status bit will turn off and the counter current value will be 0. The current value for counter CT7 will be held in V memory location V1007.

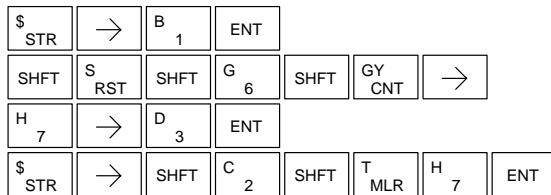
DirectSOFT



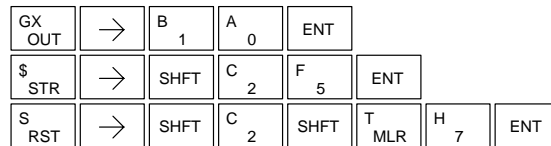
Counting diagram



Handheld Programmer Keystrokes



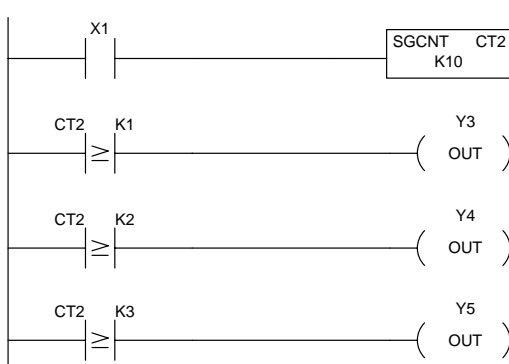
Handheld Programmer Keystrokes (cont)



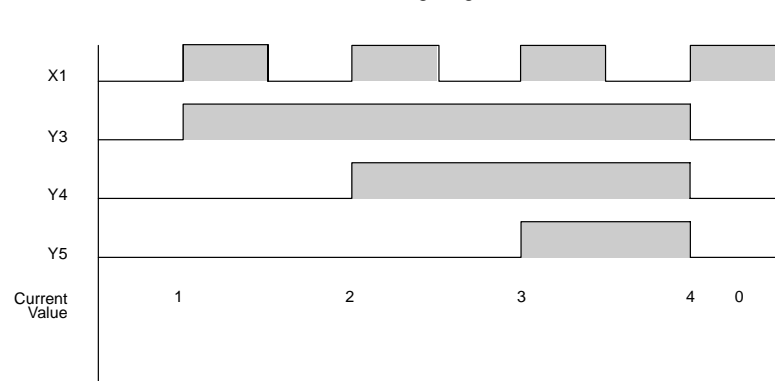
### Stage Counter Example Using Comparative Contacts

In the following example, when X1 makes an off to on transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3, Y4, and Y5 at different counts. Although this is not shown in the example, when the counter is reset using the Reset instruction, the counter status bit will turn off and the current value will be 0. The current value for counter CT2 will be held in V memory location V1002.

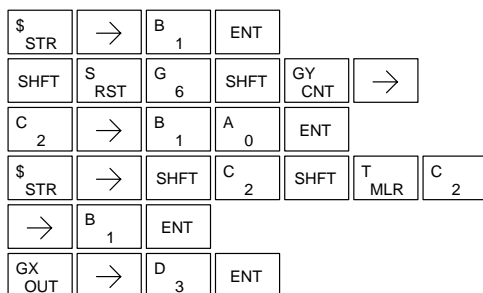
DirectSOFT



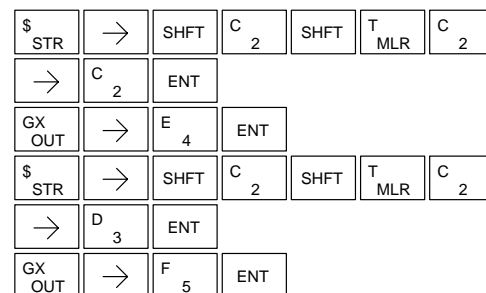
Counting diagram



Handheld Programmer Keystrokes



Handheld Programmer Keystrokes (cont)



**Up Down Counter (UDC)**

This Up/Down Counter counts up on each off to on transition of the Up input and counts down on each off to on transition of the Down input. The counter is reset to 0 when the Reset input is on. The count range is 0–99999999. The count input not being used must be off in order for the active count input to function.

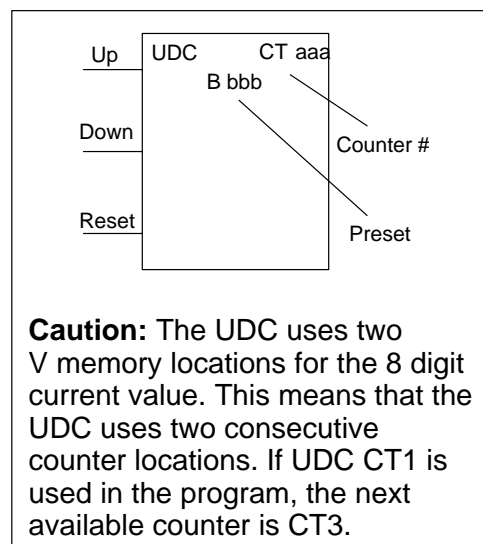
**Instruction Specification**

**Counter Reference (CTaaa):** Specifies the counter number.

**Preset Value (Bbbb):** Constant value (K) or two consecutive V memory locations.

**Current Values:** Current count is a double word value accessed by referencing the associated V or CT memory locations\*. The V-memory location is the counter location + 1000. For example, the counter current value for CT5 resides in V memory location V1005 and V1006.

**Discrete Status Bit:** The discrete status bit is accessed by referencing the associated CT memory location. Operating as a “counter done bit” it will be on if the value is equal to or greater than the preset value. For example the discrete status bit for counter 2 would be CT2.



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

Operand Data Type		DL05 Range	
A/B		aaa	bbb
Counters	CT	0–176	—
V memory (preset only)	V	—	1200–7377 7400–7577
Pointers (preset only)	P	—	1200–7377 7400–7577
Constants (preset only)	K	—	0–99999999
Counter discrete status bits	CT/V	0–176 or V41140–41147	
Counter current values	V/CT*	0–176	

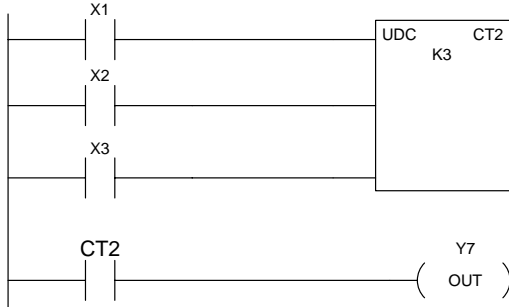
**NOTE:** \* With the HPP, both the Counter discrete status bits and current value are accessed with the same data reference. **DirectSOFT** uses separate references, such as “CT2” for discrete status bit for Counter CT2, and “CTA2” for the current value of Counter CT2.



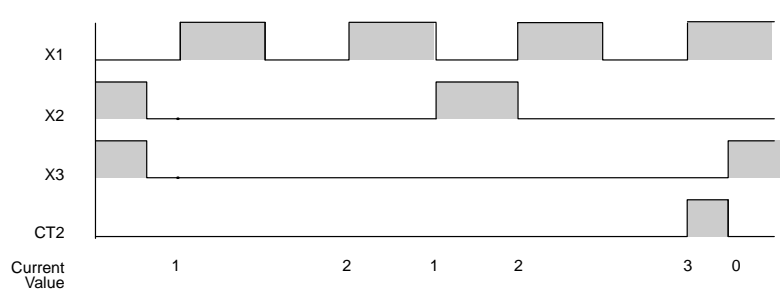
### Up / Down Counter Example Using Discrete Status Bits

In the following example if X2 and X3 are off, when X1 toggles from off to on the counter will increment by one. If X1 and X3 are off the counter will decrement by one when X2 toggles from off to on. When the count value reaches the preset value of 3, the counter status bit will turn on. When the reset X3 turns on, the counter status bit will turn off and the current value will be 0.

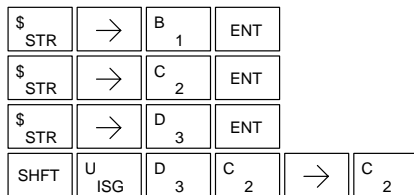
DirectSOFT



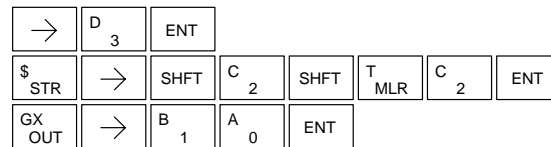
Counting Diagram



Handheld Programmer Keystrokes



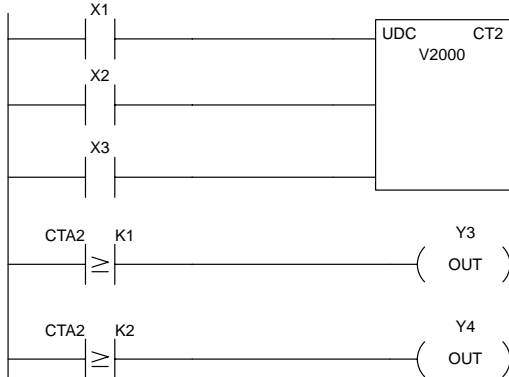
Handheld Programmer Keystrokes (cont)



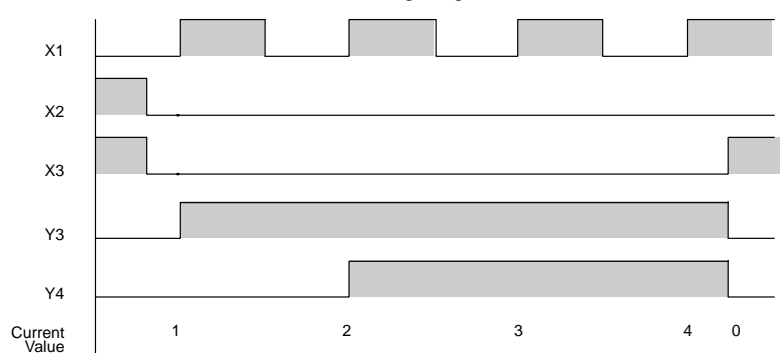
### Up / Down Counter Example Using Comparative Contacts

In the following example, when X1 makes an off to on transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3 and Y4 at different counts. When the reset (X3) turns on, the counter status bit will turn off, the current value will be 0, and the comparative contacts will turn off.

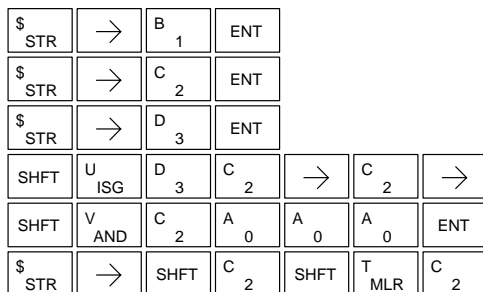
DirectSOFT



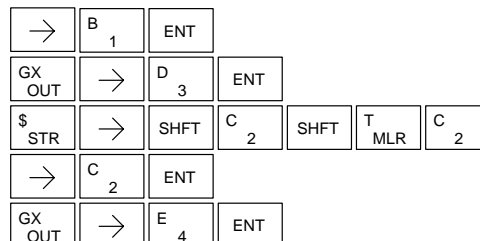
Counting Diagram



Handheld Programmer Keystrokes



Handheld Programmer Keystrokes (cont)

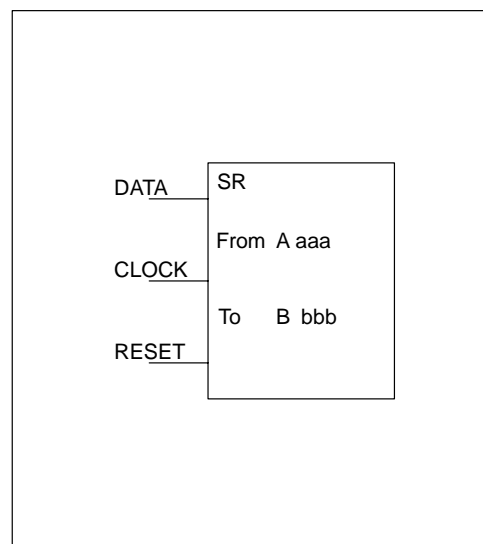


## Shift Register (SR)

The Shift Register instruction shifts data through a predefined number of control relays. The control ranges in the shift register block must start at the beginning of an 8 bit boundary use 8-bit blocks.

The Shift Register has three contacts.

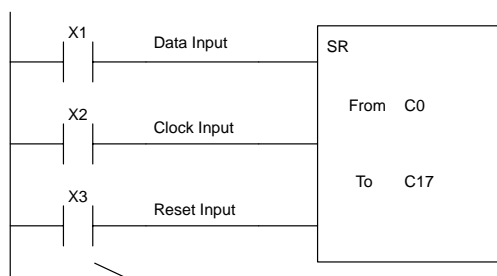
- Data — determines the value (1 or 0) that will enter the register
- Clock — shifts the bits one position on each low to high transition
- Reset — resets the Shift Register to all zeros.



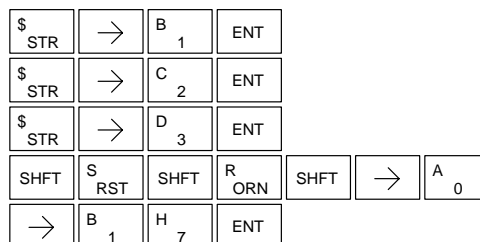
With each off to on transition of the clock input, the bits which make up the shift register block are shifted by one bit position and the status of the data input is placed into the starting bit position in the shift register. The direction of the shift depends on the entry in the From and To fields. From C0 to C17 would define a block of sixteen bits to be shifted from left to right. From C17 to C0 would define a block of sixteen bits, to be shifted from right to left. The maximum size of the shift register block depends on the number of available control relays. The minimum block size is 8 control relays.

Operand Data Type		DL05 Range	
A/B		aaa	bbb
Control Relay	C	0-777	0-777

### DirectSOFT



### Handheld Programmer Keystrokes



### Inputs on Successive Scans

### Shift Register Bits

Data	Clock	Reset		C0	C17
1	0-1-0	0	—	■	
0	0-1-0	0	—	■	
0	0-1-0	0	—		■
1	0-1-0	0	—	■	■
0	0-1-0	0	—	■	■
0	0	1	—		

■ - indicates on      □ - indicates off



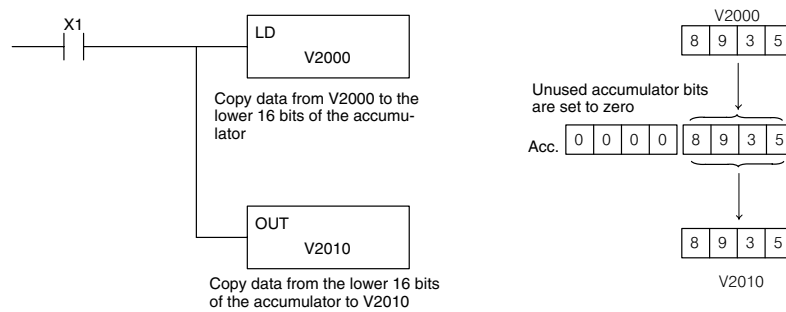
## Accumulator / Stack Load and Output Data Instructions

### Using the Accumulator

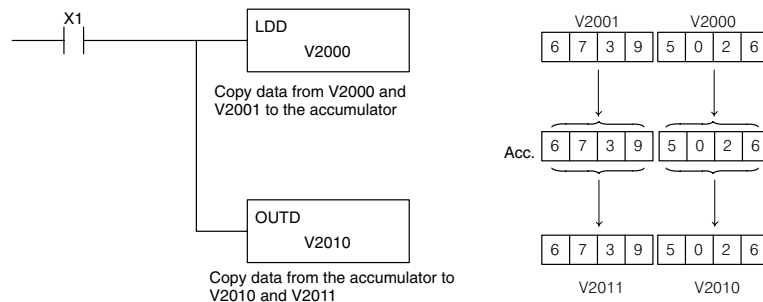
The accumulator in the DL05 internal CPUs is a 32 bit register which is used as a temporary storage location for data that is being copied or manipulated in some manor. For example, you have to use the accumulator to perform math operations such as add, subtract, multiply, etc. Since there are 32 bits, you can use up to an 8-digit BCD number. The accumulator is reset to 0 at the end of every CPU scan.

### Copying Data to the Accumulator

The Load and Out instructions and their variations are used to copy data from a V-memory location to the accumulator, or, to copy data from the accumulator to V memory. The following example copies data from V-memory location V2000 to V-memory location V2010.

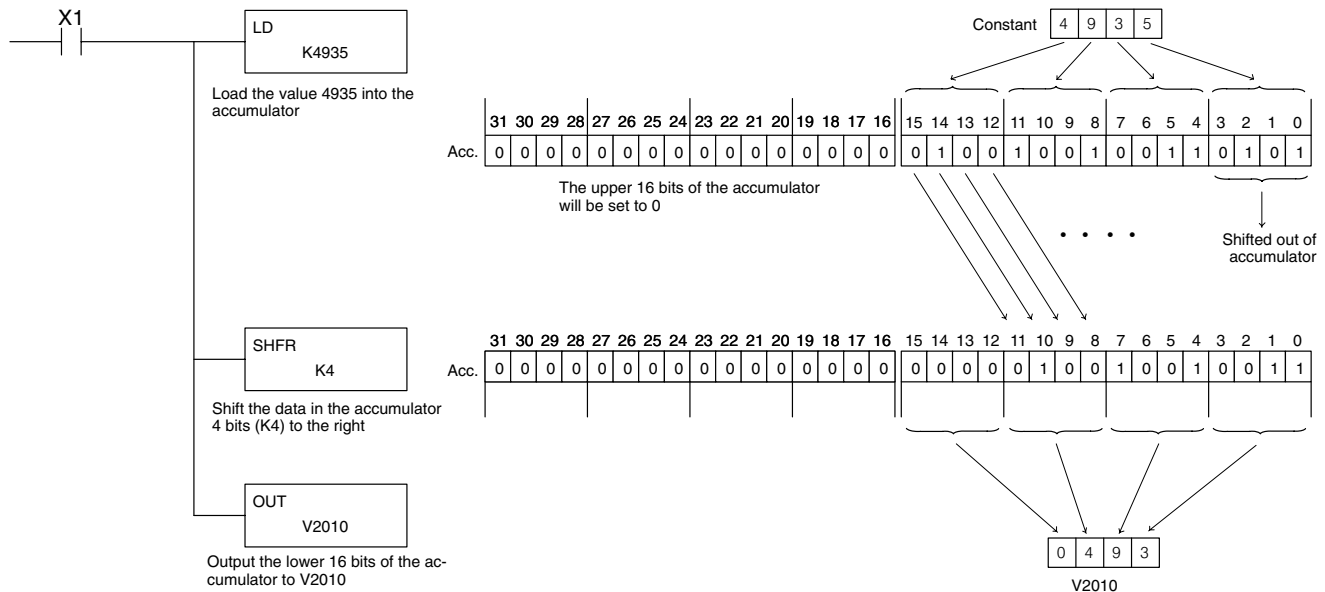


Since the accumulator is 32 bits and V memory locations are 16 bits the Load Double and Out Double (or variations thereof) use two consecutive V-memory locations or 8 digit BCD constants to copy data either to the accumulator from a V-memory address or from a V-memory address to the accumulator. For example if you wanted to copy data from V2000 and V2001 to V2010 and V2011 the most efficient way to perform this function would be as follows:



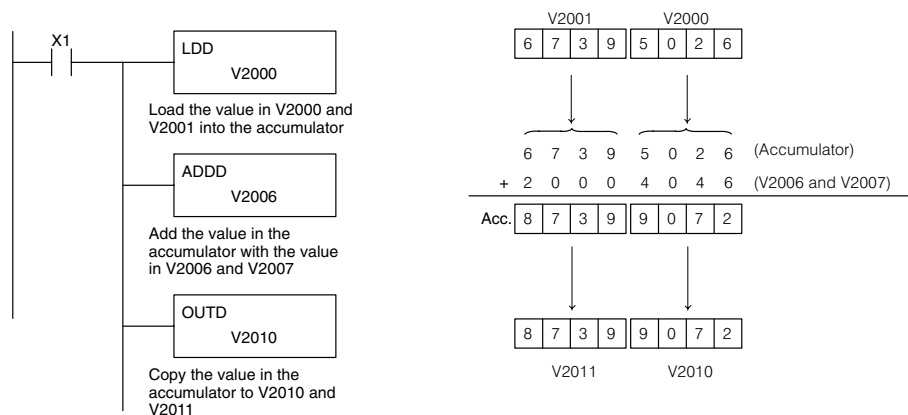
### Changing the Accumulator Data

Instructions that manipulate data also use the accumulator. The result of the manipulated data resides in the accumulator. The data that was being manipulated is cleared from the accumulator. The following example loads the constant value 4935 into the accumulator, shifts the data right 4 bits, and outputs the result to V2010.



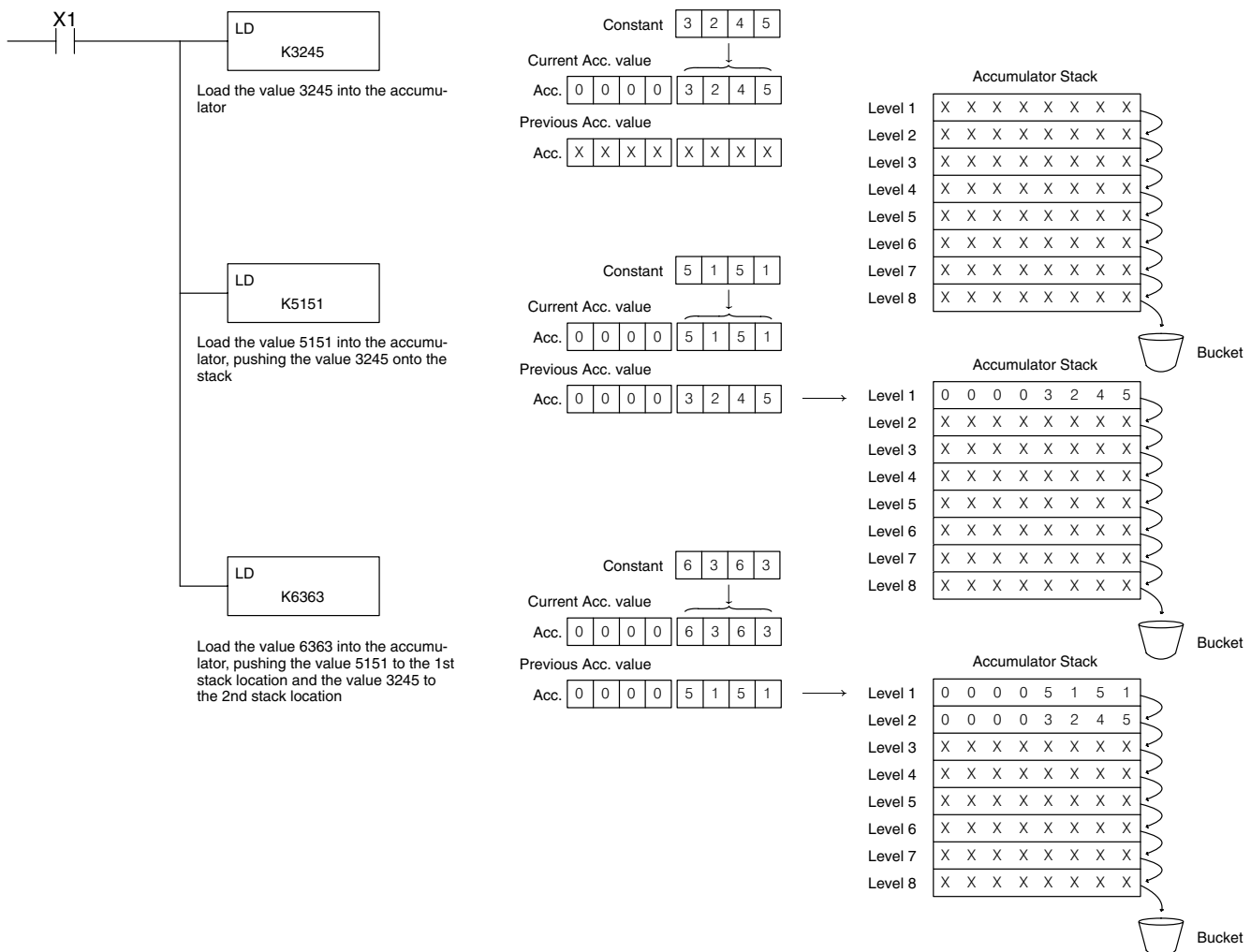
Some of the data manipulation instructions use 32 bits. They use two consecutive V memory locations or an 8 digit BCD constant to manipulate data in the accumulator.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is added with the value in V2006 and V2007 using the Add Double instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

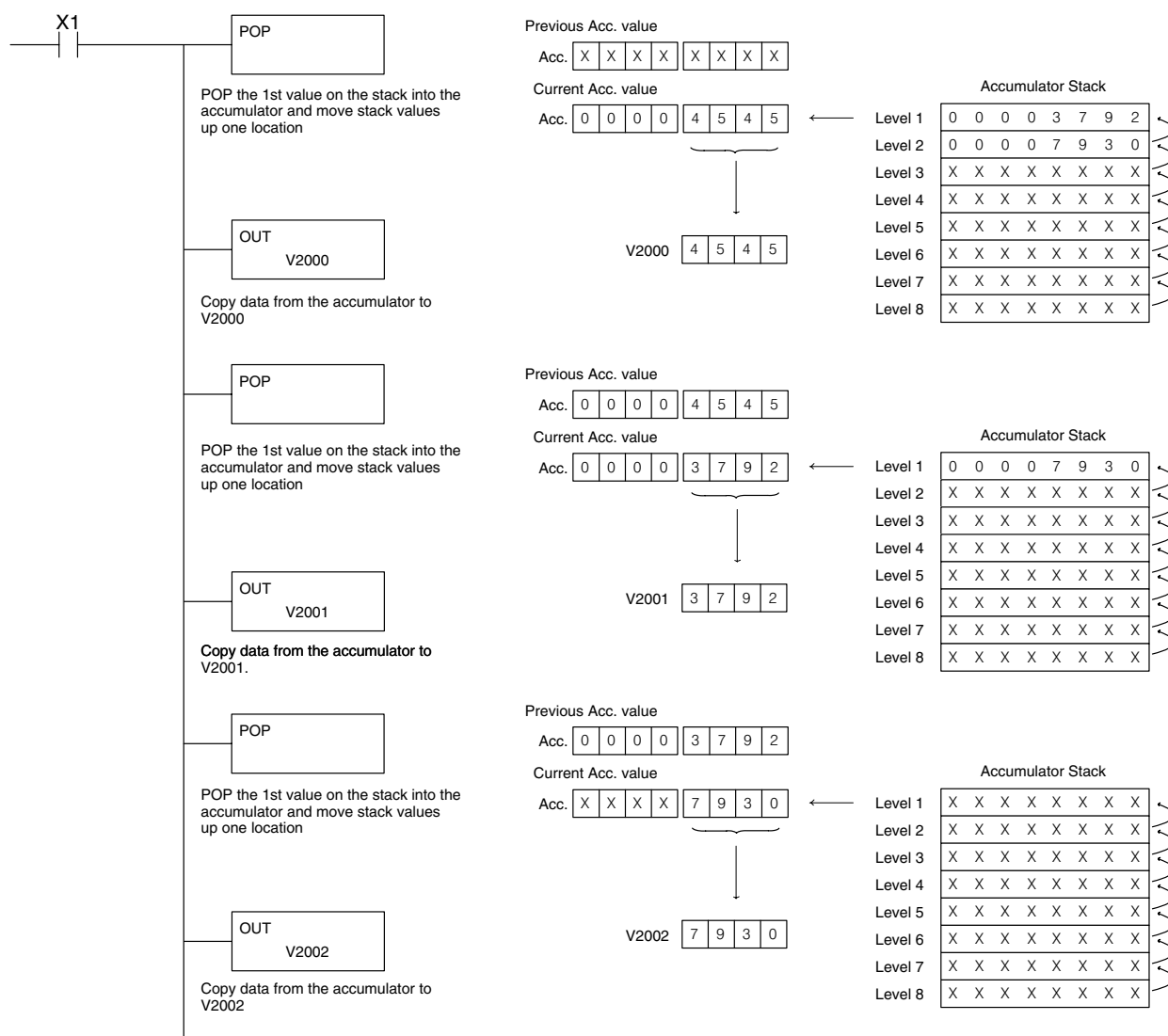


## Using the Accumulator Stack

The accumulator stack is used for instructions that require more than one parameter to execute a function or for user defined functionality. The accumulator stack is used when more than one Load instruction is executed without the use of an Out instruction. The first load instruction in the scan places a value into the accumulator. Every Load instruction thereafter without the use of an Out instruction places a value into the accumulator and the value that was in the accumulator is placed onto the accumulator stack. The Out instruction nullifies the previous load instruction and does not place the value that was in the accumulator onto the accumulator stack when the next load instruction is executed. Every time a value is placed onto the accumulator stack the other values in the stack are pushed down one location. The accumulator is eight levels deep (eight 32 bit registers). If there is a value in the eighth location when a new value is placed onto the stack, the value in the eighth location is pushed off the stack and cannot be recovered.



The POP instruction rotates values upward through the stack into the accumulator. When a POP is executed the value which was in the accumulator is cleared and the value that was on top of the stack is in the accumulator. The values in the stack are shifted up one position in the stack.



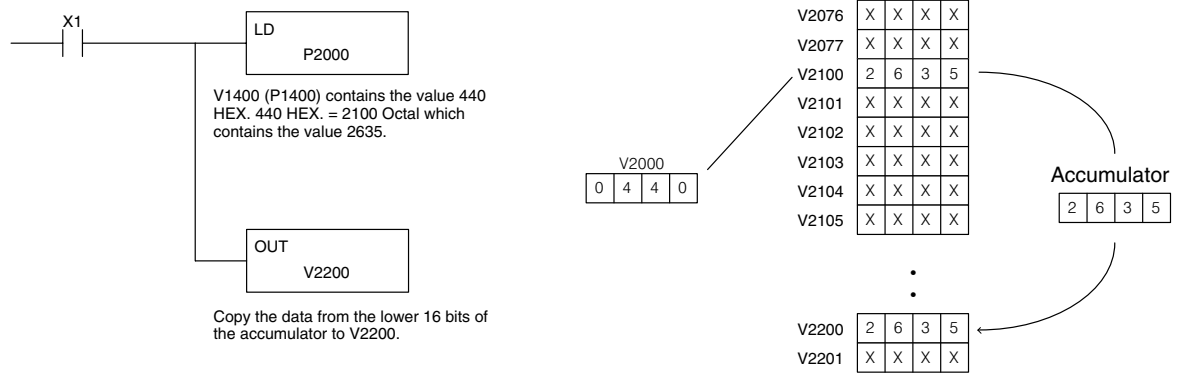
## Using Pointers

Many of the DL05 series instructions will allow V-memory pointers as a operand (commonly known as indirect addressing). Pointers allow instructions to obtain data from V-memory locations referenced by the pointer value.

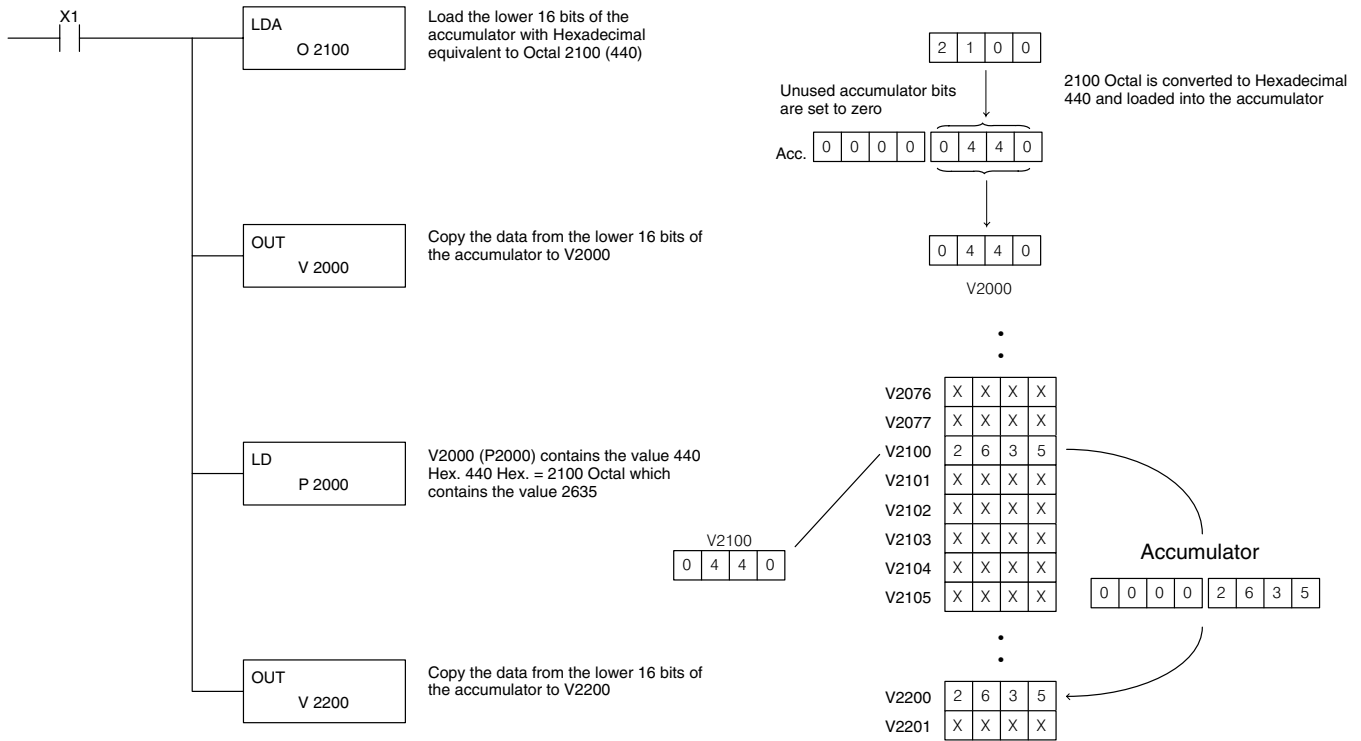


**NOTE:** DL05 V-memory addressing is in octal. However, the pointers reference a V-memory location with values viewed as HEX. Use the Load Address (LDA) instruction to move an address into the pointer location. This instruction performs the Octal to Hexadecimal conversion automatically.

In the following simple example we are using a pointer operand in a Load instruction. V-memory location 2000 is being used as the pointer location. V2000 contains the value 440 which the CPU views as the Hex equivalent of the Octal address V-memory location V2100. The CPU will copy the data from V2100 which in this example contains the value 2635 into the lower word of the accumulator.

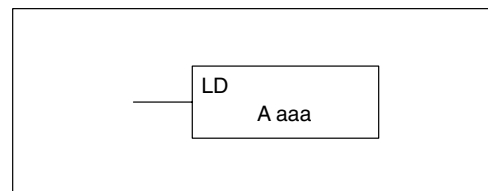


The following example is identical to the one above with one exception. The LDA (Load Address) instruction automatically converts the Octal address to Hex.



**Load  
(LD)**

The Load instruction is a 16 bit instruction that loads the value (Aaaa), which is either a V memory location or a 4 digit constant, into the lower 16 bits of the accumulator. The upper 16 bits of the accumulator are set to 0.



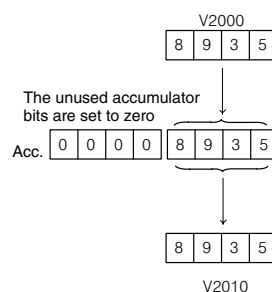
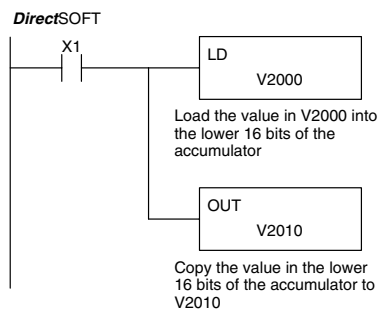
Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P
Constant	K

Discrete Bit Flags	Description
SP53	on when the pointer is outside of the available range.
SP70	on when the value loaded into the accumulator by any instruction is zero.
SP76	on when the result in the accumulator is negative.



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

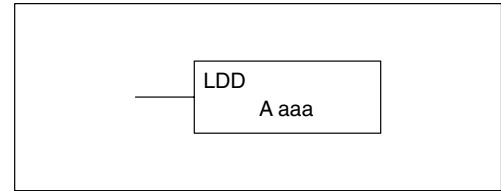
In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator and output to V2010.

**Handheld Programmer Keystrokes**

\$ STR	→	B 1	X SET
SHFT	L ANDST	D 3	→
C 2	A 0	A 0	A 0
GX OUT	→	SHFT	V AND
		C 2	A 0
		B 1	A 0
			ENT

**Load Double  
(LDD )**

The Load Double instruction is a 32 bit instruction that loads the value (Aaaa), which is either two consecutive V memory locations or an 8 digit constant value, into the accumulator.



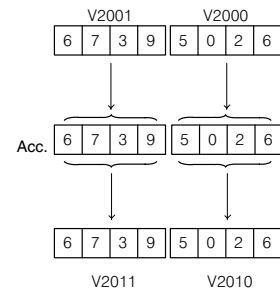
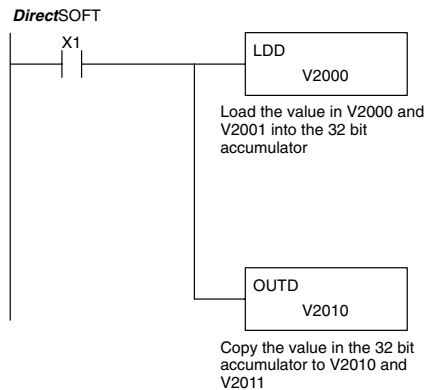
Operand Data Type		DL05 Range
A		aaa
V memory	V	All (See page 4-28)
Pointer	P	All V mem. (See page 4-28)
Constant	K	0-FFFFFFFF

Discrete Bit Flags	Description
SP53	on when the pointer is outside of the available range.
SP70	on when the value loaded into the accumulator by any instruction is zero.
SP76	on when the value loaded into the accumulator by any instruction is zero.



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

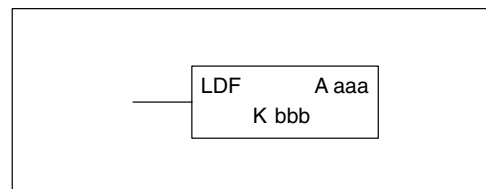
In the following example, when X1 is on, the 32 bit value in V2000 and V2001 will be loaded into the accumulator and output to V2010 and V2011.

**Handheld Programmer Keystrokes**

\$ STR	→	B 1	ENT
SHFT	L ANDST	D 3	D 3 →
C 2	A 0	A 0	A 0 ENT
GX OUT	SHFT	D 3	→
C 2	A 0	B 1	A 0 ENT

## Load Formatted (LDF)

The Load Formatted instruction loads 1–32 consecutive bits from discrete memory locations into the accumulator. The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be loaded. Unused accumulator bit locations are set to zero.



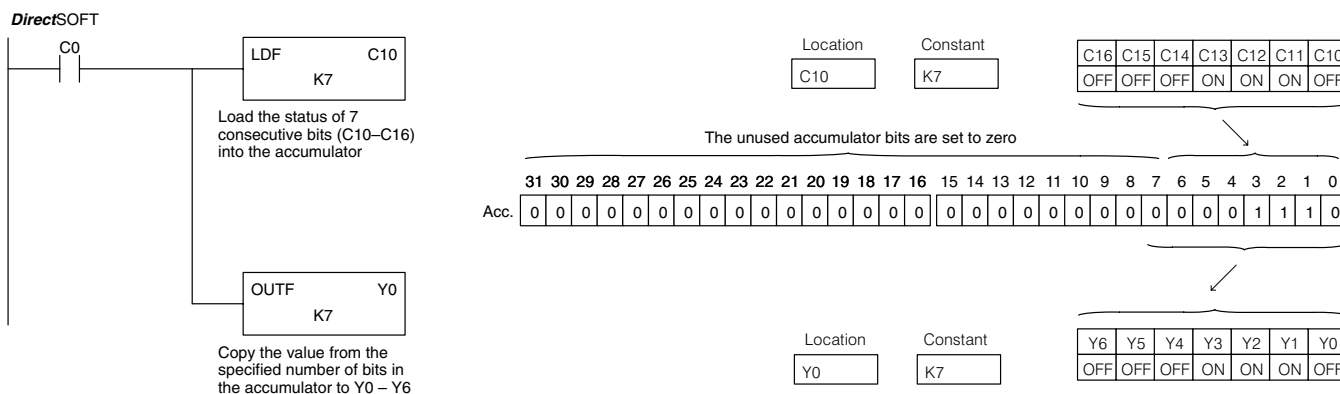
Operand Data Type		DL05 Range	
	A	aaa	bbb
Inputs	X	0–377	—
Outputs	Y	0–377	—
Control Relays	C	0–777	—
Stage Bits	S	0–377	—
Timer Bits	T	0–177	—
Counter Bits	CT	0–177	—
Special Relays	SP	0–777	—
Constant	K	—	1–32

Discrete Bit Flags	Description
SP70	on when the value loaded into the accumulator by any instruction is zero.
SP76	on when the value loaded into the accumulator by any instruction is zero.



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

In the following example, when C0 is on, the binary pattern of C10–C16 (7 bits) will be loaded into the accumulator using the Load Formatted instruction. The lower 7 bits of the accumulator are output to Y0–Y6 using the Out Formatted instruction.



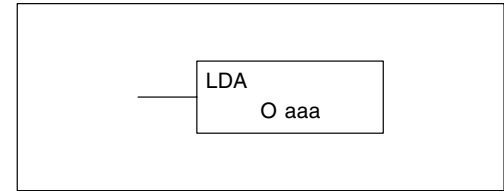
### Handheld Programmer Keystrokes

\$	→	SHFT	C	A	ENT
STR			2	0	
SHFT	L	D	F	→	
ANDST		3	5		
SHFT	C	B	A	→	H
	2	1	0		7 ENT
GX	SHFT	F	→		
OUT		5			
A	→	H	ENT		
0		7			



## Load Address (LDA)

The Load Address instruction is a 16 bit instruction. It converts any octal value or address to the HEX equivalent value and loads the HEX value into the accumulator. This instruction is useful when an address parameter is required since all addresses for the DL05 system are in octal.



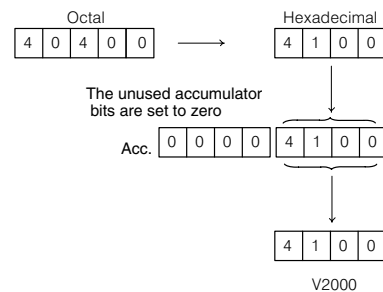
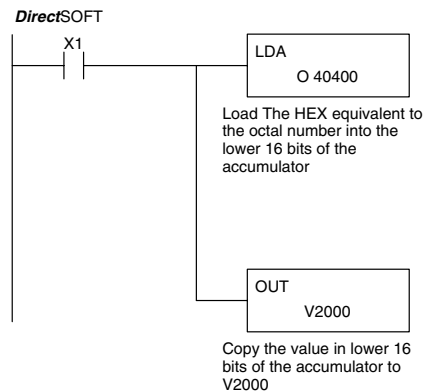
Operand Data Type	DL05 Range
	aaa
Octal Address      O	All V mem. (See page 4–28)

Discrete Bit Flags	Description
SP70	on when the value loaded into the accumulator by any instruction is zero.
SP76	on when the value loaded into the accumulator by any instruction is zero.

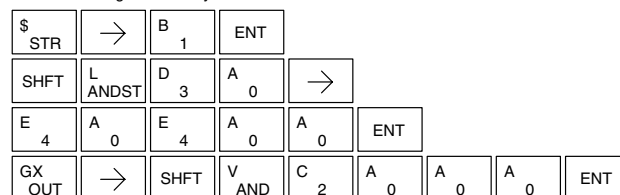


**NOTE:** Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.

In the following example when X1 is on, the octal number 40400 will be converted to a HEX 4100 and loaded into the accumulator using the Load Address instruction. The value in the lower 16 bits of the accumulator is copied to V2000 using the Out instruction.

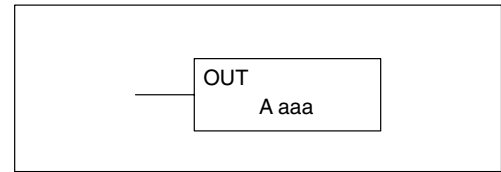


## Handheld Programmer Keystrokes



**Out  
(OUT)**

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

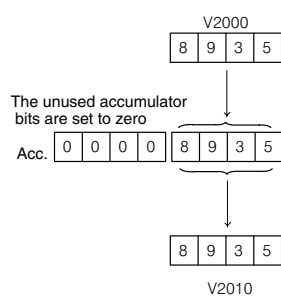
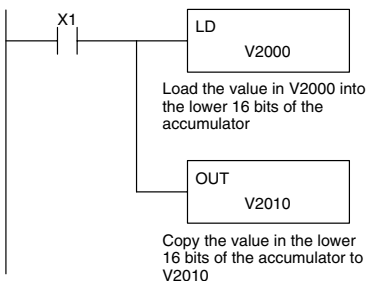


Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P
	All V mem. (See page 4-28)

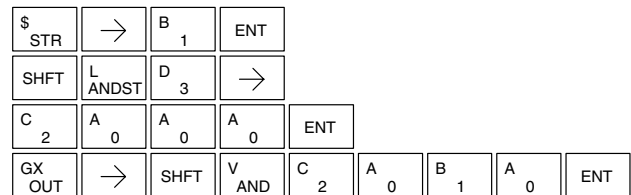
Discrete Bit Flags	Description
SP53	on when the pointer is outside of the available range.

In the following example, when X1 is on, the value in V2000 will be loaded into the lower 16 bits of the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator are copied to V2010 using the Out instruction.

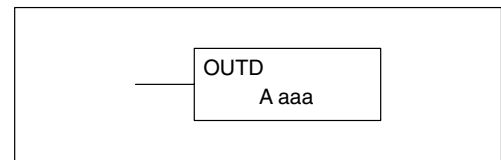
DirectSOFT



Handheld Programmer Keystrokes

**Out Double  
(OUTD)**

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

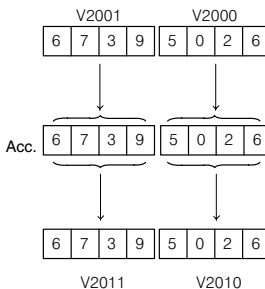
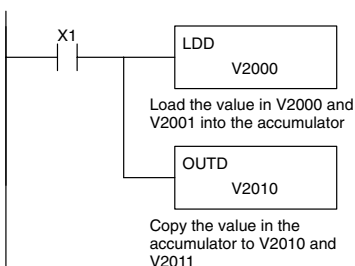


Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P
	All V mem. (See page 4-28)

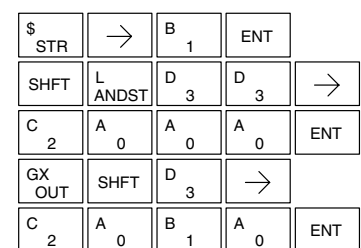
Discrete Bit Flags	Description
SP53	on when the pointer is outside of the available range.

In the following example, when X1 is on, the 32 bit value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is output to V2010 and V2011 using the Out Double instruction.

DirectSOFT

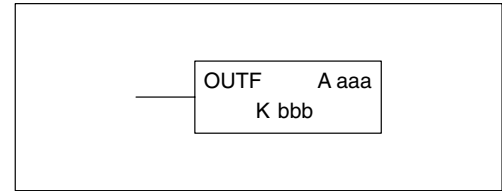


Handheld Programmer Keystrokes



## Out Formatted (OUTF)

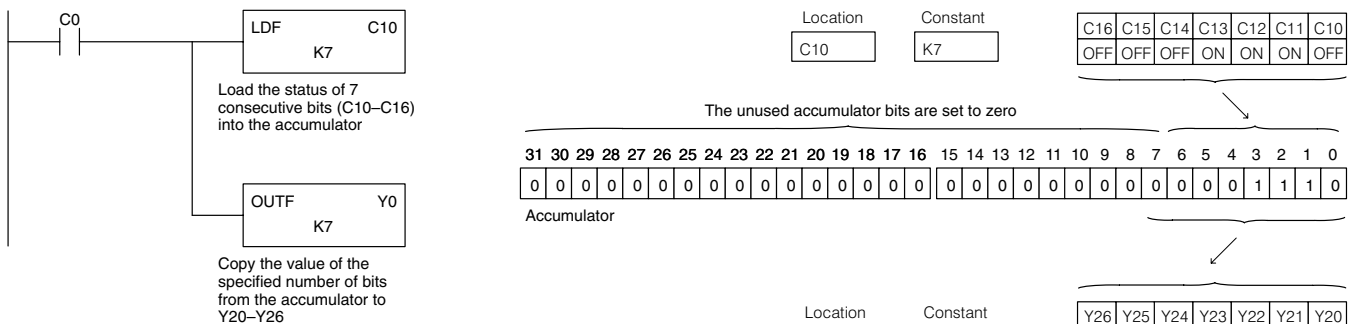
The Out Formatted instruction outputs 1–32 bits from the accumulator to the specified discrete memory locations. The instruction requires a starting location (Aaaa) for the destination and the number of bits (Kbbb) to be output.



Operand Data Type	A	DL05 Range	
		aaa	bbb
Inputs	X	0–377	—
Outputs	Y	0–377	—
Control Relays	C	0–777	—
Constant	K	—	1–32

In the following example, when C0 is on, the binary pattern of C10–C16 (7 bits) will be loaded into the accumulator using the Load Formatted instruction. The lower 7 bits of the accumulator are output to Y0–Y6 using the Out Formatted instruction.

DirectSOFT

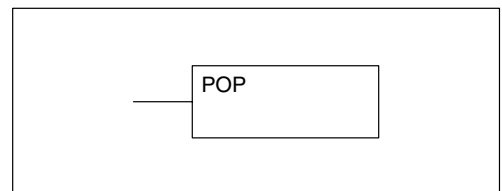


Handheld Programmer Keystrokes

\$	→	SHFT	C	A	ENT
STR			2	0	
SHFT	L	D	F	→	
	ANDST	3	5		
SHFT	C	B	A	→	H
	2	1	0		7
					ENT
GX	SHFT	F	→		
OUT		5			
A	→	H	ENT		
0		7			

## Pop (POP)

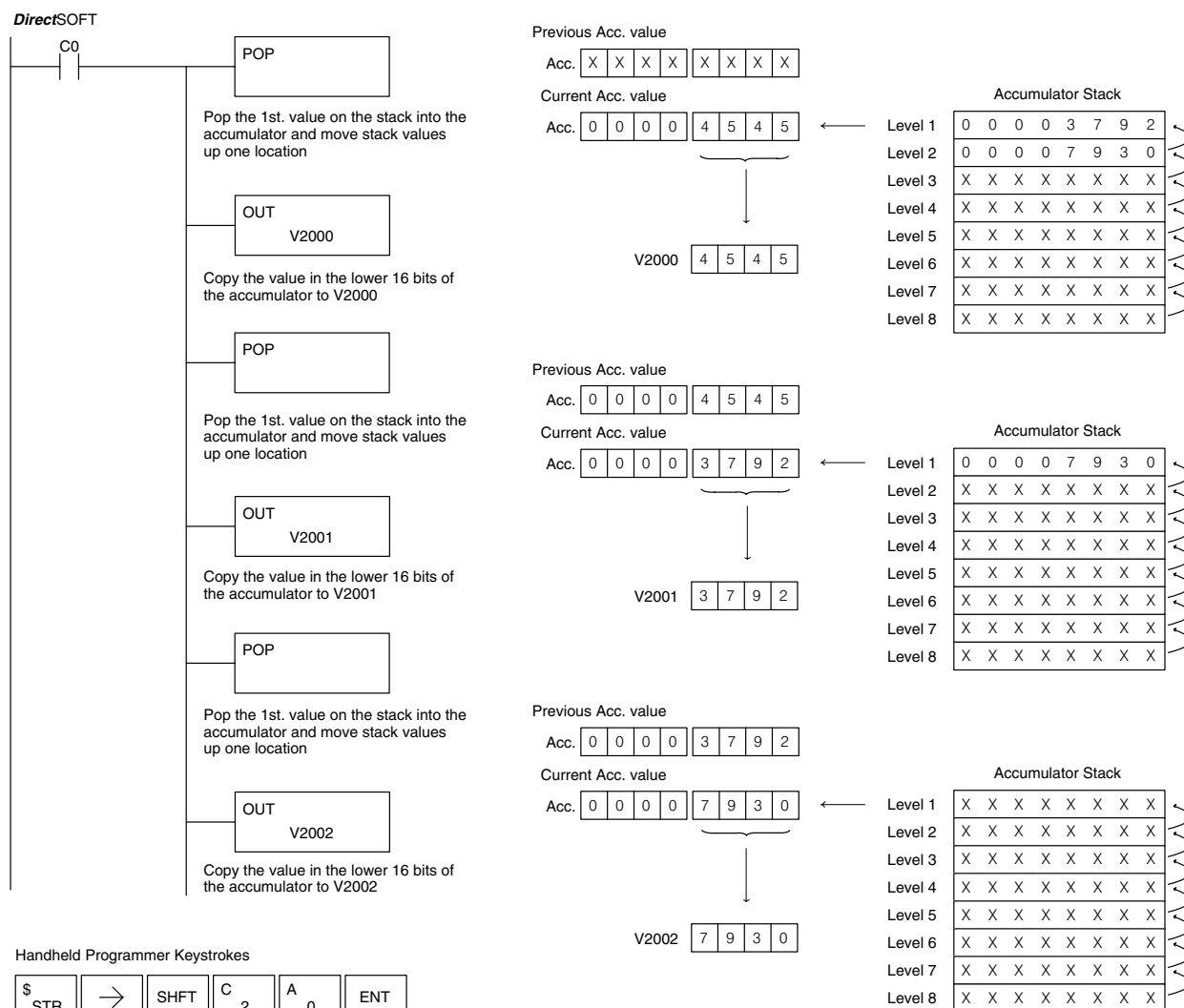
The Pop instruction moves the value from the first level of the accumulator stack (32 bits) to the accumulator and shifts each value in the stack up one level.



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

## Pop Instruction Continued

In the example below, when C0 is on, the value 4545 that was on top of the stack is moved into the accumulator using the Pop instruction. The value is output to V2000 using the Out instruction. The next Pop moves the value 3792 into the accumulator and outputs the value to V2001. The last Pop moves the value 7930 into the accumulator and outputs the value to V2002. Please note if the value in the stack were greater than 16 bits (4 digits) the Out Double instruction would be used and 2 V memory locations for each Out Double must be allocated.



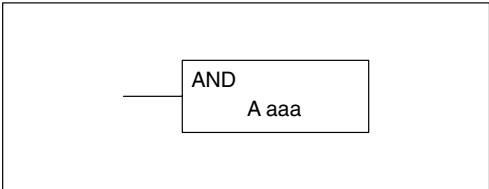
### Handheld Programmer Keystrokes

\$ STR	→	SHFT	C <sub>2</sub>	A <sub>0</sub>	ENT				
SHFT	P <sub>CV</sub>	SHFT	O <sub>INST#</sub>	P <sub>CV</sub>	ENT				
GX OUT	→	SHFT	V <sub>AND</sub>	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT	
SHFT	P <sub>CV</sub>	SHFT	O <sub>INST#</sub>	P <sub>CV</sub>	ENT				
GX OUT	→	SHFT	V <sub>AND</sub>	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	B <sub>1</sub>	ENT	
SHFT	P <sub>CV</sub>	SHFT	O <sub>INST#</sub>	P <sub>CV</sub>	ENT				
GX OUT	→	SHFT	V <sub>AND</sub>	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	C <sub>2</sub>	ENT	

# Logical Instructions (Accumulator)

## And (AND)

The And instruction is a 16 bit instruction that logically ands the value in the lower 16 bits of the accumulator with a specified V memory location (Aaaa). The result resides in the accumulator. The discrete status flag indicates if the result of the And is zero.



Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P
	All (See page 4-28)

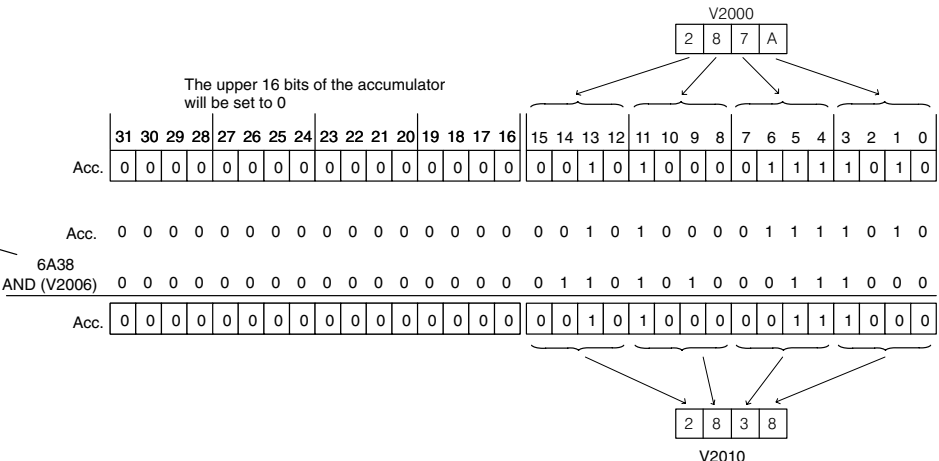
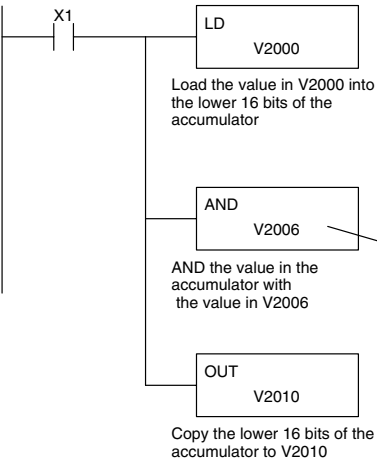
Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	on when the value loaded into the accumulator by any instruction is zero.



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator is anded with the value in V2006 using the And instruction. The value in the lower 16 bits of the accumulator is output to V2010 using the Out instruction.

DirectSOFT

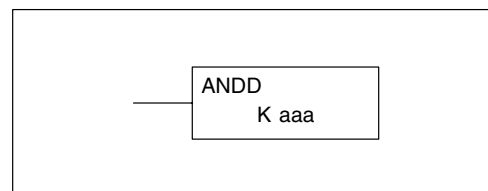


Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT	
V AND	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT	
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT	

### And Double (ANDD)

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



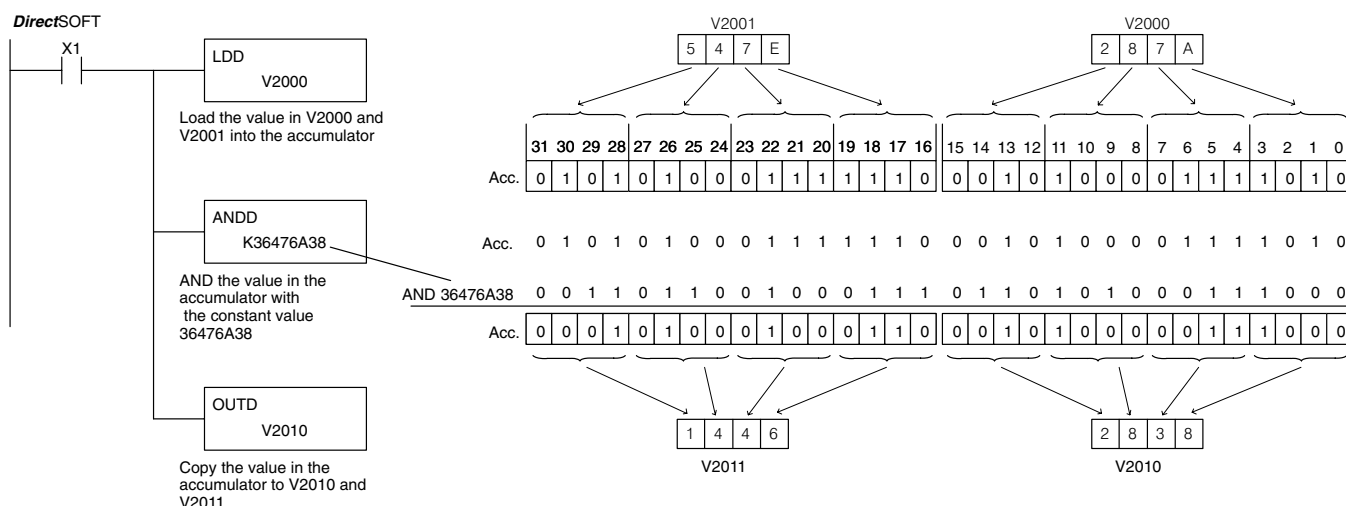
Operand Data Type		DL05 Range
		aaa
V memory	V	All (See page 4–28)
Pointer	P	All (See page 4–28)
Constant	K	0–FFFFFFFF

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



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is anded with 36476A38 using the And double instruction. The value in the accumulator is output to V2010 and V2011 using the Out Double instruction.

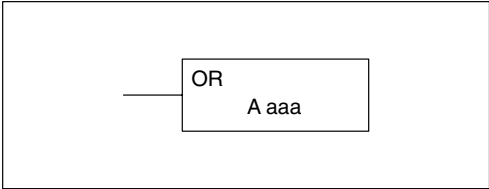


## Handheld Programmer Keystrokes

\$ STR	→	B <sub>1</sub>	ENT																	
SHFT	L ANDST	D <sub>3</sub>	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT											
V AND	SHFT	D <sub>3</sub>	→	SHFT	K JMP	D <sub>3</sub>	G <sub>6</sub>	E <sub>4</sub>	H <sub>7</sub>	G <sub>6</sub>	SHFT	A <sub>0</sub>	SHFT	D <sub>3</sub>	I <sub>8</sub>	ENT				
GX OUT	SHFT	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	B <sub>1</sub>	A <sub>0</sub>	ENT												

# Or (OR)

The Or instruction is a 16 bit instruction that logically ors the value in the lower 16 bits of the accumulator with a specified V memory location (Aaaa). The result resides in the accumulator. The discrete status flag indicates if the result of the Or is zero.



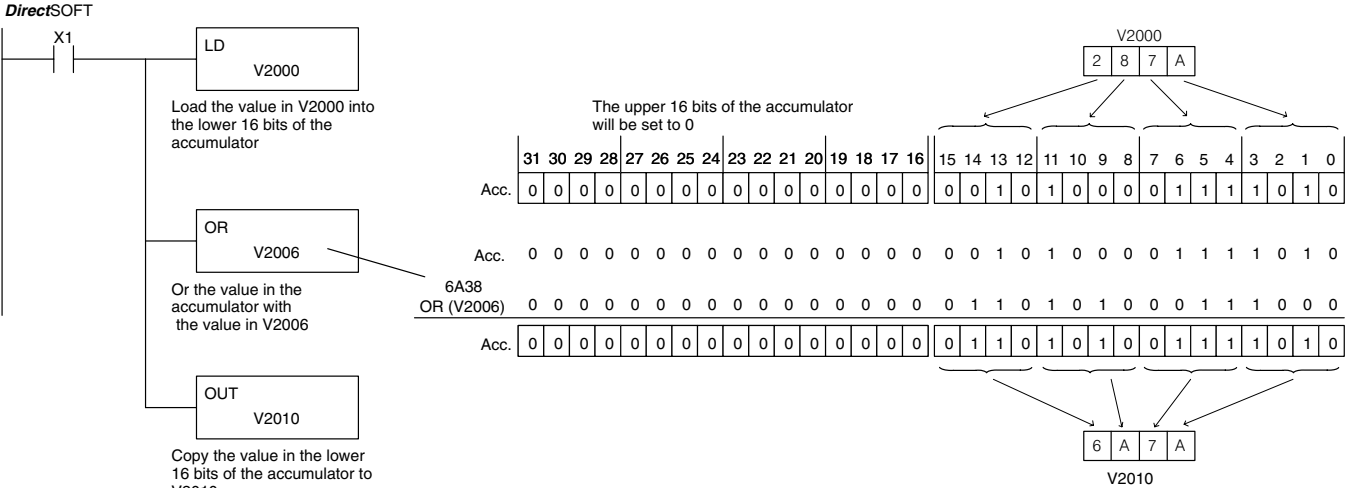
Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P

Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	on when the value loaded into the accumulator by any instruction is zero.



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator is ored with V2006 using the Or instruction. The value in the lower 16 bits of the accumulator are output to V2010 using the Out instruction.

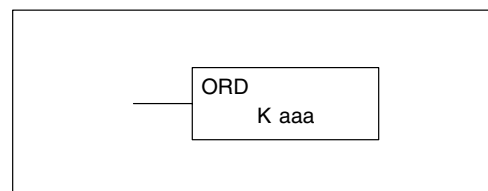


## Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT	
Q OR	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT	
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT	

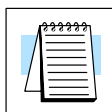
**Or Double  
(ORD)**

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



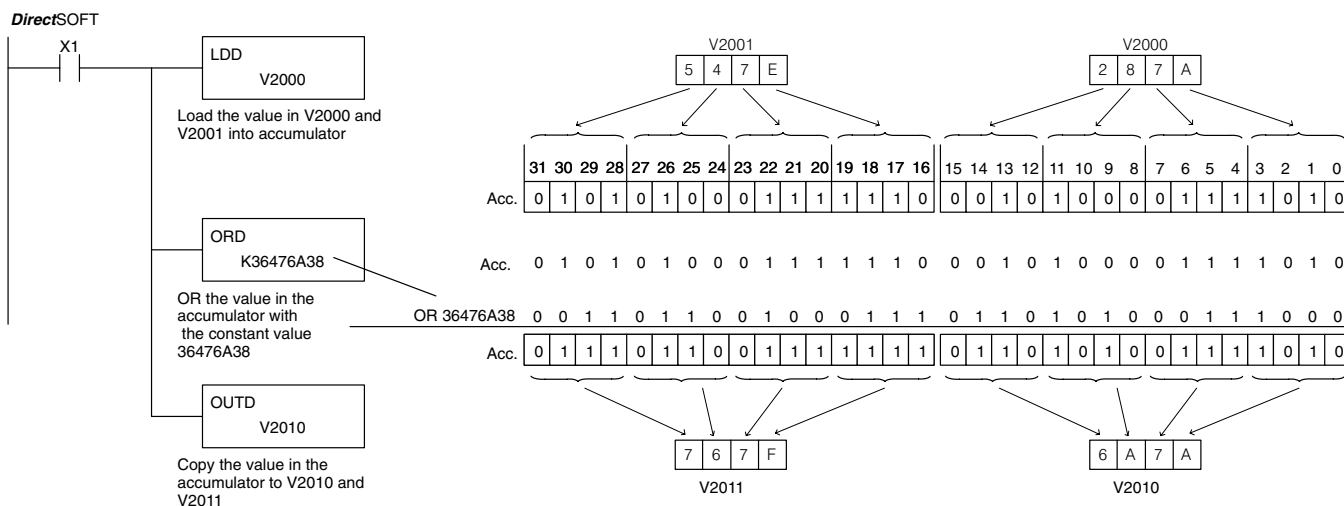
Operand Data Type		DL05 Range
		aaa
V memory	V	All (See page 4-28)
Pointer	P	All (See page 4-28)
Constant	K	0-FFFFFFFF

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



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is ored with 36476A38 using the Or Double instruction. The value in the accumulator is output to V2010 and V2011 using the Out Double instruction.



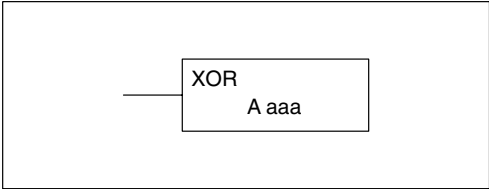
## Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT																							
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT																	
Q OR	SHFT	D 3	→	SHFT	K JMP	D 3	G 6	E 4	H 7	G 6	SHFT	A 0	SHFT	D 3	I 8	ENT										
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT																		



# Exclusive Or (XOR)

The Exclusive Or instruction is a 16 bit instruction that performs an exclusive or of the value in the lower 16 bits of the accumulator and a specified V memory location (Aaaa). The result resides in the in the accumulator. The discrete status flag indicates if the result of the XOR is zero.



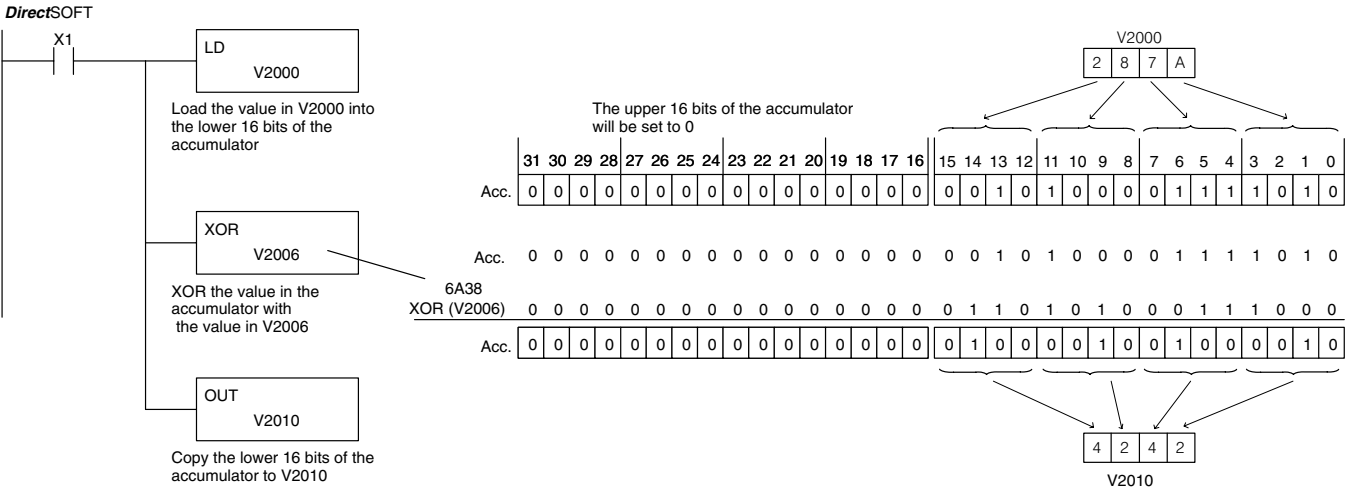
Operand Data Type	DL05 Range	
A	aaa	
V memory	V	All (See page 4–28)
Pointer	P	All (See page 4–28)

Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	on when the value loaded into the accumulator by any instruction is zero.



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator is exclusive ored with V2006 using the Exclusive Or instruction. The value in the lower 16 bits of the accumulator are output to V2010 using the Out instruction.

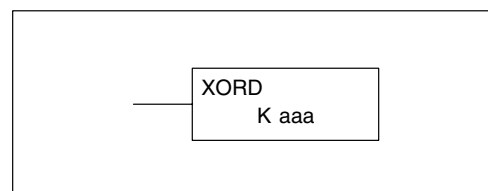


## Handheld Programmer Keystrokes

\$ STR	→	SHFT	X SET	B 1	ENT							
SHFT	L ANDST	D 3	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT		
SHFT	X SET	SHFT	Q OR	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT	
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT				

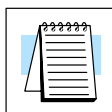
**Exclusive Or Double (XORD)**

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



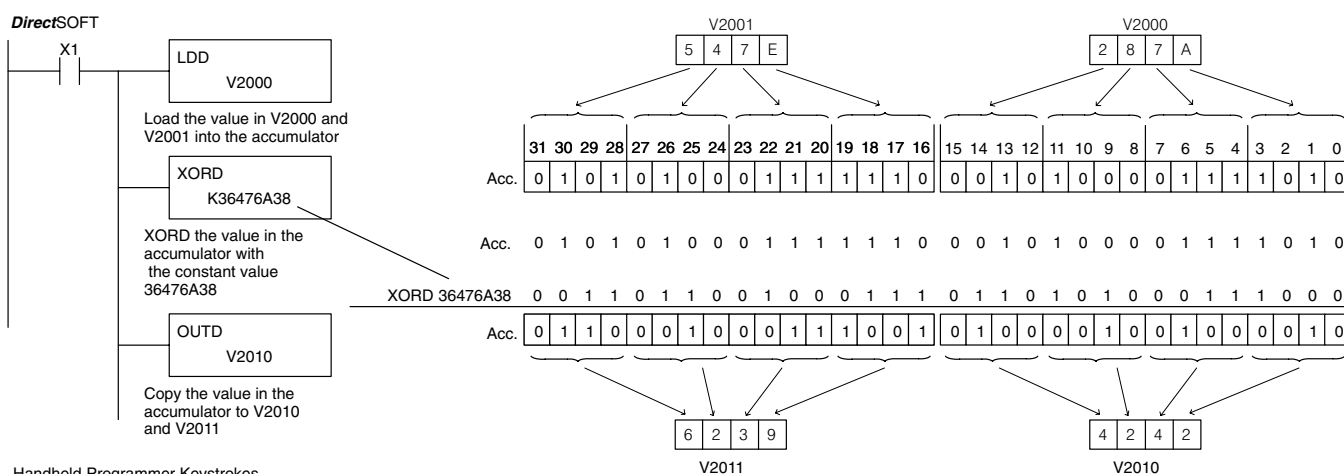
Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P
Constant	K
	0-FFFFFFF

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



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is exclusively ored with 36476A38 using the Exclusive Or Double instruction. The value in the accumulator is output to V2010 and V2011 using the Out Double instruction.



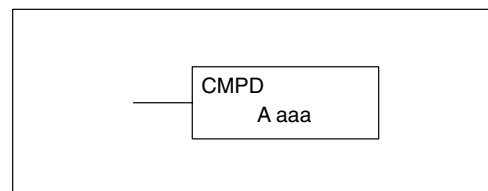
## Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT
SHFT	L ANDST	D 3	D 3
SHFT	X SET	Q OR	SHFT D 3
D 3	G 6	E 4	H 7
GX OUT	SHFT D 3	→	C 2
			A 0
			B 1
			A 0
			ENT



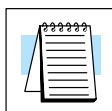
## Compare Double (CMPD)

The Compare Double instruction is a 32-bit instruction that compares the value in the accumulator with the value (Aaaa), which is either two consecutive V memory locations or an 8-digit (max.) constant. The corresponding status flag will be turned on indicating the result of the comparison.



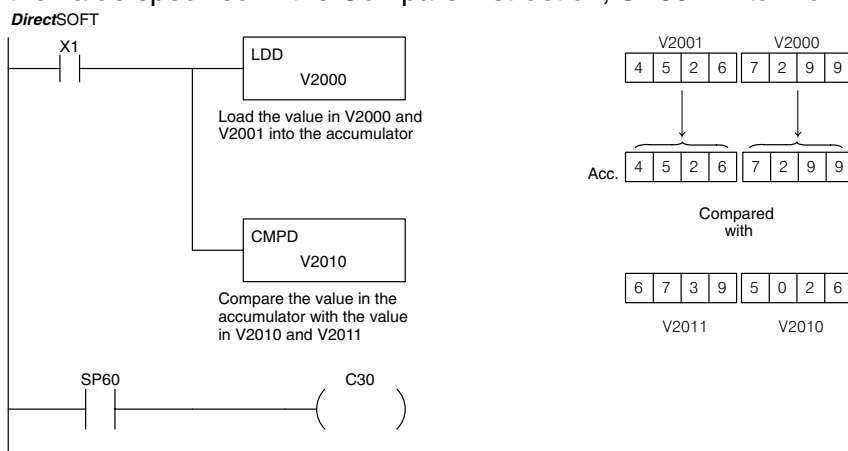
Operand Data Type	DL05 Range
A	aaa
V memory	V All (See page 4–28)
Pointer	P All (See page 4–28)
Constant	K 0–FFFFFFF

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



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is compared with the value in V2010 and V2011 using the CMPD instruction. The corresponding discrete status flag will be turned on indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the Compare instruction, SP60 will turn on energizing C30.



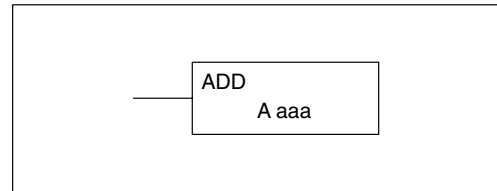
## Handheld Programmer Keystrokes

\$ STR	→	B <sub>1</sub>	ENT										
SHFT	L ANDST	D <sub>3</sub>	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT				
SHFT	C <sub>2</sub>	SHFT	M ORST	P CV	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	B <sub>1</sub>	A <sub>0</sub>	ENT		
\$ STR	→	SHFT	SP STRN	G <sub>6</sub>	A <sub>0</sub>	ENT							
GX OUT	→	SHFT	C <sub>2</sub>	D <sub>3</sub>	A <sub>0</sub>	ENT							

## Math Instructions

### Add (ADD)

Add is a 16 bit instruction that adds a BCD value in the accumulator with a BCD value in a V memory location (Aaaa). The result resides in the accumulator.



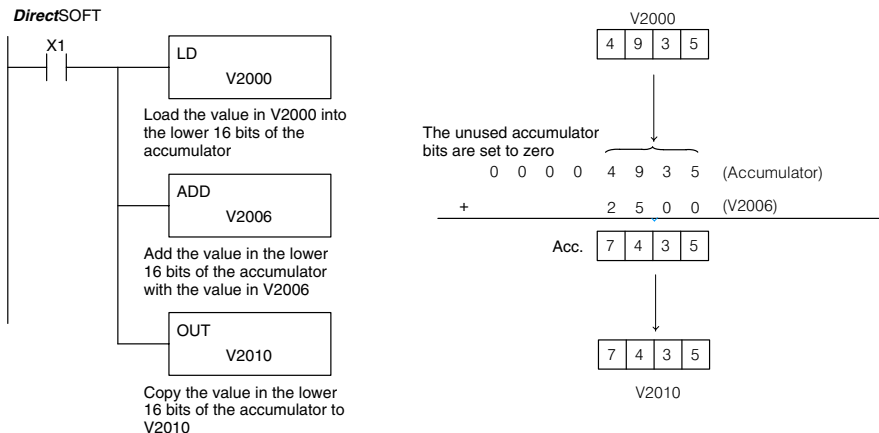
Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16 bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator are added to the value in V2006 using the Add instruction. The value in the accumulator is copied to V2010 using the Out instruction.

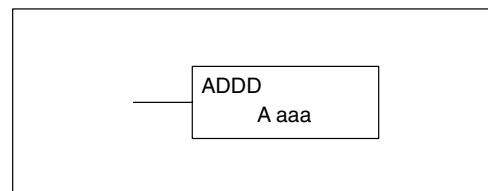


Handheld Programmer Keystrokes

\$ STR	→	B <sub>1</sub>	ENT										
SHFT	L ANDST	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT					
SHFT	A <sub>0</sub>	D <sub>3</sub>	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	G <sub>6</sub>	ENT				
GX OUT	→	SHFT	V AND	C <sub>2</sub>	A <sub>0</sub>	B <sub>1</sub>	A <sub>0</sub>	ENT					

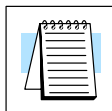
**Add Double  
(ADDD)**

Add Double is a 32 bit instruction that adds the BCD value in the accumulator with a BCD value (Aaaa), which is either two consecutive V memory locations or an 8-digit (max.) BCD constant. The result resides in the accumulator.



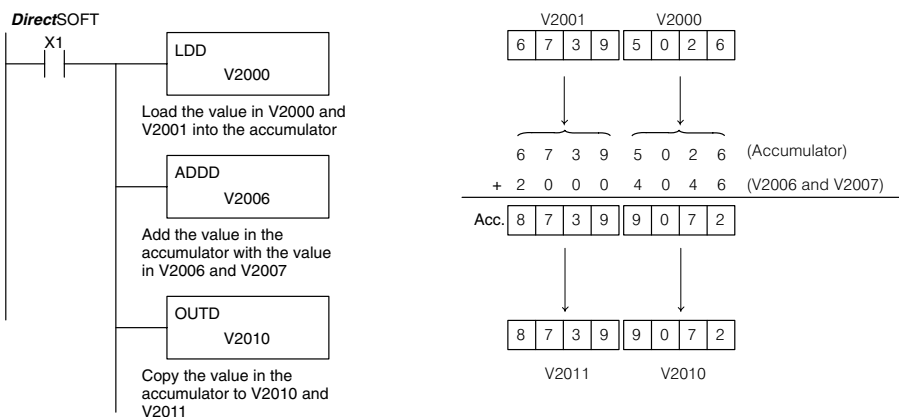
Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P
Constant	K

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16 bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is added with the value in V2006 and V2007 using the Add Double instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

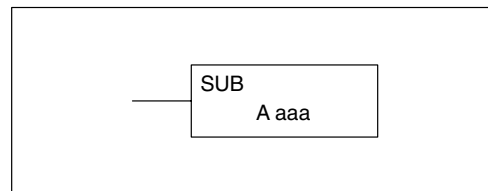


Handheld Programmer Keystrokes

\$ STR	→	B <sub>1</sub>	ENT										
SHFT	L ANDST	D <sub>3</sub>	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT				
SHFT	A <sub>0</sub>	D <sub>3</sub>	D <sub>3</sub>	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	G <sub>6</sub>	ENT			
GX OUT	SHFT	D <sub>3</sub>	→	SHFT	V AND	C <sub>2</sub>	A <sub>0</sub>	B <sub>1</sub>	A <sub>0</sub>	ENT			

## Subtract (SUB)

Subtract is a 16 bit instruction that subtracts the BCD value (Aaaa) in a V memory location from the BCD value in the lower 16 bits of the accumulator. The result resides in the accumulator.



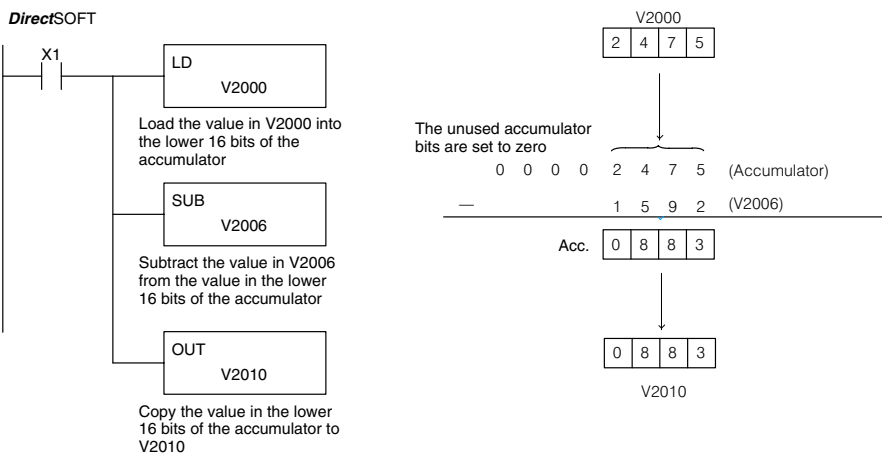
Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P

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



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in V2006 is subtracted from the value in the accumulator using the Subtract instruction. The value in the accumulator is copied to V2010 using the Out instruction.

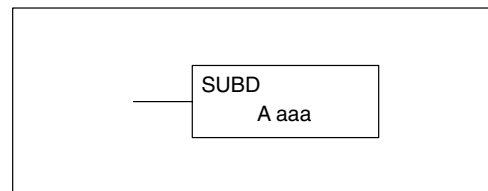


### Handheld Programmer Keystrokes

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

## Subtract Double (SUBD)

Subtract Double is a 32 bit instruction that subtracts the BCD value (Aaaa), which is either two consecutive V memory locations or an 8-digit (max.) constant, from the BCD value in the accumulator. The result resides in the accumulator.



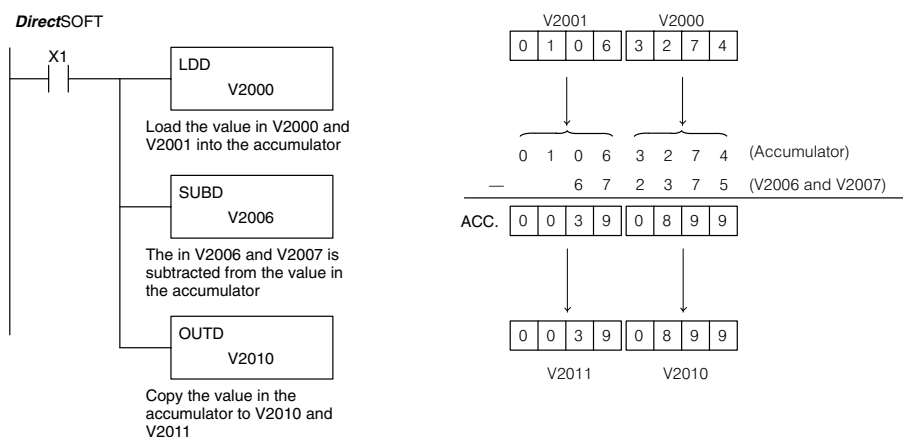
Operand Data Type	DL05 Range
A	aaa
V memory	V All (See page 4–28)
Pointer	P All (See page 4–28)
Constant	K 0–99999999

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



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in V2006 and V2007 is subtracted from the value in the accumulator. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



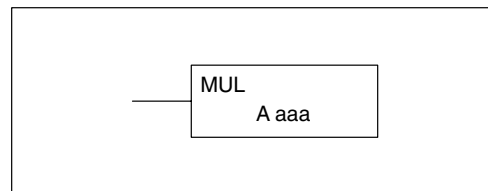
## Handheld Programmer Keystrokes

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



## Multiply (MUL)

Multiply is a 16 bit instruction that multiplies the BCD value (Aaaa), which is either a V memory location or a 4-digit (max.) constant, by the BCD value in the lower 16 bits of the accumulator. The result can be up to 8 digits and resides in the accumulator.



Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P
Constant	K
	0-9999

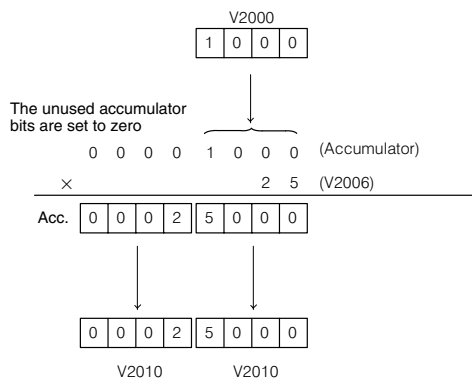
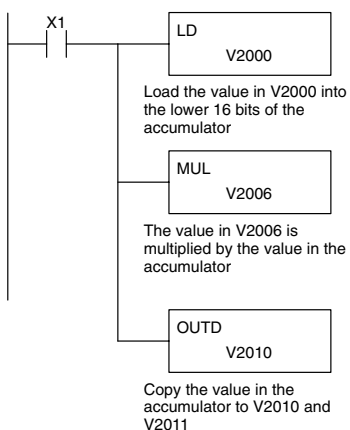
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in V2006 is multiplied by the value in the accumulator. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

### DirectSOFT

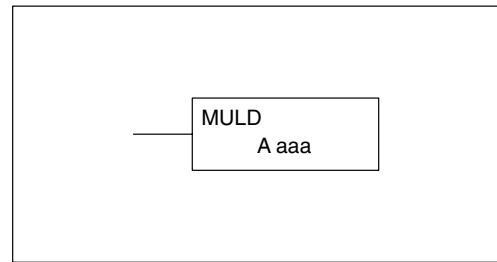


### Handheld Programmer Keystrokes

\$ STR	→	B <sub>1</sub>	ENT												
SHFT	L ANDST	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT							
SHFT	M ORST	U ISG	L ANDST	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	G <sub>6</sub>	ENT						
GX OUT	SHFT	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	B <sub>1</sub>	A <sub>0</sub>	ENT							

### Multiply Double (MULD)

Multiply Double is a 32 bit instruction that multiplies the 8-digit BCD value in the accumulator by the 8-digit BCD value in the two consecutive V-memory locations specified in the instruction. The lower 8 digits of the results reside in the accumulator. Upper digits of the result reside in the accumulator stack.



Operand Data Type	DL05 Range
A	aaa
Vmemory	V
Pointer	P
	All (See page 4-28)

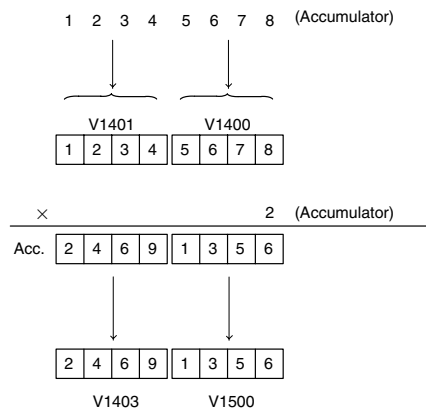
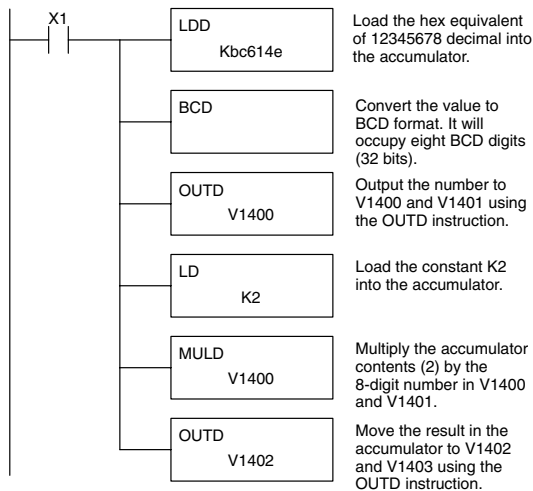
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.



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

In the following example, when X1 is on, the constant Kbc614e hex will be loaded into the accumulator. When converted to BCD the number is "12345678". That number is stored in V1400 and V1401. After loading the constant K2 into the accumulator, we multiply it times 12345678, which is 24691356.

#### DirectSOFT Display

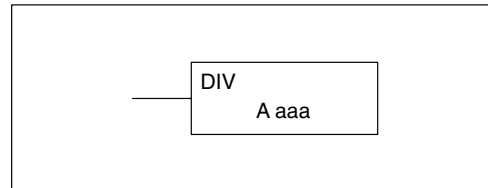


#### Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT	SHFT	L ANDST	D 3	D 3	→	SHFT	K JMP	SHFT
B 1	C 2	SHFT	G 6	B 1	E 4	ENT	SHFT	B 1	C 2	D 3	ENT
GX OUT	SHFT	D 3	→	B 1	E 4	A 0	A 0	ENT			
SHFT	L ANDST	D 3	→	SHFT	K JMP	C 2	ENT				
SHFT	M ORST	U ISG	L ANDST	D 3	→	B 1	E 4	A 0	A 0		
GX OUT	SHFT	D 3	→	B 1	E 4	A 0	C 2	ENT			

## Divide (DIV)

Divide is a 16 bit instruction that divides the BCD value in the accumulator by a BCD value (Aaaa), which is either a V memory location or a 4-digit (max.) constant. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.



Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P
Constant	K
	1-9999

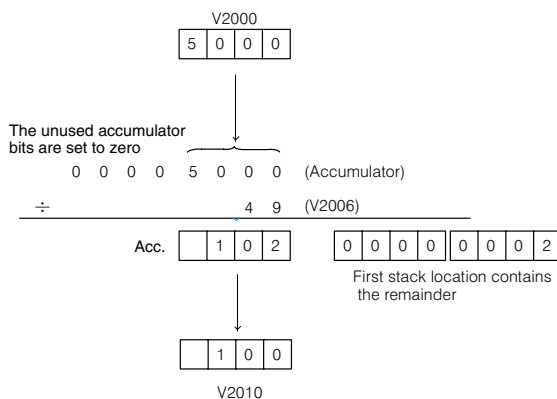
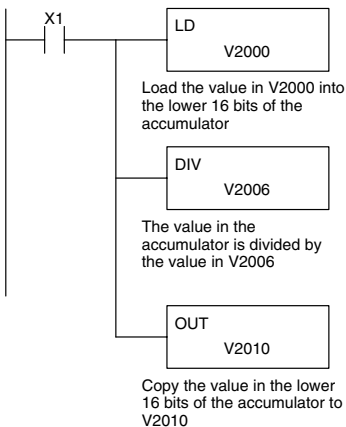
Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed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



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator will be divided by the value in V2006 using the Divide instruction. The value in the accumulator is copied to V2010 using the Out instruction.

### DirectSOFT

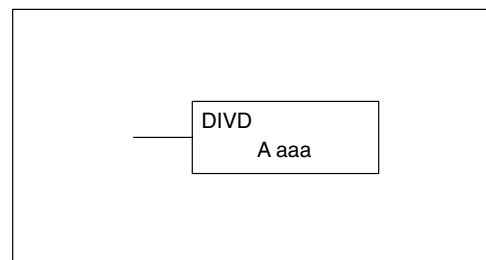


### Handheld Programmer Keystrokes

\$ STR	→	B <sub>1</sub>	ENT												
SHFT	L ANDST	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT							
SHFT	D <sub>3</sub>	I <sub>8</sub>	V AND	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	G <sub>6</sub>	ENT						
GX OUT	→	SHFT	V AND	C <sub>2</sub>	A <sub>0</sub>	B <sub>1</sub>	A <sub>0</sub>	ENT							

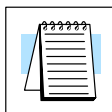
## Divide Double (DIVD)

Divide Double is a 32 bit instruction that divides the BCD value in the accumulator by a BCD value (Aaaa), which must be obtained from two consecutive V memory locations. (You cannot use a constant as the parameter in the box.) The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.



Operand Data Type	DL05 Range
A	aaa
Vmemory	V
Pointer	P

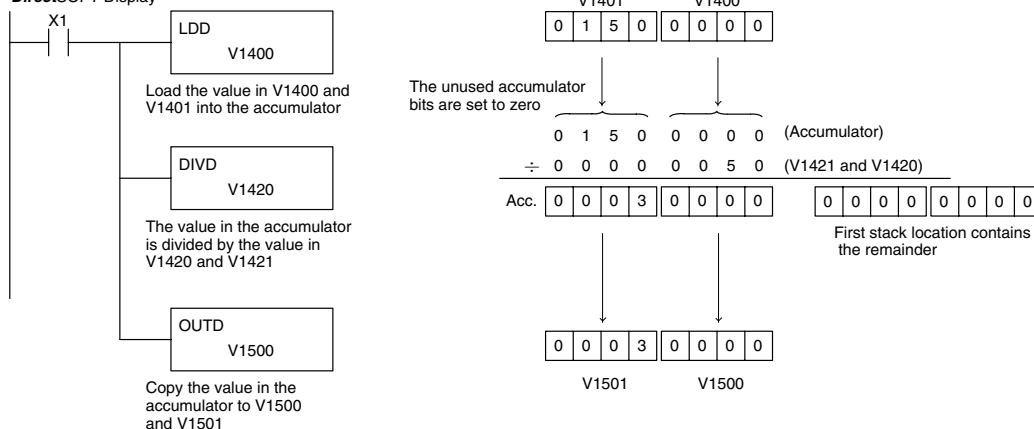
Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed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



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

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is divided by the value in V1420 and V1421 using the Divide Double instruction. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

DirectSOFT Display

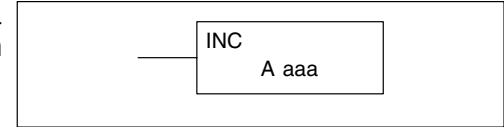


Handheld Programmer Keystrokes

\$	→	B	ENT	SHFT	L	D	D	→
STR		1			ANDST	3	3	
B	E	A	A	ENT	SHFT	D	I	V
1	4	0	0			3	8	AND
→	B	E	C	A	ENT	GX	SHFT	D
	1	4	2	0		OUT		3
→	B	F	A	A	ENT			
	1	5	0	0				

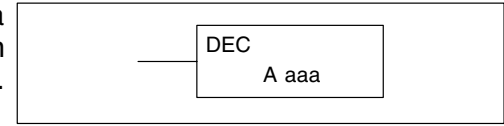
## Increment (INC)

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



## Decrement (DEC)

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



Operand Data Type	DL05 Range
A	aaa
Vmemory V	All (See p. 4-28)
Pointer P	All (See p. 4-28)

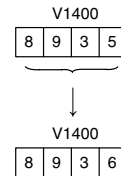
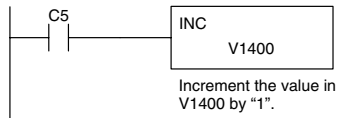
Discrete Bit Flags	Description
SP63	on when the result of the instruction causes the value in the accumulator to be zero.
SP75	on when a BCD instruction is executed and a NON-BCD number was encountered.



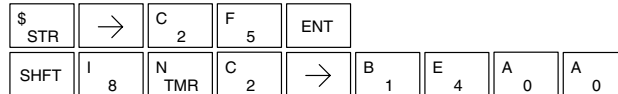
**NOTE:** Status flags are valid only until another instruction uses the same flag.

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

DirectSOFT Display

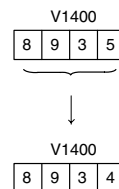
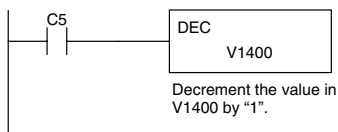


Handheld Programmer Keystrokes



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

DirectSOFT Display

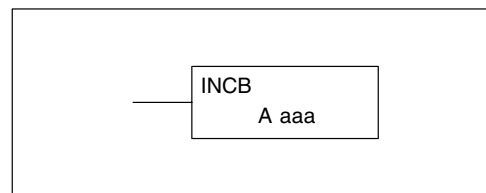


Handheld Programmer Keystrokes



**Increment Binary (INCB)**

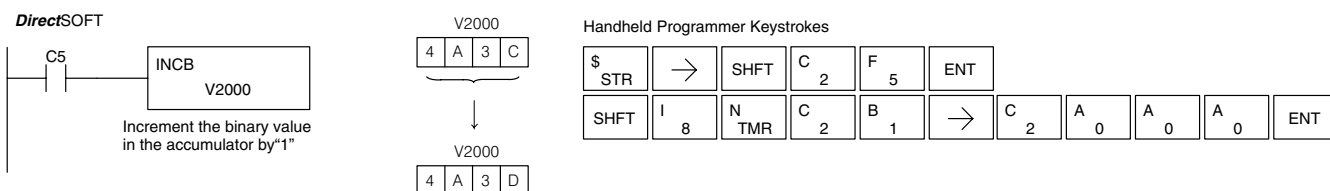
The Increment Binary instruction increments a binary value in a specified V memory location by “1” each time the instruction is executed.



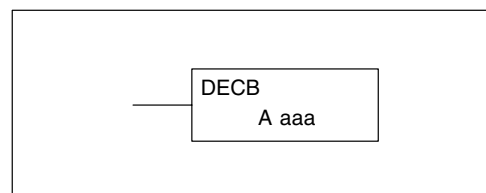
Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P
	All (See page 4-28)

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

In the following example when C5 is on, the binary value in V2000 is increased by 1.

**Decrement Binary (DECB)**

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



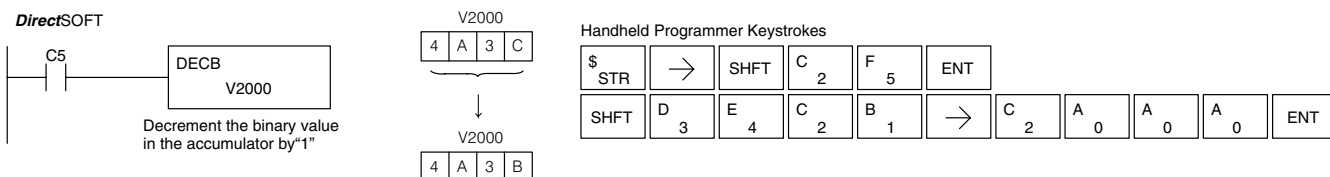
Operand Data Type	DL05 Range
A	aaa
V memory	V
Pointer	P
	All (See page 4-28)

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

**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

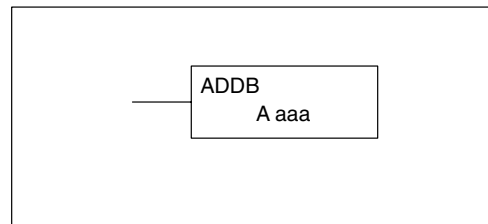


In the following example when C5 is on, the value in V2000 is decreased by 1.



## Add Binary (ADDB)

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



Operand Data Type	DL05 Range
A	aaa
Vmemory	V
Pointer	P
Constant	K
	0-FFFF

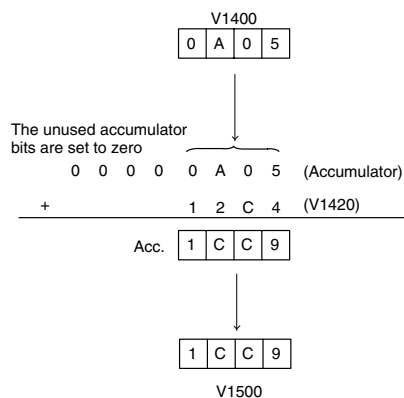
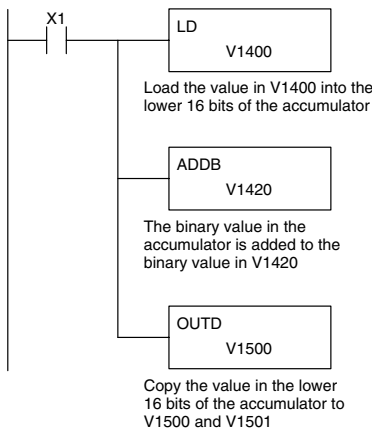
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16 bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in a incorrect sign bit.



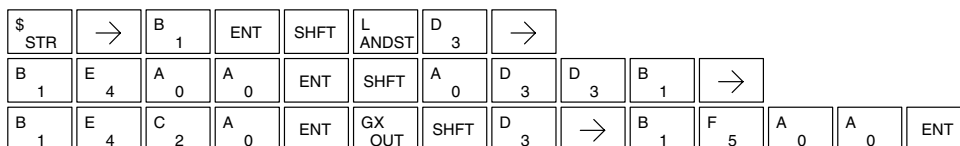
**NOTE:** Status flags are valid only until another instruction uses the same flag.

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in the accumulator will be added to the binary value in V1420 using the Add Binary instruction. The value in the accumulator is copied to V1500 and V1501 using the Out instruction.

DirectSOFT Display

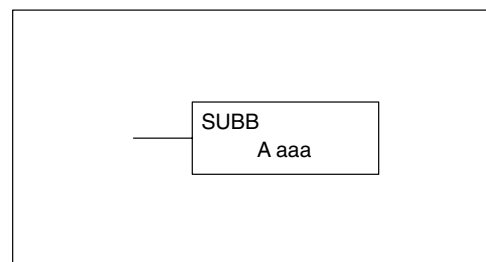


Handheld Programmer Keystrokes



**Subtract Binary (SUBB)**

Subtract Binary is a 16 bit instruction that subtracts the unsigned 2's complement binary value (Aaaa), which is either a V memory location or a 16-bit 2's complement binary value, from the binary value in the accumulator. The result resides in the accumulator.



Operand Data Type	DL05 Range
A	aaa
Vmemory	V
Pointer	P
Constant	K
	0-FFFF

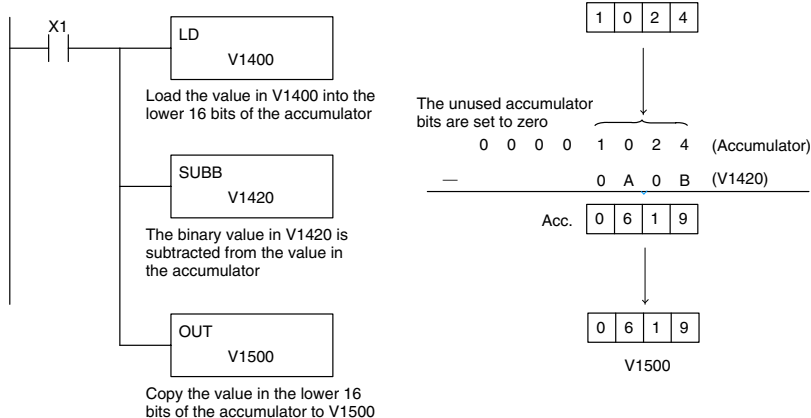
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16 bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.



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

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in V1420 is subtracted from the binary value in the accumulator using the Subtract Binary instruction. The value in the accumulator is copied to V1500 using the Out instruction.

DirectSOFT Display



Handheld Programmer Keystrokes

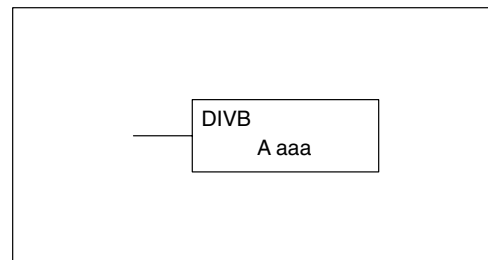
\$ STR	→	B <sub>1</sub>	ENT															
SHFT	L ANDST	D <sub>3</sub>	→	B <sub>1</sub>	E <sub>4</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT										
SHFT	S RST	SHFT	U ISG	B <sub>1</sub>	B <sub>1</sub>	→	B <sub>1</sub>	E <sub>4</sub>	C <sub>2</sub>	A <sub>0</sub>	ENT							
GX OUT	SHFT	D <sub>3</sub>	→	B <sub>1</sub>	F <sub>5</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT										





**Divide Binary  
(DIVB)**

Divide Binary is a 16 bit instruction that divides the unsigned 2's complement binary value in the accumulator by a binary value (Aaaa), which is either a V memory location or a 16-bit unsigned 2's complement binary constant. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.



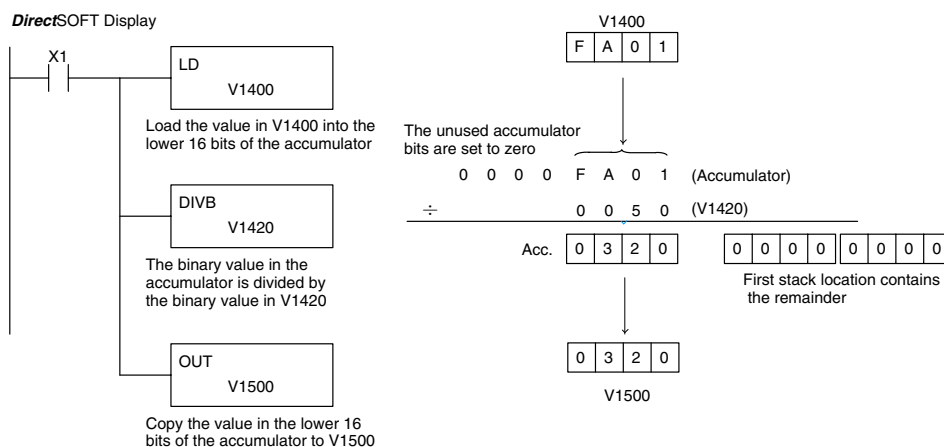
Operand Data Type	DL05 Range
A	aaa
Vmemory	V
Pointer	P
Constant	K
	0-FFFF

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



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

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in the accumulator is divided by the binary value in V1420 using the Divide Binary instruction. The value in the accumulator is copied to V1500 using the Out instruction.

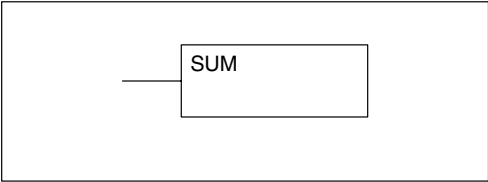
**Handheld Programmer Keystrokes**

Programmer's Keyboards																
\$ STR	→	B <sub>1</sub>	ENT													
SHFT	L ANDST	D <sub>3</sub>	→	B <sub>1</sub>	E <sub>4</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT								
SHFT	D <sub>3</sub>	I <sub>8</sub>	U ISG	B <sub>1</sub>	→	B <sub>1</sub>	E <sub>4</sub>	C <sub>2</sub>	A <sub>0</sub>	ENT						
GX OUT	SHFT	D <sub>3</sub>	→	B <sub>1</sub>	F <sub>5</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT								

# Bit Operation Instructions

## Sum (SUM)

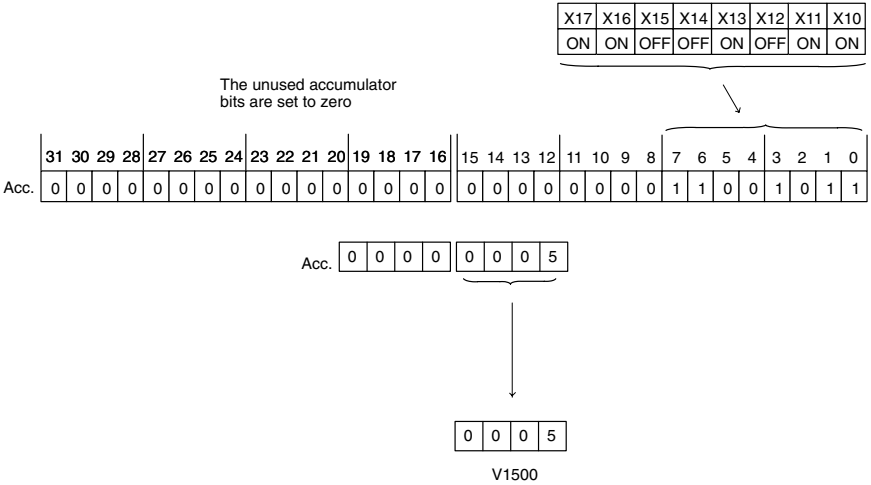
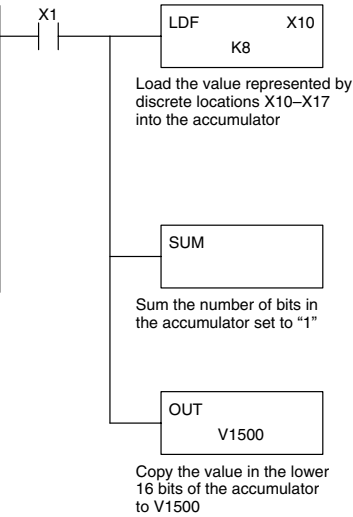
The Sum instruction counts number of bits that are set to “1” in the accumulator. The HEX result resides in the accumulator.



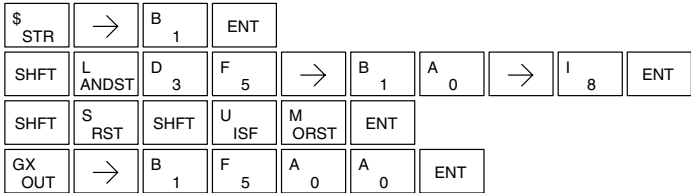
In the following example, when X1 is on, the value formed by discrete locations X10–X17 is loaded into the accumulator using the Load Formatted instruction. The number of bits in the accumulator set to “1” is counted using the Sum instruction. The value in the accumulator is copied to V1500 using the Out instruction.

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

DirectSOFT Display

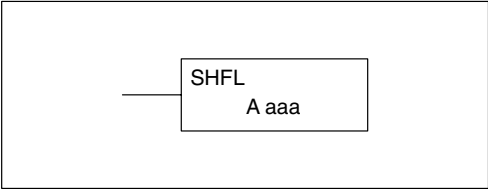


Handheld Programmer Keystrokes



## Shift Left (SHFL)

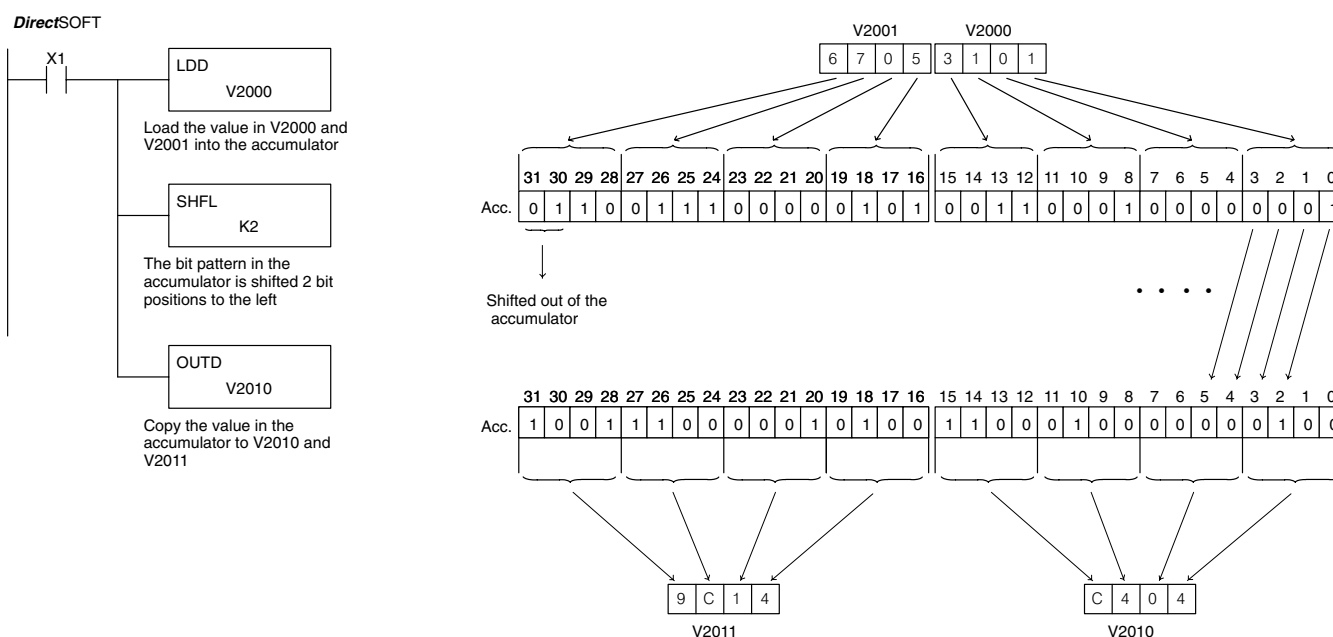
Shift Left is a 32 bit instruction that shifts the bits in the accumulator a specified number (Aaaa) of places to the left. The vacant positions are filled with zeros and the bits shifted out of the accumulator are discarded.



Operand Data Type	DL05 Range
A	aaa
V memory	V All (See page 4–28)
Constant	K 1–32

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 2 bits to the left using the Shift Left instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



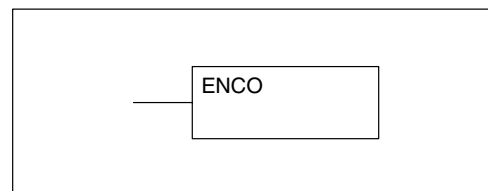
## Handheld Programmer Keystrokes

\$ STR	→	B <sub>1</sub>	ENT						
SHFT	L ANDST	D <sub>3</sub>	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT
SHFT	S RST	SHFT	H <sub>7</sub>	F <sub>5</sub>	L ANDST	→	C <sub>2</sub>	ENT	
GX OUT	SHFT	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	B <sub>1</sub>	A <sub>0</sub>	ENT	



**Encode  
(ENCO)**

The Encode instruction encodes the bit position in the accumulator having a value of 1, and returns the appropriate binary representation. If the most significant bit is set to 1 (Bit 31), the Encode instruction would place the value HEX 1F (decimal 31) in the accumulator. If the value to be encoded is 0000 or 0001, the instruction will place a zero in the accumulator. If the value to be encoded has more than one bit position set to a “1”, the least significant “1” will be encoded and SP53 will be set on.



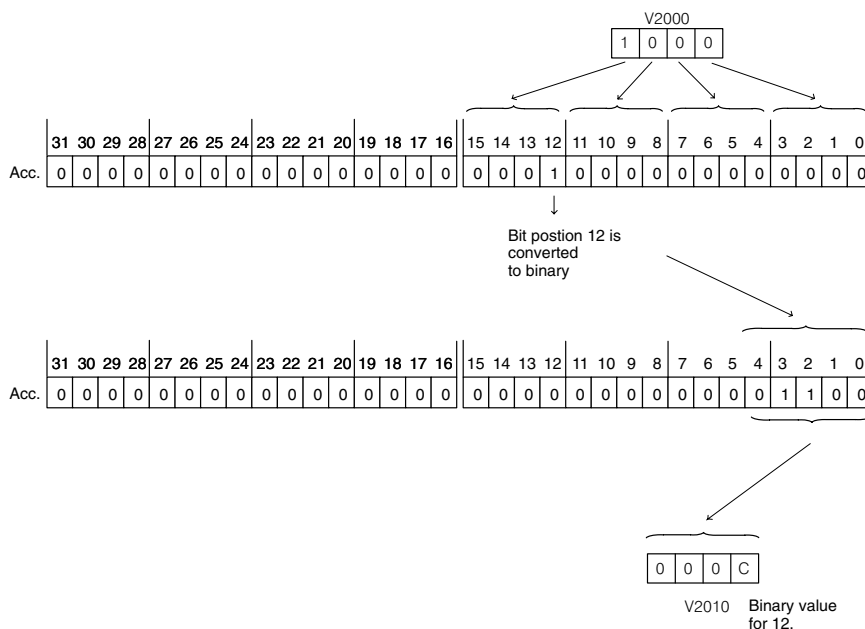
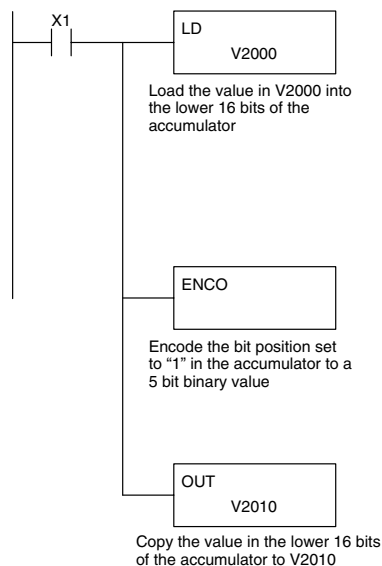
Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.



**NOTE:** The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, The value in V2000 is loaded into the accumulator using the Load instruction. The bit position set to a “1” in the accumulator is encoded to the corresponding 5 bit binary value using the Encode instruction. The value in the lower 16 bits of the accumulator is copied to V2010 using the Out instruction.

DirectSOFT

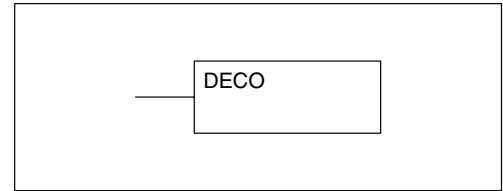


Handheld Programmer Keystrokes

\$	STR	→	B <sub>1</sub>	ENT										
SHFT	L	ANDST	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT					
SHFT	E <sub>4</sub>	N	TMR	C <sub>2</sub>	O	INST#	ENT							
GX	OUT	→	SHFT	V	AND	C <sub>2</sub>	A <sub>0</sub>	B <sub>1</sub>	A <sub>0</sub>	ENT				

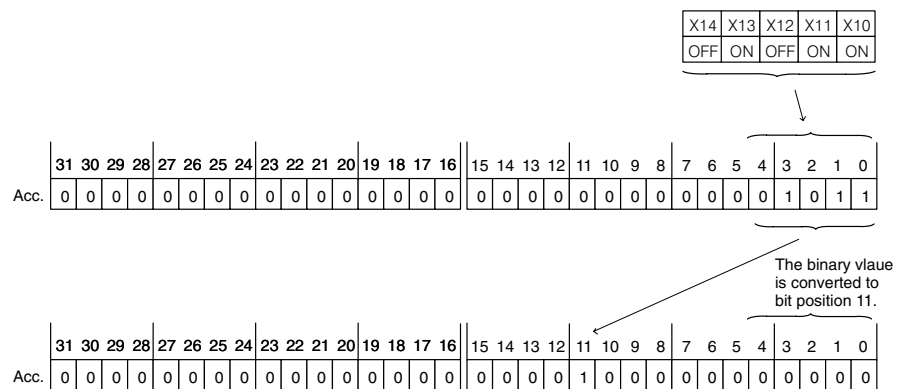
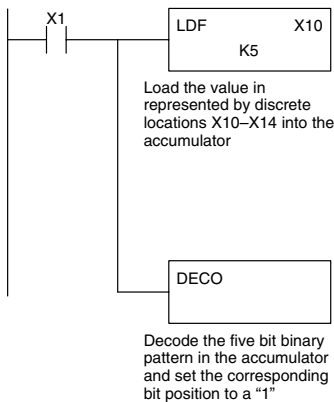
## Decode (DECO)

The Decode instruction decodes a 5 bit binary value of 0–31 (0–1F HEX) in the accumulator by setting the appropriate bit position to a 1. If the accumulator contains the value F (HEX), bit 15 will be set in the accumulator. If the value to be decoded is greater than 31, the number is divided by 32 until the value is less than 32 and then the value is decoded.

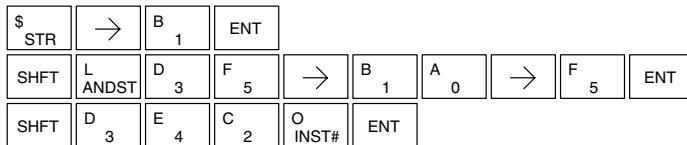


In the following example when X1 is on, the value formed by discrete locations X10–X14 is loaded into the accumulator using the Load Formatted instruction. The five bit binary pattern in the accumulator is decoded by setting the corresponding bit position to a “1” using the Decode instruction.

### DirectSOFT



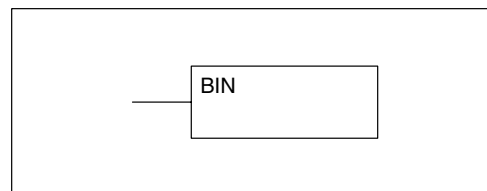
### Handheld Programmer Keystrokes



# Number Conversion Instructions (Accumulator)

## Binary (BIN)

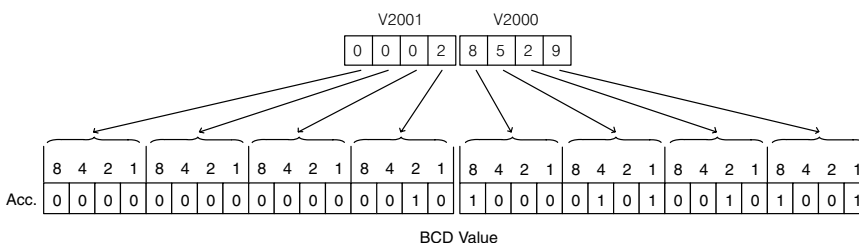
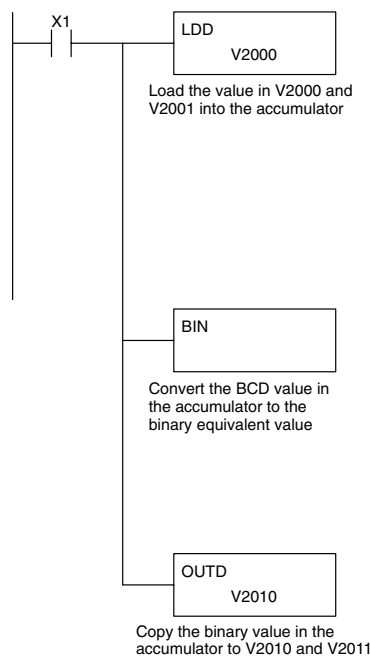
The Binary instruction converts a BCD value in the accumulator to the equivalent binary value. The result resides in the accumulator.



In the following example, when X1 is on, the value in V2000 and V2001 is loaded into the accumulator using the Load Double instruction. The BCD value in the accumulator is converted to the binary (HEX) equivalent using the BIN instruction. The binary value in the accumulator is copied to V2010 and V2011 using the Out Double instruction. (The handheld programmer will display the binary value in V2010 and V2011 as a HEX value.)

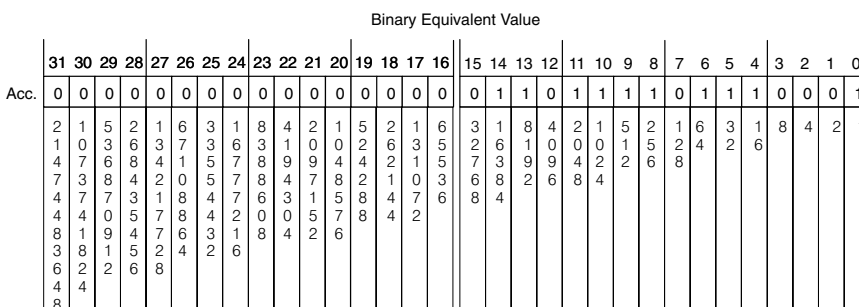
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	on when a BCD instruction is executed and a NON-BCD number was encountered.

DirectSOFT

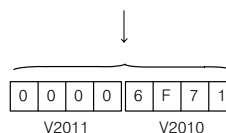


BCD Value

$$28529 = 16384 + 8192 + 2048 + 1024 + 512 + 256 + 64 + 32 + 16 + 1$$

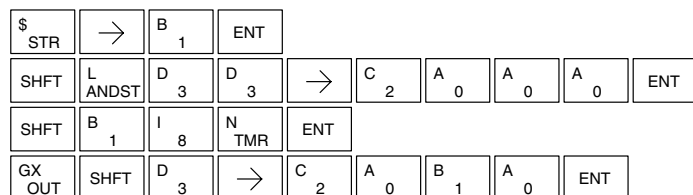


Binary Equivalent Value



The binary (HEX) value copied to V2010

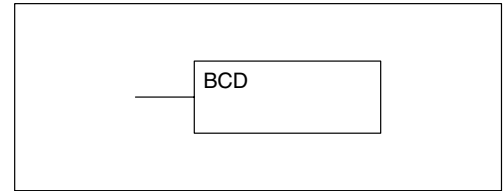
Handheld Programmer Keystrokes





## Binary Coded Decimal (BCD)

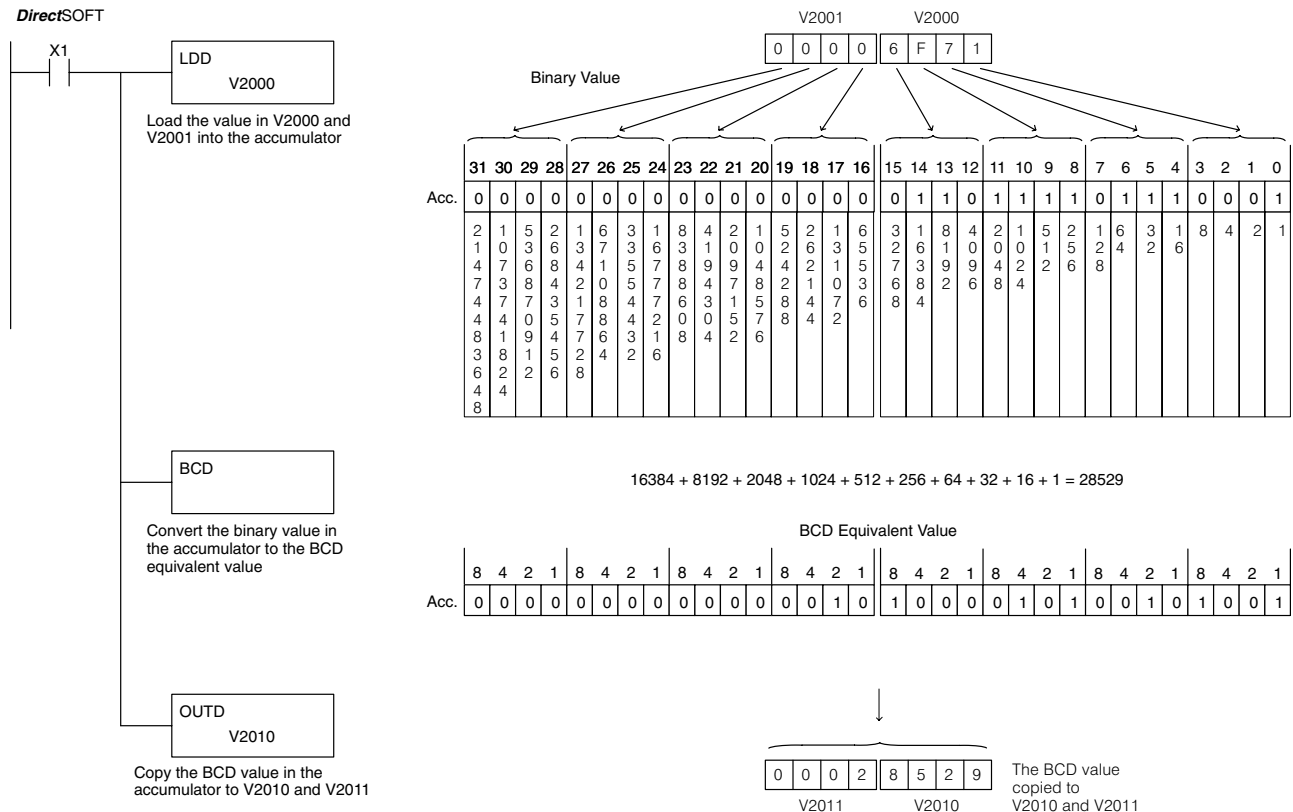
The Binary Coded Decimal instruction converts a binary value in the accumulator to the equivalent BCD value. The result resides in the accumulator.



In the following example, when X1 is on, the binary (HEX) value in V2000 and V2001 is loaded into the accumulator using the Load Double instruction. The binary value in the accumulator is converted to the BCD equivalent value using the BCD instruction. The BCD value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

DirectSOFT

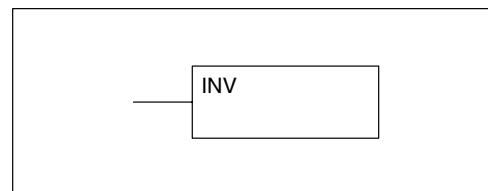


Handheld Programmer Keystrokes

\$ STR	→	B <sub>1</sub>	ENT													
SHFT	L ANDST	D <sub>3</sub>	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT							
SHFT	B <sub>1</sub>	C <sub>2</sub>	D <sub>3</sub>	ENT												
GX OUT	SHFT	D <sub>3</sub>	→	C <sub>2</sub>	A <sub>0</sub>	B <sub>1</sub>	A <sub>0</sub>	ENT								

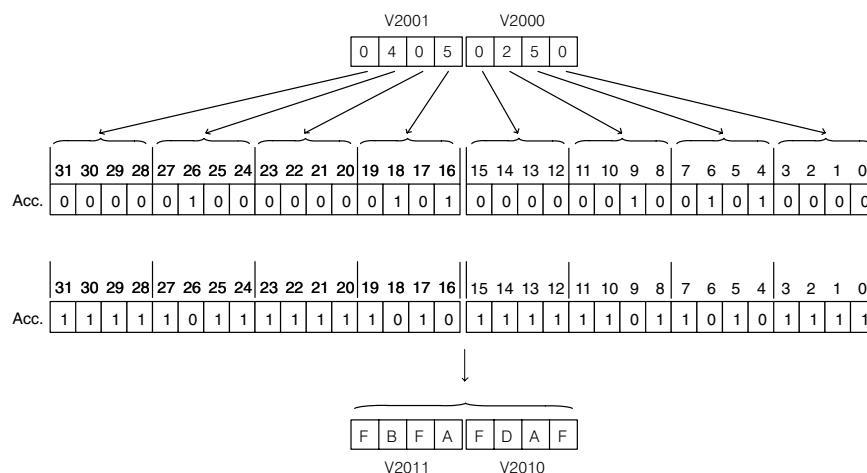
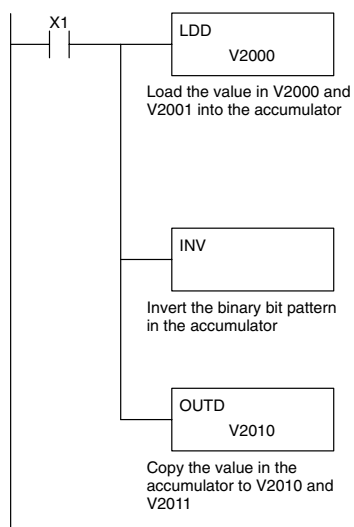
**Invert  
(INV)**

The Invert instruction inverts or takes the one's complement of the 32 bit value in the accumulator. The result resides in the accumulator.

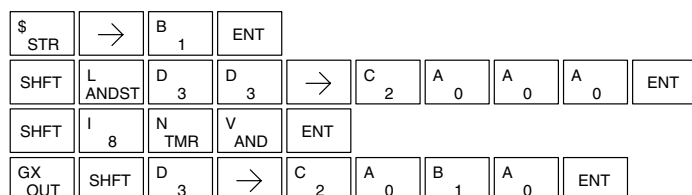


In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is inverted using the Invert instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

DirectSOFT

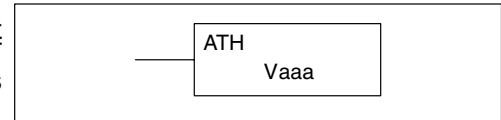


Handheld Programmer Keystrokes



**ASCII to HEX  
(ATH)**

The ASCII TO HEX instruction converts a table of ASCII values to a specified table of HEX values. ASCII values are two digits and their HEX equivalents are one digit.



This means an ASCII table of four V memory locations would only require two V memory locations for the equivalent HEX table. The function parameters are loaded into the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program an ASCII to HEX table function. The example on the following page shows a program for the ASCII to HEX table function.

Step 1: — Load the number of V memory locations for the ASCII table into the first level of the accumulator stack.

Step 2: — Load the starting V memory location for the ASCII table into the accumulator. This parameter must be a HEX value.

Step 3: — Specify the starting V memory location (Vaaa) for the HEX table in the ATH instruction.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

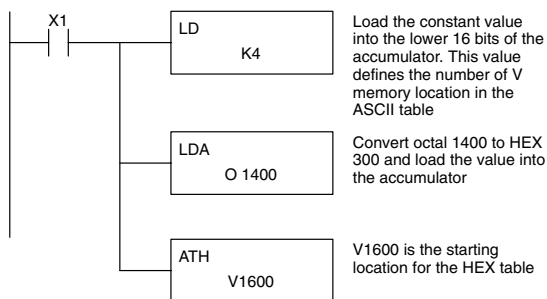
Operand Data Type		DL05 Range
		aaa
Vmemory	V	All (See p. 4–28)

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

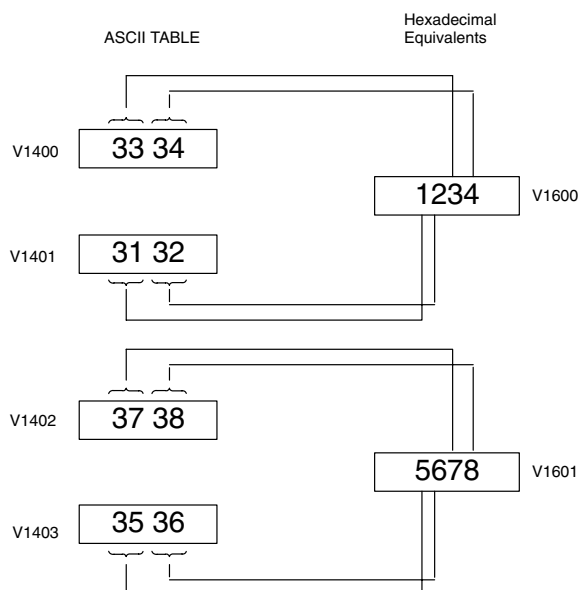
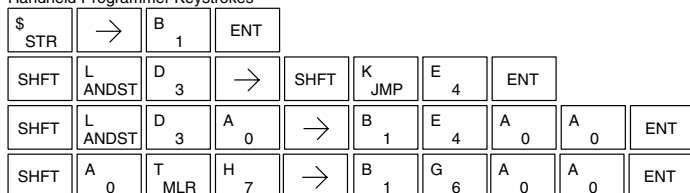
In the example on the following page, when X1 is ON the constant (K4) is loaded into the accumulator using the Load instruction and will be placed in the first level of the accumulator stack when the next Load instruction is executed. The starting location for the ASCII table (V1400) is loaded into the accumulator using the Load Address instruction. The starting location for the HEX table (V1600) is specified in the ASCII to HEX instruction. The table below lists valid ASCII values for ATH conversion.

ASCII Values Valid for ATH Conversion			
ASCII Value	Hex Value	ASCII Value	Hex Value
30	0	38	8
31	1	39	9
32	2	41	A
33	3	42	B
34	4	43	C
35	5	44	D
36	6	45	E
37	7	46	F

**DirectSOFT Display**

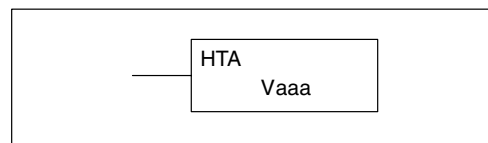


## Handheld Programmer Keystrokes



## HEX to ASCII (HTA)

The HEX to ASCII instruction converts a table of HEX values to a specified table of ASCII values. HEX values are one digit and their ASCII equivalents are two digits.



This means a HEX table of two V memory locations would require four V memory locations for the equivalent ASCII table. The function parameters are loaded into the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program a HEX to ASCII table function. The example on the following page shows a program for the HEX to ASCII table function.

Step 1: — Load the number of V memory locations in the HEX table into the first level of the accumulator stack.

Step 2: — Load the starting V memory location for the HEX table into the accumulator. This parameter must be a HEX value.

Step 3: — Specify the starting V memory location (Vaaa) for the ASCII table in the HTA instruction.

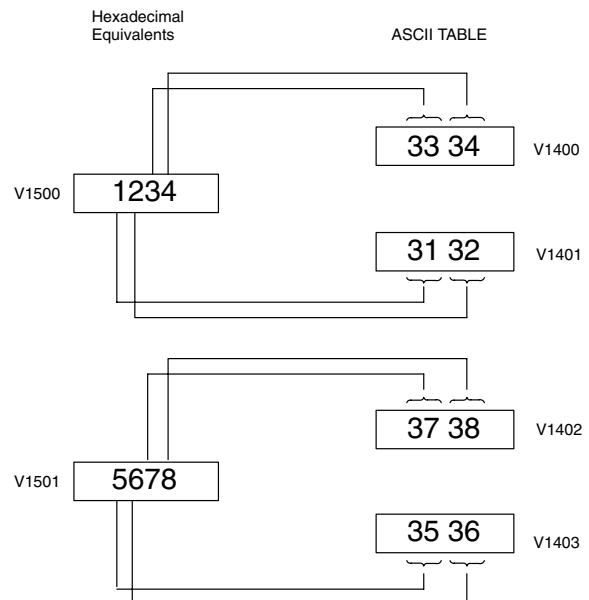
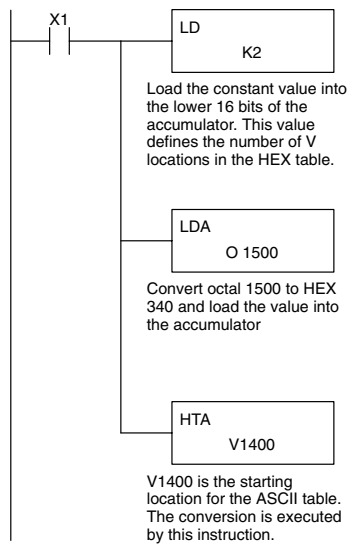
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type	DL05 Range
	aaa
Vmemory V	All (See p. 4-28)

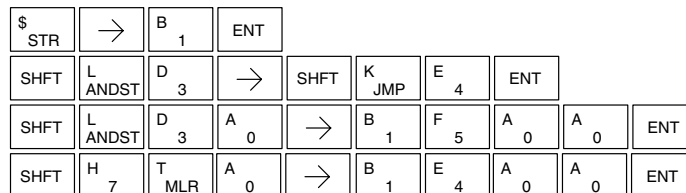
Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

In the following example, when X1 is ON the constant (K2) is loaded into the accumulator using the Load instruction. The starting location for the HEX table (V1500) is loaded into the accumulator using the Load Address instruction. The starting location for the ASCII table (V1400) is specified in the HEX to ASCII instruction.

**DirectSOFT Display**



**Handheld Programmer Keystrokes**

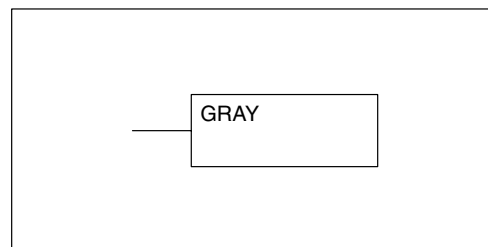


The table below lists valid ASCII values for HTA conversion.

ASCII Values Valid for HTA Conversion			
Hex Value	ASCII Value	Hex Value	ASCII Value
0	30	8	38
1	31	9	39
2	32	A	41
3	33	B	42
4	34	C	43
5	35	D	44
6	36	E	45
7	37	F	46

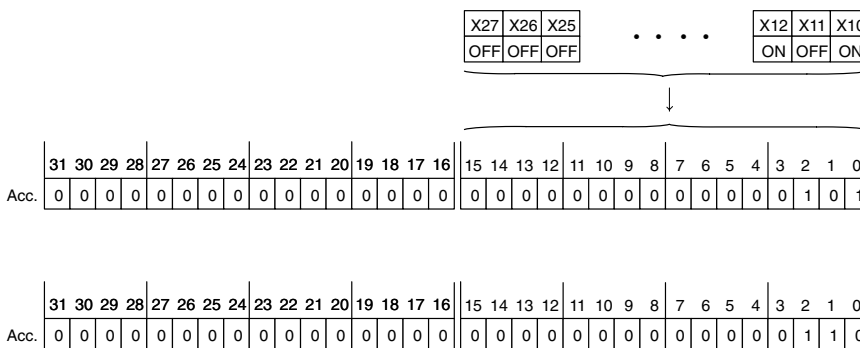
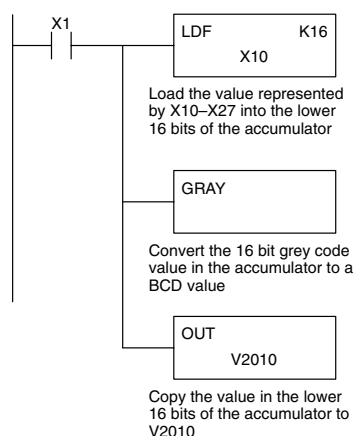
**Gray Code  
(GRAY)**

The Gray code instruction converts a 16 bit gray code value to a BCD value. The BCD conversion requires 10 bits of the accumulator. The upper 22 bits are set to "0". This instruction is designed for use with devices (typically encoders) that use the grey code numbering scheme. The Gray Code instruction will directly convert a gray code number to a BCD number for devices having a resolution of 512 or 1024 counts per revolution. If a device having a resolution of 360 counts per revolution is to be used you must subtract a BCD value of 76 from the converted value to obtain the proper result. For a device having a resolution of 720 counts per revolution you must subtract a BCD value of 152.



In the following example, when X1 is ON the binary value represented by X10–X27 is loaded into the accumulator using the Load Formatted instruction. The gray code value in the accumulator is converted to BCD using the Gray Code instruction. The value in the lower 16 bits of the accumulator is copied to V2010.

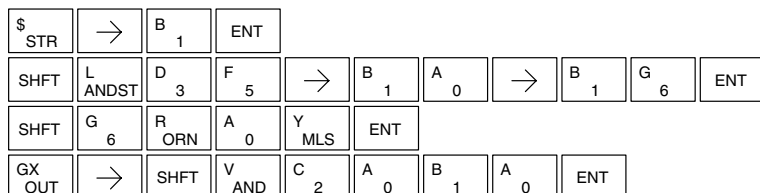
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

**DirectSOFT**

Gray Code      BCD

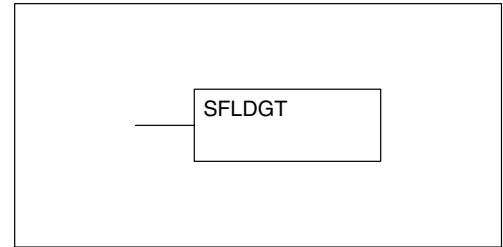
0000000000	0000
0000000001	0001
0000000011	0002
0000000010	0003
0000000110	0004
0000000111	0005
000000101	0006
000000100	0007
.	.
.	.
.	.
1000000001	1022
1000000000	1023

0 0 0 6  
V2010

**Handheld Programmer Keystrokes**

## Shuffle Digits (SFLDGT)

The Shuffle Digits instruction shuffles a maximum of 8 digits rearranging them in a specified order. This function requires parameters to be loaded into the first level of the accumulator stack and the accumulator with two additional instructions. Listed below are the steps necessary to use the shuffle digit function. The example on the following page shows a program for the Shuffle Digits function.



Step 1:— Load the value (digits) to be shuffled into the first level of the accumulator stack.

Step 2:— Load the order that the digits will be shuffled to into the accumulator.

Note:— If the number used to specify the order contains a 0 or 9–F, the corresponding position will be set to 0.  
See example on the next page.

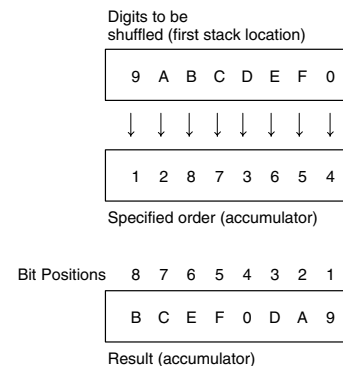
Note:—If the number used to specify the order contains duplicate numbers, the most significant duplicate number is valid. The result resides in the accumulator.  
See example on the next page.

Step 3:— Insert the SFLDGT instruction.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

## Shuffle Digits Block Diagram

There are a maximum of 8 digits that can be shuffled. The bit positions in the first level of the accumulator stack defines the digits to be shuffled. They correspond to the bit positions in the accumulator that define the order the digits will be shuffled. The digits are shuffled and the result resides in the accumulator.

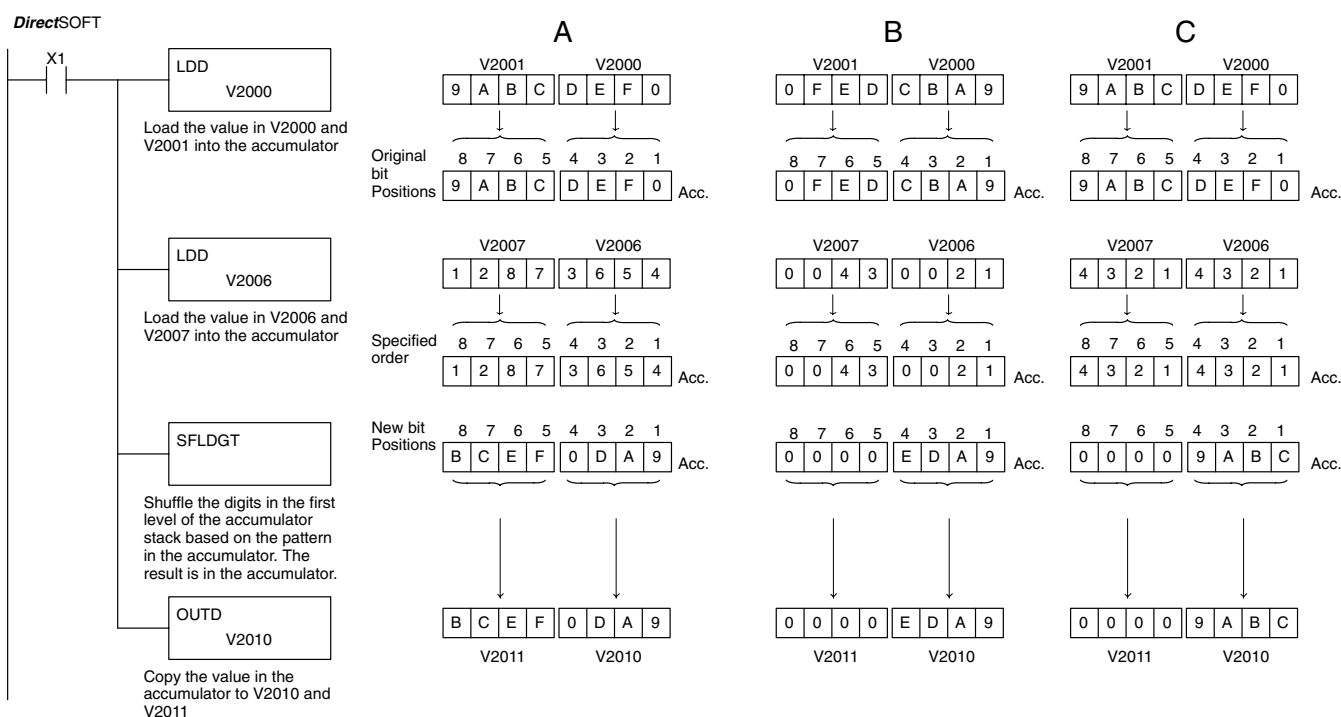


In the following example when X1 is on, The value in the first level of the accumulator stack will be reorganized in the order specified by the value in the accumulator.

Example A shows how the shuffle digits works when 0 or 9–F is not used when specifying the order the digits are to be shuffled. Also, there are no duplicate numbers in the specified order.

Example B shows how the shuffle digits works when a 0 or 9–F is used when specifying the order the digits are to be shuffled. Notice when the Shuffle Digits instruction is executed, the bit positions in the first stack location that had a corresponding 0 or 9–F in the accumulator (order specified) are set to “0”.

Example C shows how the shuffle digits works when duplicate numbers are used specifying the order the digits are to be shuffled. Notice when the Shuffle Digits instruction is executed, the most significant duplicate number in the order specified is used in the result.



#### Handheld Programmer Keystrokes

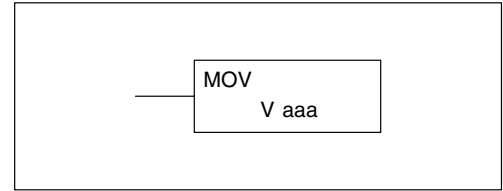
\$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	G 6	ENT
SHFT	S RST	SHFT	F 5	L ANDST	D 3	G 6	T MLR		ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0		ENT



## Table Instructions

### Move (MOV)

The Move instruction moves the values from a V memory table to another V memory table the same length. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Move function.



- Step 1:— Load the number of V memory locations to be moved into the first level of the accumulator stack. This parameter is a HEX value (K40 max, 100 octal).
- Step 2:— Load the starting V memory location for the locations to be moved into the accumulator. This parameter is a HEX value.
- Step 3:— Insert the MOVE instruction which specifies starting V memory location (Vaaa) for the destination table.

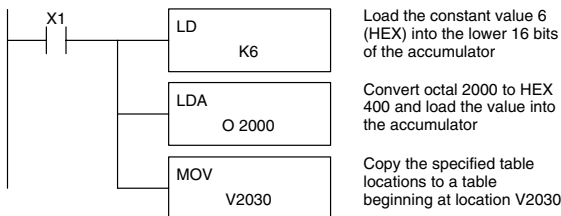
**Helpful Hint:** — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL05 Range
		aaa
V memory	V	All (See page 4-28)
Pointer	P	All (See page 4-28)

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

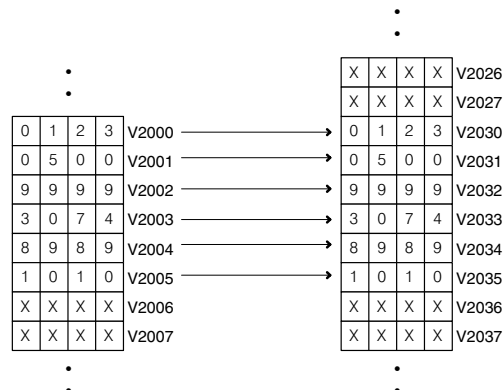
In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 2000 (V2000), the starting location for the source table is loaded into the accumulator. The destination table location (V2030) is specified in the Move instruction.

#### DirectSOFT



#### Handheld Programmer Keystrokes

\$ STR	→	B <sub>1</sub>	ENT										
SHFT	L ANDST	D <sub>3</sub>	→	SHFT	K JMP	G <sub>6</sub>	ENT						
SHFT	L ANDST	D <sub>3</sub>	A <sub>0</sub>	→	C <sub>2</sub>	A <sub>0</sub>	A <sub>0</sub>	A <sub>0</sub>	ENT				
SHFT	M ORST	O INST#	V AND	→	C <sub>2</sub>	A <sub>0</sub>	D <sub>3</sub>	A <sub>0</sub>	ENT				

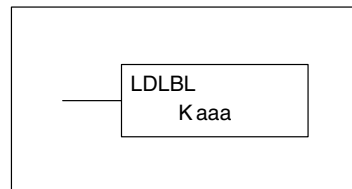
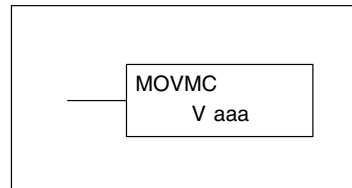


**Move Memory  
Cartridge /  
Load Label****(MOVMC), (LDLBL)**

The Move Memory Cartridge and the Load Label instructions are used to copy data from program ladder memory to V memory. The Load Label instruction is used with the MOVMC instruction when copying data *from* program ladder memory *to* V memory.

To copy data from the program ladder memory to V memory, the function parameters are loaded into the first two levels of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Move Memory Cartridge and Load Label functions.

- Step 1:— Load the number of words to be copied into the second level of the accumulator stack.
- Step 2:— Load the offset for the data label area in ladder memory and the beginning of the V memory block into the first level of the stack.
- Step 3:— Load the *source data label* (LDLBL Kaaa) into the accumulator when copying data from ladder memory to V memory. This is the source location of the value.
- Step 4:— Insert the MOVMC instruction which specifies destination in V-memory (Vaaa). This is the copy destination.



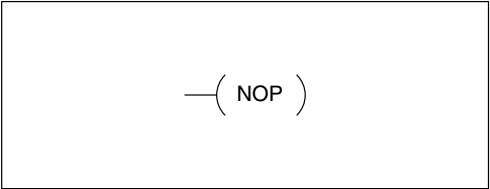
Operand Data Type		DL05 Range
	A	aaa
V memory	V	All (See page 4-28)



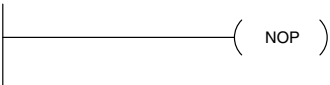
CPU Control Instructions

No Operation  
(NOP)

The No Operation is an empty (not programmed) memory location.



DirectSOFT

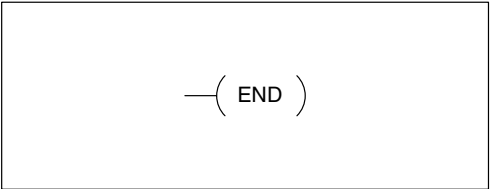


Handheld Programmer Keystrokes



End  
(END)

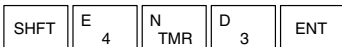
The End instruction marks the termination point of the normal program scan. An End instruction is required at the end of the main program body. If the End instruction is omitted an error will occur and the CPU will not enter the Run Mode. Data labels, subroutines and interrupt routines are placed after the End instruction. The End instruction is not conditional; therefore, no input contact is allowed.



DirectSOFT

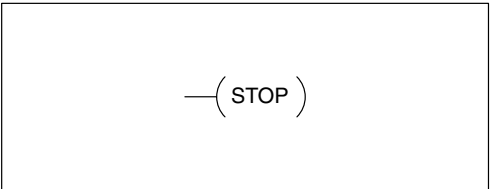


Handheld Programmer Keystrokes



Stop  
(STOP)

The Stop instruction changes the operational mode of the CPU from Run to Program (Stop) mode. This instruction is typically used to stop PLC operation in an error condition.



In the following example, when C0 turns on, the CPU will stop operation and switch to the program mode.

DirectSOFT



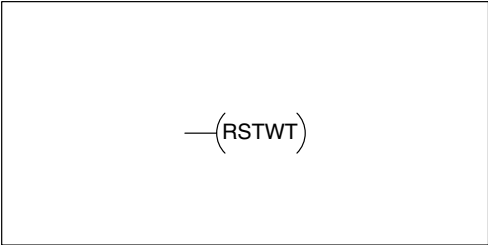
Handheld Programmer Keystrokes



Discrete Bit Flags	Description
SP16	On when the DL05 goes into the TERM_PRG mode.
SP53	On when the DL05 goes into the PRG mode.

**Reset Watch Dog  
Timer  
(RSTWT)**

The Reset Watch Dog Timer instruction resets the CPU scan timer. The default setting for the watch dog timer is 200ms. Scan times very seldom exceed 200ms, but it is possible. For/next loops, subroutines, interrupt routines, and table instructions can be programmed such that the scan becomes longer than 200ms. When instructions are used in a manner that could exceed the watch dog timer setting, this instruction can be used to reset the timer.



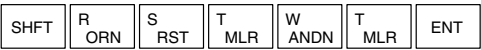
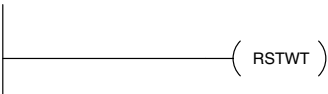
A software timeout error (E003) will occur and the CPU will enter the program mode if the scan time exceeds the watch dog timer setting. Placement of the RSTWT instruction in the program is very important. The instruction has to be executed before the scan time exceeds the watch dog timer's setting.

If the scan time is consistently longer than the watch dog timer's setting, the timeout value may be permanently increased from the default value of 200ms by AUX 55 on the HPP or the appropriate auxiliary function in your programming package. This eliminates the need for the RSTWT instruction.

In the following example the CPU scan timer will be reset to 0 when the RSTWT instruction is executed. See the For/Next instruction for a detailed example.

DirectSOFT

Handheld Programmer Keystrokes



## Program Control Instructions

### For / Next (FOR) (NEXT)

The For and Next instructions are used to execute a section of ladder logic between the For and Next instruction a specified numbers of times. When the For instruction is enabled, the program will loop the specified number of times. If the For instruction is not energized the section of ladder logic between the For and Next instructions is not executed.

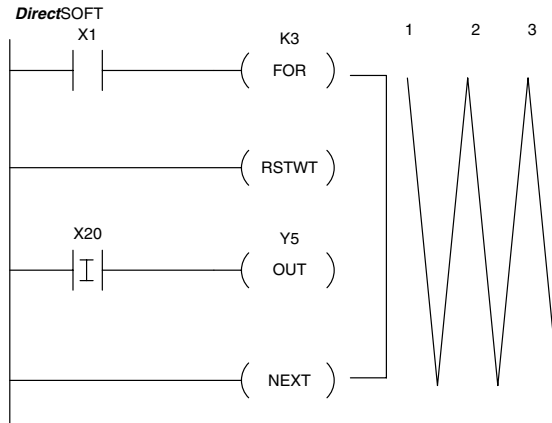
For / Next instructions cannot be nested. The normal I/O update and CPU housekeeping is suspended while executing the For / Next loop. The program scan can increase significantly, depending on the amount of times the logic between the For and Next instruction is executed. With the exception of immediate I/O instructions, I/O will not be updated until the program execution is completed for that scan. Depending on the length of time required to complete the program execution, it may be necessary to reset the watch dog timer inside of the For / Next loop using the RSTWT instruction.

—( A aaa  
FOR )

—( NEXT )

Operand Data Type		DL05 Range
	A	aaa
V memory	V	All (See page 4-28)
Constant	K	1-9999

In the following example, when X1 is on, the application program inside the For / Next loop will be executed three times. If X1 is off the program inside the loop will not be executed. The immediate instructions may or may not be necessary depending on your application. Also, The RSTWT instruction is not necessary if the For / Next loop does not extend the scan time larger the Watch Dog Timer setting. For more information on the Watch Dog Timer, refer to the RSTWT instruction.



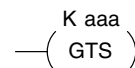
Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT			
SHFT	F 5	O INST#	R ORN	→	D 3	ENT
SHFT	R ORN	S RST	T MLR	W ANDN	T MLR	ENT
\$ STR	SHFT	I 8	→	C 2	A 0	ENT
GX OUT	→	F 5	ENT			
SHFT	N TMR	E 4	X SET	T MLR	ENT	

**Goto Subroutine  
(GTS)  
(SBR)**

The Goto Subroutine instruction allows a section of ladder logic to be placed outside the main body of the program execute only when needed. There can be a maximum of 64 GTS instructions and 64 SBR instructions used in a program. The GTS instructions can be nested up to 8 levels. An error E412 will occur if the maximum limits are exceeded. Typically this will be used in an application where a block of program logic may be slow to execute and is not required to execute every scan. The subroutine label and all associated logic is placed after the End statement in the program. When the subroutine is called from the main program, the CPU will execute the subroutine (SBR) with the same constant number (K) as the GTS instruction which called the subroutine.

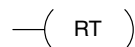
By placing code in a subroutine it is only scanned and executed when needed since it resides after the End instruction. Code which is not scanned does not impact the overall scan time of the program.




Operand Data Type		DL05 Range
		aaa
Constant	K	1-FFFF

**Subroutine Return  
(RT)**

When a Subroutine Return is executed in the subroutine the CPU will return to the point in the main body of the program from which it was called. The Subroutine Return is used as termination of the subroutine which must be the last instruction in the subroutine and is a stand alone instruction (no input contact on the rung).

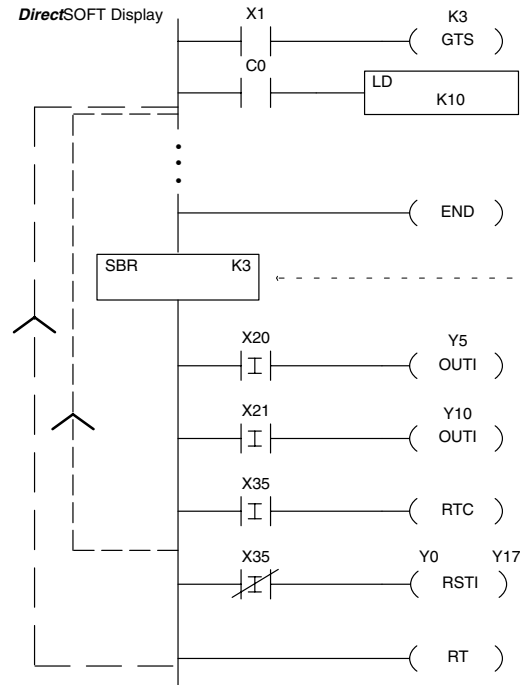

**Subroutine Return  
Conditional  
(RTC)**

The Subroutine Return Conditional instruction is an optional instruction used with an input contact to implement a conditional return from the subroutine. The Subroutine Return (RT) is still required for termination of the Subroutine.

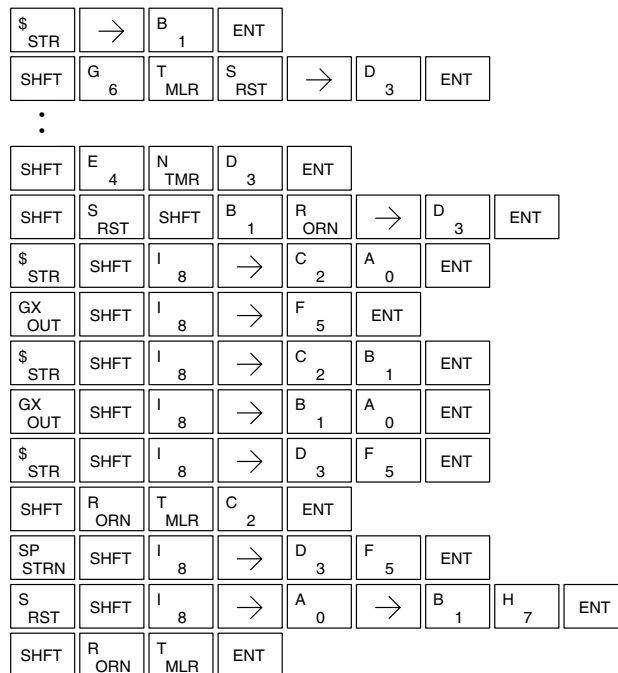




In the following example, when X1 is on, Subroutine K3 will be called. The CPU will jump to the Subroutine Label K3 and the ladder logic in the subroutine will be executed. If X35 is on the CPU will return to the main program at the RTC instruction. If X35 is not on Y0-Y17 will be reset to off and then the CPU will return to the main body of the program.

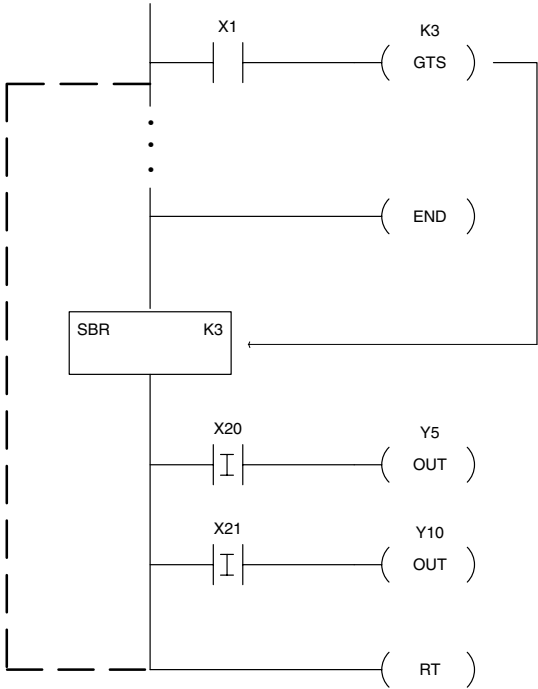


Handheld Programmer Keystrokes



In the following example, when X1 is on, Subroutine K3 will be called. The CPU will jump to the Subroutine Label K3 and the ladder logic in the subroutine will be executed. The CPU will return to the main body of the program after the RT instruction is executed.

DirectSOFT

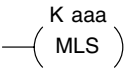


Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT			
SHFT	G 6	T MLR	S RST	→	D 3	ENT
:						
SHFT	E 4	N TMR	D 3	ENT		
SHFT	S RST	SHFT	B 1	R ORN	→	D 3
\$ STR	SHFT	I 8	→	C 2	A 0	ENT
GX OUT	→	F 5	ENT			
\$ STR	SHFT	I 8	→	C 2	B 1	ENT
GX OUT	→	B 1	A 0	ENT		
SHFT	R ORN	T MLR	ENT			

**Master Line Set (MLS)**

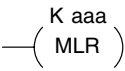
The Master Line Set instruction allows the program to control sections of ladder logic by forming a new power rail controlled by the main left power rail. The main left rail is always master line 0. When a MLS K1 instruction is used, a new power rail is created at level 1. Master Line Sets and Master Line Resets can be used to nest power rails up to seven levels deep.



Operand Data Type		DL05 Range
		aaa
Constant	K	1-7

**Master Line Reset (MLR)**

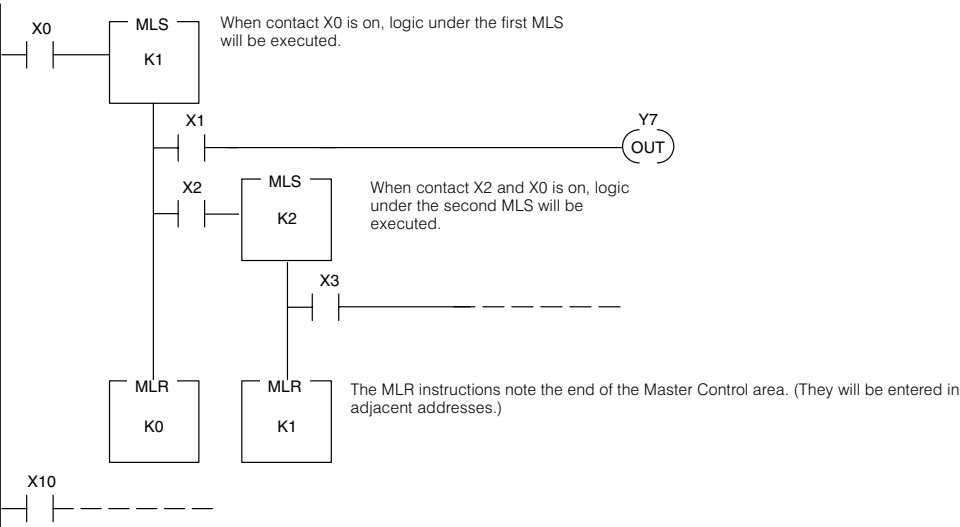
The Master Line Reset instruction marks the end of control for the corresponding MLS instruction. The MLR reference is one less than the corresponding MLS.



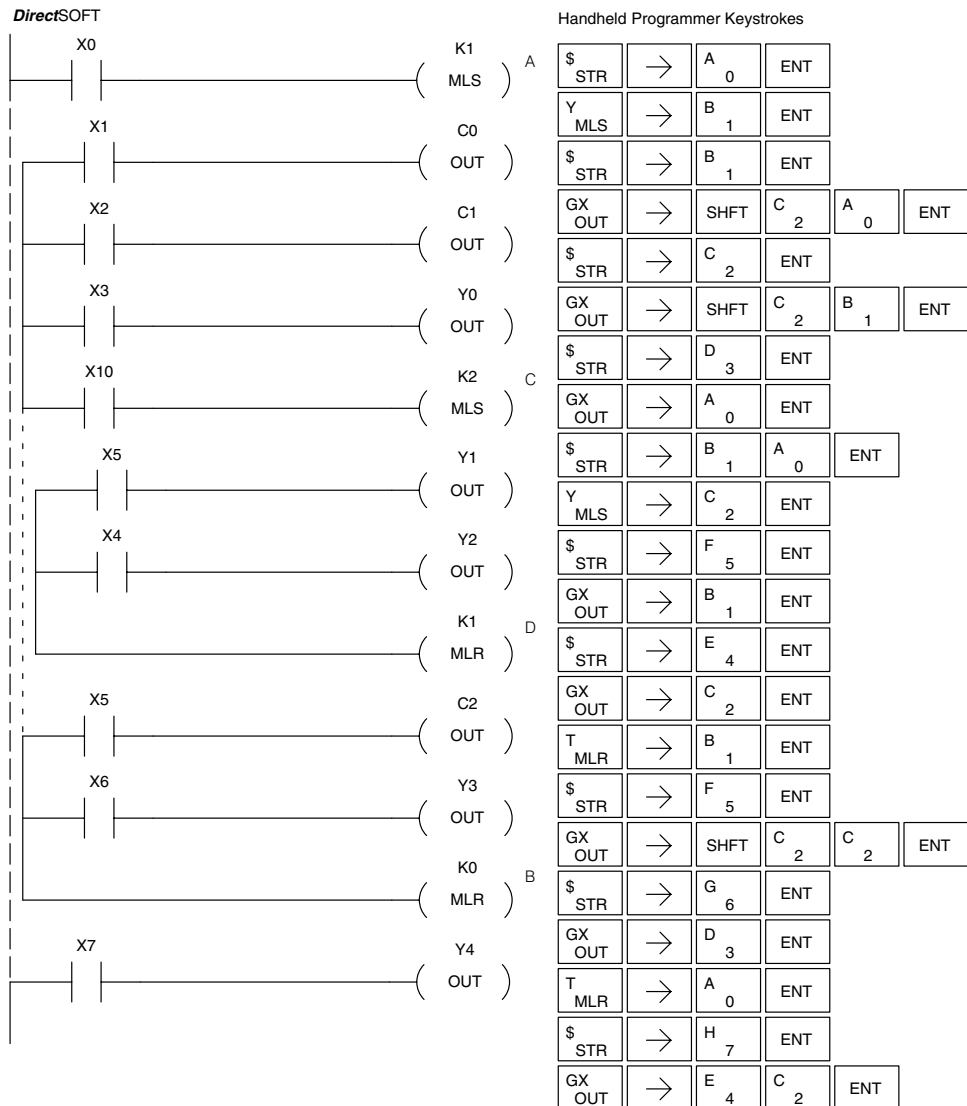
Operand Data Type		DL05 Range
		aaa
Constant	K	0-7

**Understanding Master Control Relays**

The Master Line Set (MLS) and Master Line Reset (MLR) instructions allow you to quickly enable (or disable) sections of the RLL program. This provides program control flexibility. The following example shows how the MLS and MLR instructions operate by creating a sub power rail for control logic.



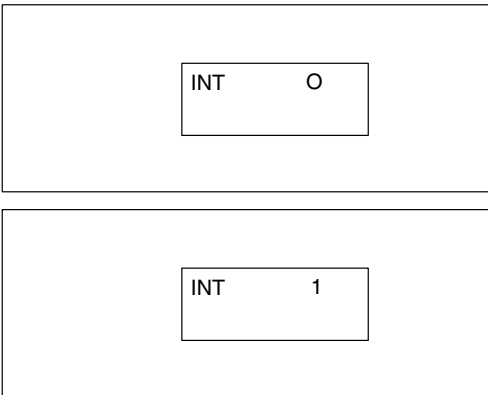
**MLS/MLR Example** In the following MLS/MLR example logic between the first MLS K1 (A) and MLR K0 (B) will function only if input X0 is on. The logic between the MLS K2 (C) and MLR K1 (D) will function only if input X10 and X0 is on. The last rung is not controlled by either of the MLS coils.



# Interrupt Instructions

## Interrupt (INT)

The Interrupt instruction allows a section of ladder logic to be placed below the main body of the program and executed only when needed. High-Speed I/O Modes 10, 20, and 40 can generate an interrupt. With Mode 40, you may select an external interrupt (input X0), or a time-based interrupt (5–999 mS).



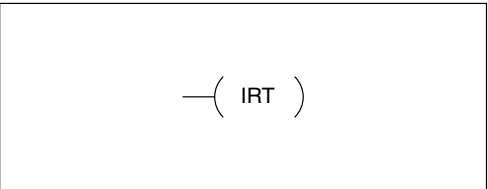
Typically, interrupts are used in an application when a fast response to an input is needed or a program section must execute faster than the normal CPU scan. The interrupt label and all associated logic must be placed after the End statement in the program. When an interrupt occurs, the CPU will complete execution of the current instruction it is processing in ladder logic, then execute the interrupt routine. After interrupt routine execution, the ladder program resumes from the point at which it was interrupted.

See Chapter 3, the section on Mode 40 (Interrupt) Operation for more details on interrupt configuration. In the DL05, only one hardware interrupt is available.

Operand Data Type	DL05 Range
Constant O	0, 1

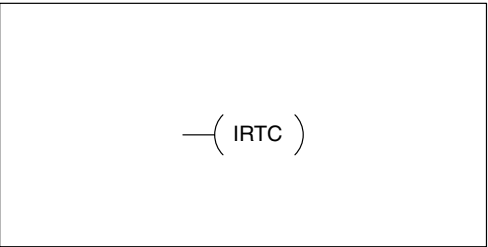
## Interrupt Return (IRT)

An Interrupt Return is normally executed as the last instruction in the interrupt routine. It returns the CPU to the point in the main program from which it was called. The Interrupt Return is a stand-alone instruction (no input contact on the rung).



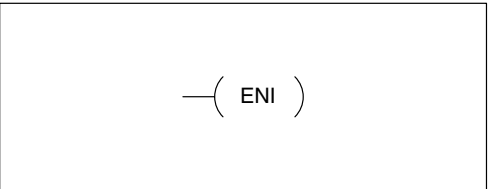
## Interrupt Return Conditional (IRTC)

The Interrupt Return Conditional instruction is an optional instruction used with an input contact to implement a conditional return from the interrupt routine. The Interrupt Return is required to terminate the interrupt routine.



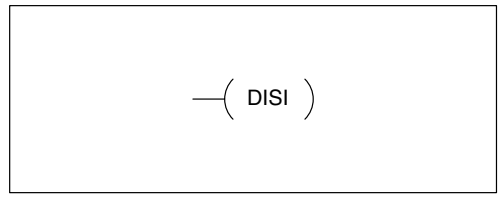
## Enable Interrupts (ENI)

The Enable Interrupt instruction is placed in the main ladder program (before the End instruction), enabling the interrupt. The interrupt remains enabled until the program executes a Disable Interrupt instruction.



Disable Interrupts (DISI)

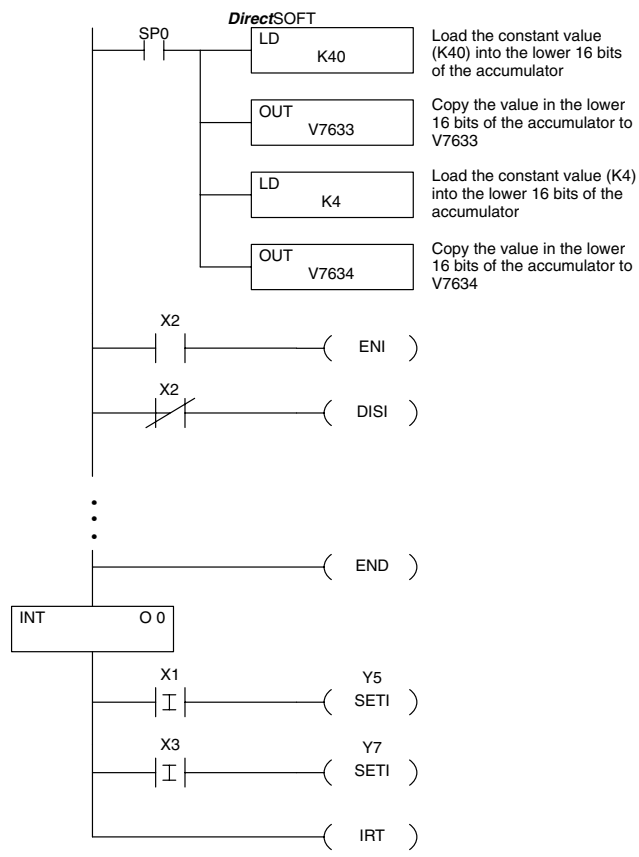
A Disable Interrupt instruction in the main body of the application program (before the End instruction) will disable the interrupt (either external or timed). The interrupt remains disabled until the program executes an Enable Interrupt instruction.



External Interrupt Program Example

In the following example, we do some initialization on the first scan, using the first-scan contact SP0. The interrupt feature is the HSIO Mode 40. Then we configure X0 as the external interrupt by writing to its configuration register, V7634. See Chapter 3, Mode 40 Operation for more details.

During program execution, when X2 is on the interrupt is enabled. When X2 is off the interrupt will be disabled. When an interrupt signal (X0) occurs the CPU will jump to the interrupt label INT O 0. The application ladder logic in the interrupt routine will be performed. The CPU will return to the main body of the program after the IRT instruction is executed.



Handheld Programmer Keystrokes

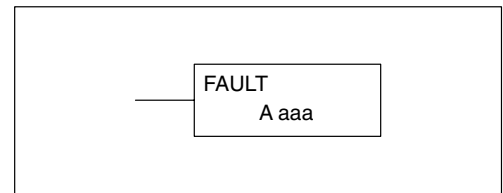
\$ STR	→	SHFT	SP STRN	A 0	ENT
SHFT	L ANDST	D 3	→	SHFT	K JMP E 4 A 0 ENT
GX OUT	→	SHFT	V AND	H 7	G 6 D 3 D 3 ENT
SHFT	L ANDST	D 3	→	SHFT	K JMP E 4 ENT
GX OUT	→	SHFT	V AND	H 7	G 6 D 3 E 4 ENT
\$ STR	→	C 2	ENT		
SHFT	E 4	N TMR	I 8	ENT	
SP STRN	→	C 2	ENT		
SHFT	D 3	I 8	S RST	I 8	ENT
.					
SHFT	E 4	N TMR	D 3	ENT	
SHFT	I 8	N TMR	T MLR	→	A 0 ENT
\$ STR	SHFT	I 8	→	B 1	ENT
X SET	SHFT	I 8	→	F 5	ENT
\$ STR	SHFT	I 8	→	D 3	ENT
X SET	SHFT	I 8	→	H 7	ENT
SHFT	I 8	R ORN	T MLR	ENT	



## Message Instructions

### Fault (FAULT)

The Fault instruction is used to display a message on the handheld programmer or in the **DirectSOFT** status bar. The message has a maximum of 23 characters and can be either V memory data, numerical constant data or ASCII text.



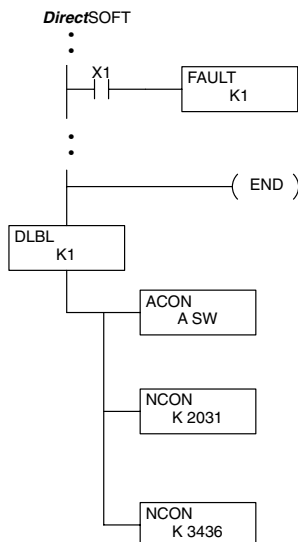
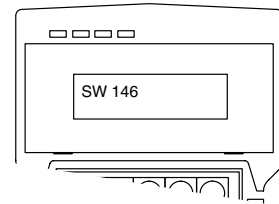
To display the value in a V memory location, specify the V memory location in the instruction. To display the data in ACON (ASCII constant) or NCON (Numerical constant) instructions, specify the constant (K) value for the corresponding data label area.

Operand Data Type		DL05 Range
A		aaa
V memory	V	All (See page 4-28)
Constant	K	1-FFFF

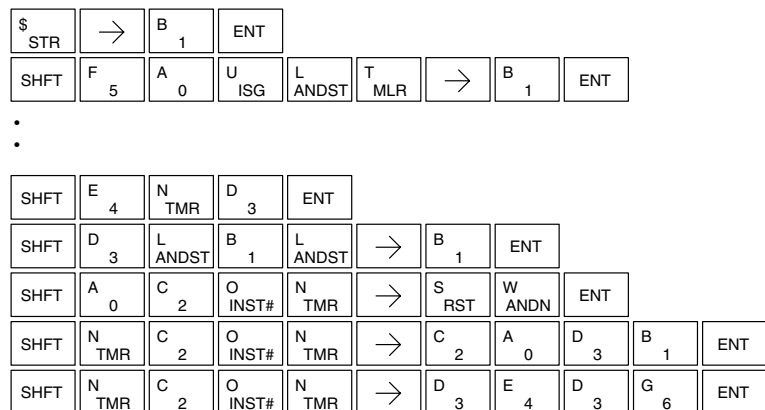
Discrete Bit Flags	Description
SP50	On when the FAULT instruction is executed.

### Fault Example

In the following example when X1 is on, the message SW 146 will display on the handheld programmer. The NCONs use the HEX ASCII equivalent of the text to be displayed. (The HEX ASCII for a blank is 20, a 1 is 31, 4 is 34 ...)



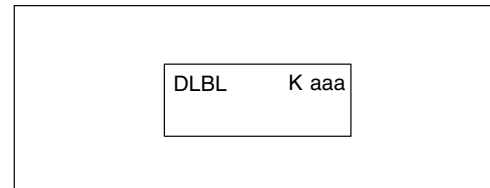
#### Handheld Programmer Keystrokes





### Data Label (DLBL)

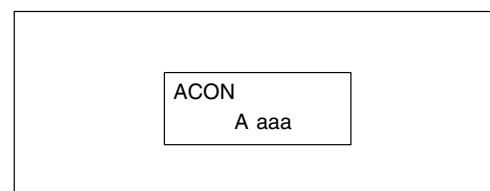
The Data Label instruction marks the beginning of an ASCII / numeric data area. DLBLs are programmed after the End statement. A maximum of 64 DLBL instructions can be used in a program. Multiple NCONs and ACONs can be used in a DLBL area.



Operand Data Type		DL05 Range
		aaa
Constant	K	1-FFFF

### ASCII Constant (ACON)

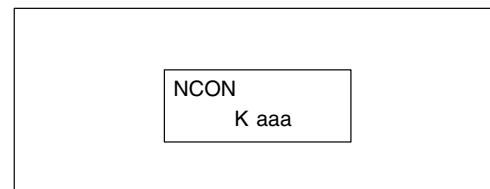
The ASCII Constant instruction is used with the DLBL instruction to store ASCII text for use with other instructions. Two ASCII characters can be stored in an ACON instruction. If only one character is stored in a ACON a leading space will be inserted.



Operand Data Type		DL05 Range
		aaa
ASCII	A	0-9 A-Z

### Numerical Constant (NCON)

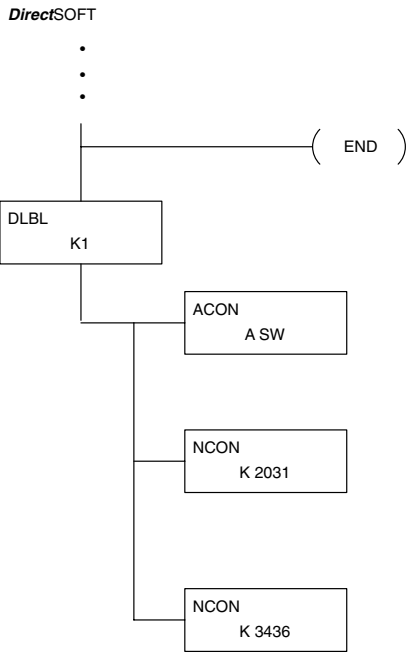
The Numerical Constant instruction is used with the DLBL instruction to store the HEX ASCII equivalent of numerical data for use with other instructions. Two digits can be stored in an NCON instruction.



Operand Data Type		DL05 Range
		aaa
Constant	K	0-FFFF

Data Label  
Example

In the following example, an ACON and two NCON instructions are used within a DLBL instruction to build a text message. See the FAULT instruction for information on displaying messages. The DV-1000 Manual also has information on displaying messages.



Handheld Programmer Keystrokes

•  
•

SHFT	E 4	N TMR	D 3	ENT						
SHFT	D 3	L ANDST	B 1	L ANDST	→	B 1	ENT			
SHFT	A 0	C 2	O INST#	N TMR	→	S RST	W ANDN	ENT		
SHFT	N TMR	C 2	O INST#	N TMR	→	C 2	A 0	D 3	B 1	ENT
SHFT	N TMR	C 2	O INST#	N TMR	→	D 3	E 4	D 3	G 6	ENT

**Print Message  
(PRINT)**

The Print Message instruction prints the embedded text or text/data variable message to the specified communications port (Port 2 on the DL05 CPU), which must have the communications port configured.

```
PRINT  A aaa
      "Hello, this is a PLC message"
```

Data Type	DL05 Range
A	aaa
Constant K	2

You may recall from the CPU specifications in Chapter 3 that the DL05's ports are capable of several protocols. Port 1 cannot be configured for the non-sequence portocol. To configure port 2 using the Handheld Programmer, use AUX 56 and follow the prompts, making the same choices as indicated below on this page. To configure a port in **DirectSOFT**, choose the PLC menu, then Setup, then Setup Secondary Comm Port.

- **Port:** From the port number list box at the top, choose "Port 2".
- **Protocol:** Click the check box to the left of "Non-sequence", and then you'll see the dialog box shown below.

- **Baud Rate:** Choose the baud rate that matches your printer.
- **Stop Bits, Parity:** Choose number of stop bits and parity setting to match your printer.
- **Memory Address:** Choose a V-memory address for **DirectSOFT** to use to store the port setup information. You will need to reserve 9 words in V-memory for this purpose. Select "Always use for printing" if it applies.



Then click the button indicated to send the Port 2 configuration to the CPU, and click Close. Then see Chapter 3 for port wiring information, in order to connect your printer to the DL05.

Port 2 on the DL05 has standard RS232 levels, and should work with most printer serial input connections.

**Text element** – this is used for printing character strings. The character strings are defined as the character (more than 0) ranged by the double quotation marks. Two hex numbers preceded by the dollar sign means an 8-bit ASCII character code. Also, two characters preceded by the dollar sign is interpreted according to the following table:

#	Character code	Description
1	\$\$	Dollar sign (\$)
2	\$"	Double quotation (")
3	\$L or \$l	Line feed (LF)
4	\$N or \$n	Carriage return line feed (CRLF)
5	\$P or \$p	Form feed
6	\$R or \$r	Carriage return (CR)
7	\$T or \$t	Tab

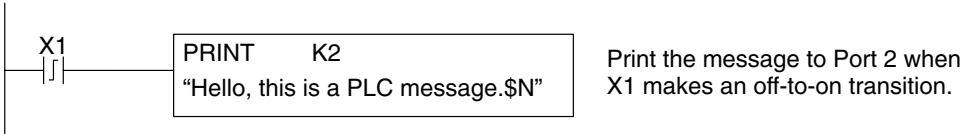
The following examples show various syntax conventions and the length of the output to the printer.

Example:

" "	Length 0 without character
"A"	Length 1 with character A
" "	Length 1 with blank
" \$"	Length 1 with double quotation mark
" \$ R \$ L "	Length 2 with one CR and one LF
" \$ 0 D \$ 0 A "	Length 2 with one CR and one LF
" \$ \$ "	Length 1 with one \$ mark

In printing an ordinary line of text, you will need to include **double quotation** marks before and after the text string. Error code 499 will occur in the CPU when the print instruction contains invalid text or no quotations. It is important to test your PRINT instruction data during the application development.

The following example prints the message to port 2. We use a PD contact, which causes the message instruction to be active for just one scan. Note the \$N at the end of the message, which produces a carriage return / line feed on the printer. This prepares the printer to print the next line, starting from the left margin.





**V-memory element** – this is used for printing V-memory contents in the integer format or real format. Use V-memory number or V-memory number with “:” and data type. The data types are shown in the table below. The Character code must be capital letters.

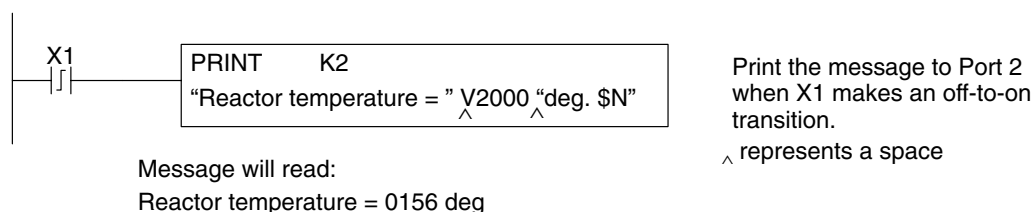
**NOTE:** There must be a space entered before and after the V-memory address to separate it from the text string. Failure to do this will result in an error code 499.

#	Character code	Description
1	none	16-bit binary (decimal number)
2	: B	4 digit BCD
3	: D	32-bit binary (decimal number)
4	: D B	8 digit BCD

Example:

V2000	Print binary data in V2000 for decimal number
V2000 : B	Print BCD data in V2000
V2000 : D	Print binary number in V2000 and V2001 for decimal number
V2000 : D B	Print BCD data in V2000 and V2001

**Example:** The following example prints a message containing text and a variable. The “reactor temperature” labels the data, which is at V2000. You can use the : B qualifier after the V2000 if the data is in BCD format, for example. The final string adds the units of degrees to the line of text, and the \$N adds a carriage return / line feed.



**V-memory text element** – this is used for printing text stored in V-memory. Use the % followed by the number of characters after V-memory number for representing the text. If you assign “0” as the number of characters, the print function will read the character count from the first location. Then it will start at the next V-memory location and read that number of ASCII codes for the text from memory.

Example:

V2000 % 16	16 characters in V2000 to V2007 are printed.
V2000 % 0	The characters in V2001 to Vxxxx (determined by the number in V2000) will be printed.

**Bit element** – this is used for printing the state of the designated bit in V-memory or a relay bit. The bit element can be assigned by the designating point (.) and bit number preceded by the V-memory number or relay number. The output type is described as shown in the table below.

#	Data format	Description
1	none	Print 1 for an ON state, and 0 for an OFF state
2	: BOOL	Print “TRUE” for an ON state, and “FALSE” for an OFF state
3	: ONOFF	Print “ON” for an ON state, and “OFF” for an OFF state

Example:

V2000 . 15	Prints the status of bit 15 in V2000, in 1/0 format
C100	Prints the status of C100 in 1/0 format
C100 : BOOL	Prints the status of C100 in TRUE/FALSE format
C100 : ON/OFF	Prints the status of C00 in ON/OFF format
V2000.15 : BOOL	Prints the status of bit 15 in V2000 in TRUE/FALSE format

The maximum numbers of characters you can print is 128. The number of characters for each element is listed in the table below:

Element type	Maximum Characters
Text, 1 character	1
16 bit binary	6
32 bit binary	11
4 digit BCD	4
8 digit BCD	8
Floating point (real number)	12
Floating point (real with exponent)	12
V-memory/text	2
Bit (1/0 format)	1
Bit (TRUE/FALSE format)	5
Bit (ON/OFF format)	3

The handheld programmer’s mnemonic is “PRINT”, followed by the DEF field.

Special relay flags SP116 and SP117 indicate the status of the DL05 CPU ports (busy, or communications error). See the appendix on special relays for a description.

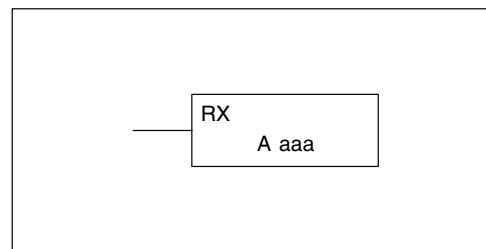
**NOTE:** You must use the appropriate special relay in conjunction with the PRINT command to ensure the ladder program does not try to PRINT to a port that is still busy from a previous PRINT or WX or RX instruction.



## Network Instructions

### Read from Network (RX)

The Read from Network instruction causes the master device on a network to read a block of data from a slave device on the same network. The function parameters are loaded into the accumulator and the first and second level of the stack. Listed below are the program steps necessary to execute the Read from Network function.



Step 1: — Load the slave address (0–90 BCD) into the low byte and “F2” into the high byte of the accumulator (the next two instructions push this word down to the second layer of the stack).

Step 2: — Load the number of bytes to be transferred into the accumulator (the next instruction pushes this word onto the top of the stack).

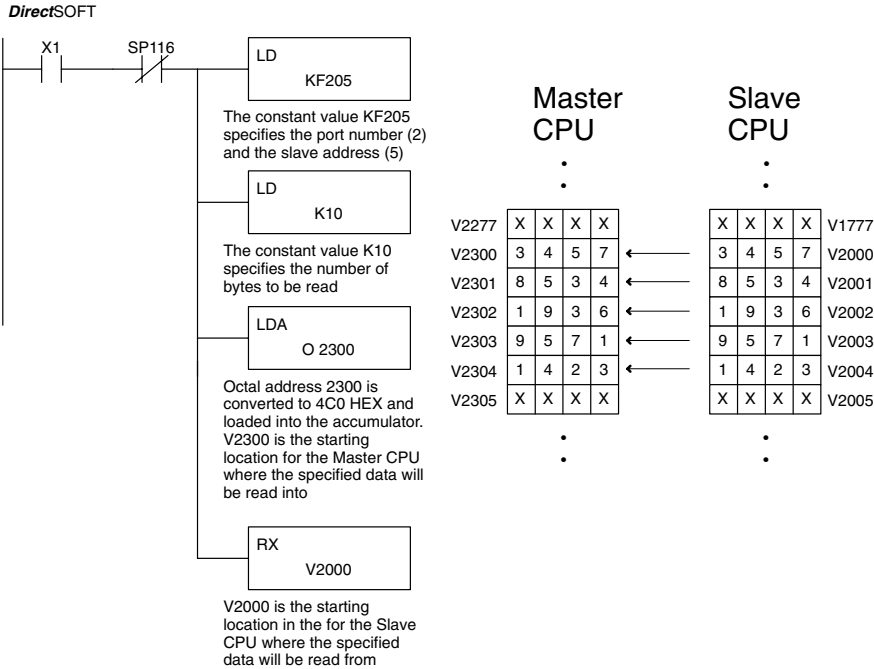
Step 3: — Load the starting Master CPU address into the accumulator. This is the memory location where the data read from the slave will be put. This parameter requires a HEX value.

Step 4: — Insert the RX instruction which specifies the starting V memory location (Aaaa) where the data will be read from in the slave.

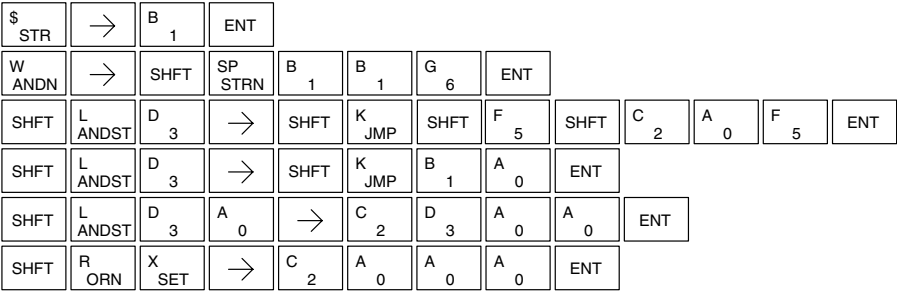
Helpful Hint: — For parameters that require HEX values, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL05 Range
A		aaa
V memory	V	All (See page 4–28)
Pointer	P	All V mem. (See page 4–28)
Inputs	X	0–377
Outputs	Y	0–377
Control Relays	C	0–777
Stage	S	0–377
Timer	T	0–177
Counter	CT	0–177
Special Relay	SP	0–777
Program Memory	\$	0–2047 (2K program mem.)

In the following example, when X1 is on and the port busy relay SP116 (see special relays) is not on, the RX instruction will access port 2 operating as a master. Ten consecutive bytes of data (V2000 – V2004) will be read from a CPU at station address 5 and copied into V memory locations V2300–V2304 in the CPU with the master port.



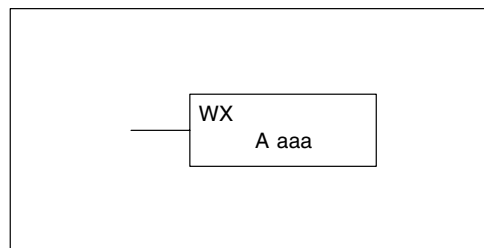
Handheld Programmer Keystrokes





**Write to Network  
(WX)**

The Write to Network instruction is used to write a block of data from the master device to a slave device on the same network. The function parameters are loaded into the accumulator and the first and second level of the stack. Listed below are the program steps necessary to execute the Write to Network function.



Step 1: — Load the slave address (0–90 BCD) into the low byte and “F2” into the high byte of the accumulator (the next two instructions push this word down to the second layer of the stack).

Step 2: — Load the number of bytes to be transferred into the accumulator (the next instruction pushes this word onto the top of the stack).

Step 3: — Load the starting Master CPU address into the accumulator. This is the memory location where the data will be written from. This parameter requires a HEX value.

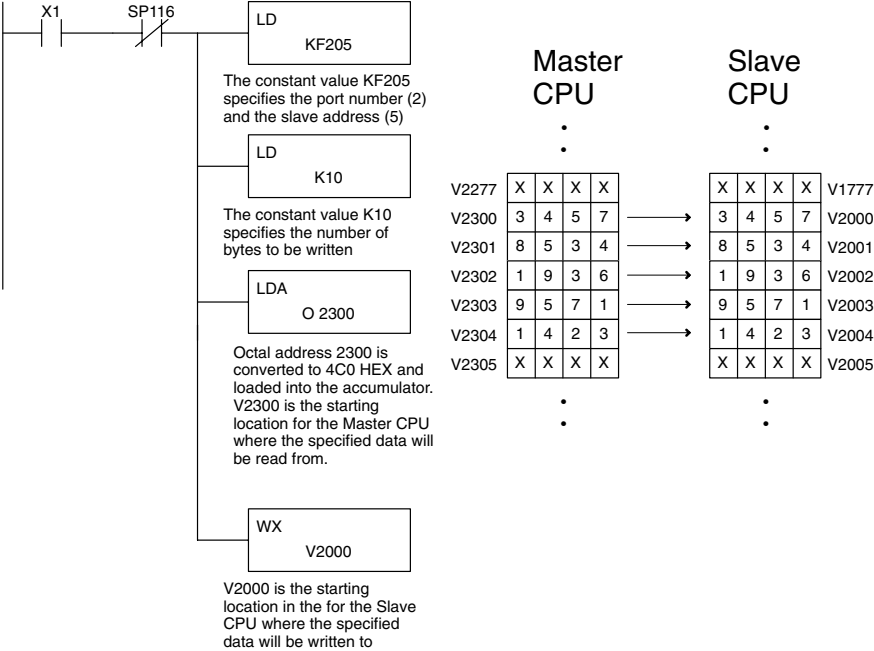
Step 4: — Insert the WX instruction which specifies the starting V memory location (Aaaa) where the data will be written to in the slave.

Helpful Hint: — For parameters that require HEX values, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

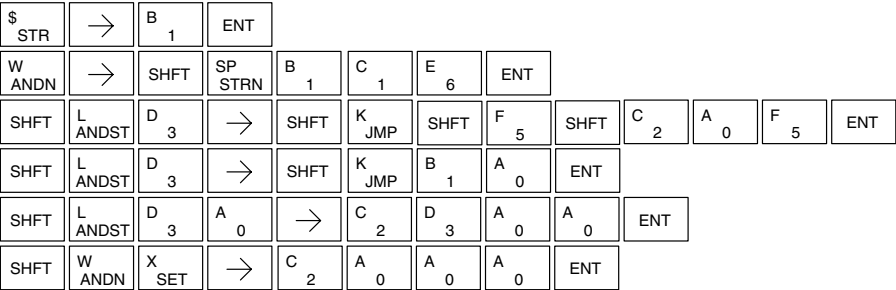
Operand Data Type		DL05 Range
	<b>A</b>	<b>aaa</b>
V memory	V	All (See page 4–28)
Pointer	P	All V mem. (See page 4–28)
Inputs	X	0–377
Outputs	Y	0–377
Control Relays	C	0–777
Stage	S	0–377
Timer	T	0–177
Counter	CT	0–177
Special Relay	SP	0–777
Program Memory	\$	0–2048 (2K program mem.)

In the following example when X1 is on and the module busy relay SP116 (see special relays) is not on, the WX instruction will access port 2 operating as a master. Ten consecutive bytes of data is read from the Master CPU and copied to V memory locations V2000–V2004 in the slave CPU at station address 5.

DirectSOFT



Handheld Programmer Keystrokes



# Drum Instruction Programming

---

In This Chapter. . . .

- Introduction
- Step Transitions
- Overview of Drum Operation
- Drum Control Techniques
- Drum Instruction
- EDrum Instruction

## Introduction

### Purpose

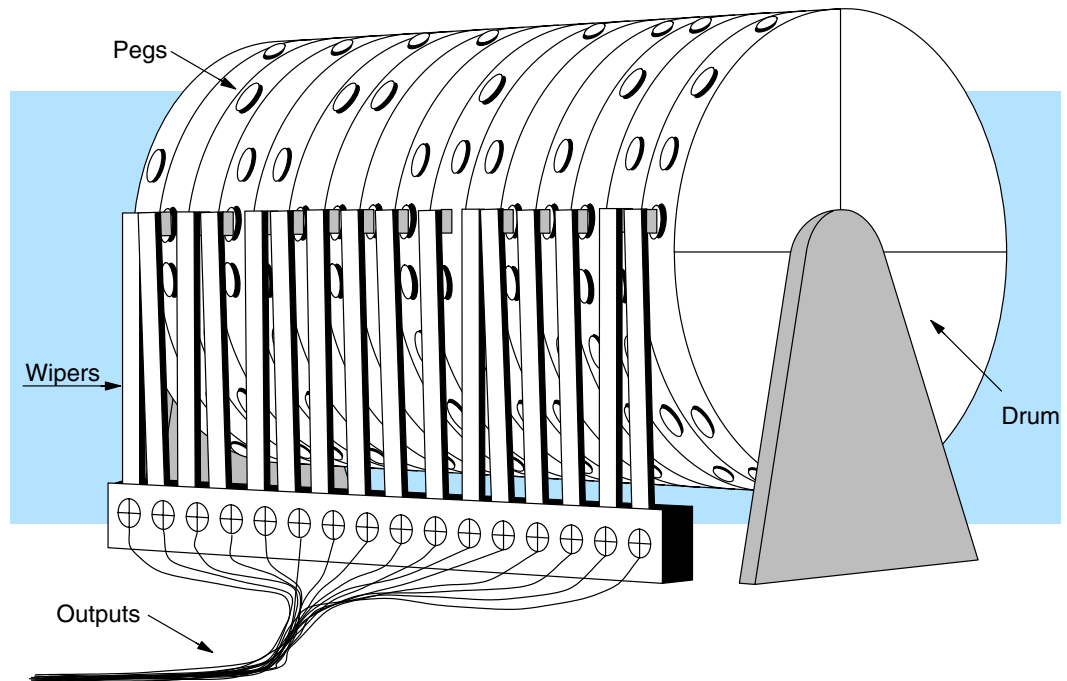
The Event Drum (EDRUM) instruction in the DL05 CPU electronically simulates an electro-mechanical drum sequencer. The instruction offers enhancements to the basic principle, which we describe first.

### 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.



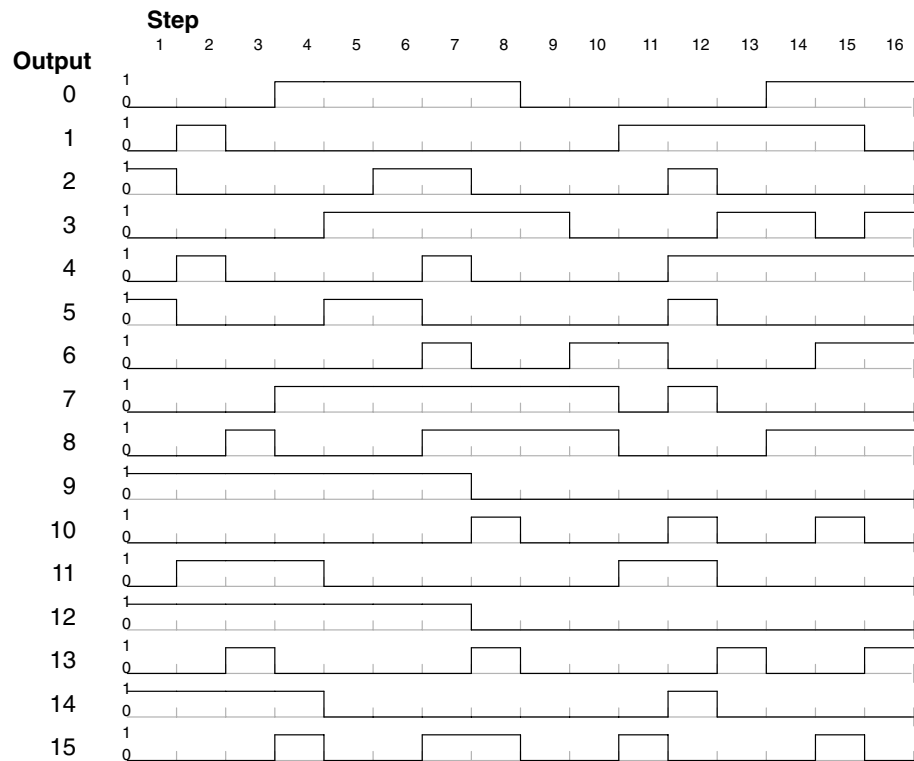
Electronic drums provide the benefits of mechanical drums and more. For example, they have a **preset** feature that is impossible for mechanical drums: The preset function lets you move from the present step *directly* to any other step on command!

## Drum Chart Representation

For editing purposes, the electronic drum is presented in chart form in **DirectSOFT** and in this manual. Imagine slicing the surface of a hollow drum cylinder between two rows of pegs, then pressing it flat. Now you can view the drum as a chart as shown below. Each row represents a step, numbered 1 through 16. Each column represents an output, numbered 0 through 15 (to match word bit numbering). The solid circles in the chart represent pegs (On state) in the mechanical drum, and the open circles are empty peg sites (Off state).

STEP	OUTPUTS															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○
2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
3	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
4	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
5	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
6	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
7	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
8	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
9	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
10	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
11	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
12	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
13	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
14	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
15	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
16	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

**Output Sequences** The mechanical drum sequencer derives its name from sequences of control changes on its electrical outputs. The following figure shows the sequence of On/Off controls generated by the drum pattern above. Compare the two, and you will find 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

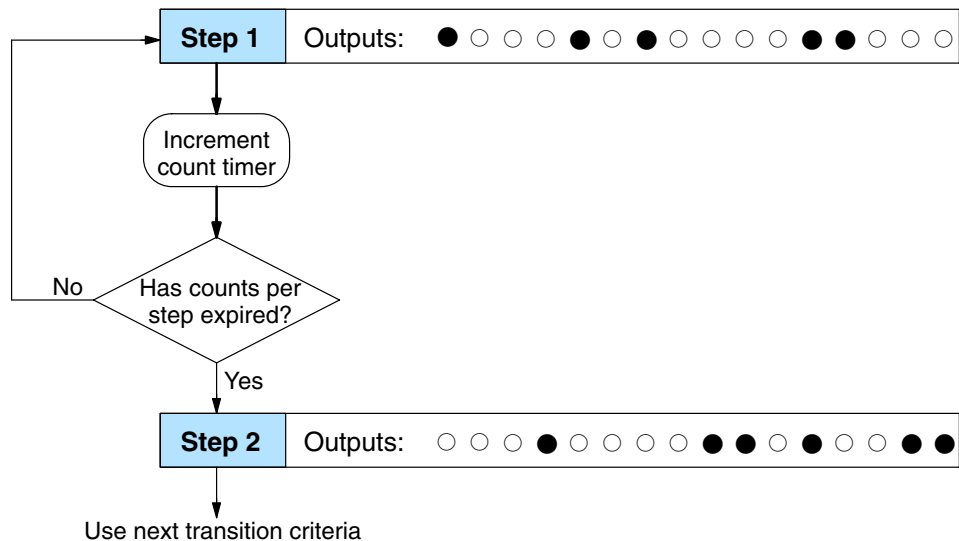
There are two types of Drum instructions in the DL05 CPU:

- 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 (○= Off, ●= On) as shown. When programming a drum instruction, you also determine both the output assignment and the On/Off state (pattern) at that time. All steps use the same output assignment, but each step may have its own unique output pattern.

### Timer-Only Transitions

Drums move from 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:

$$\text{Time in step} = 0.01 \text{ seconds} \times \text{Timebase} \times \text{Counts per step}$$

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:

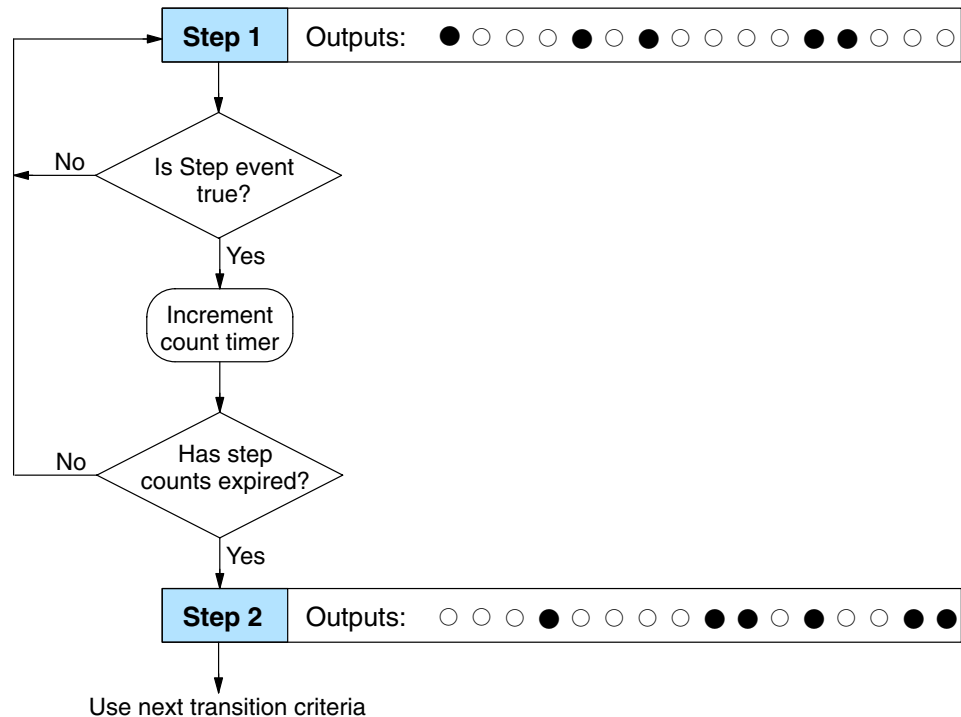
$$\begin{aligned}\text{Max Time per step} &= 0.01 \text{ seconds} \times 9999 \times 9999 \\ &= 999,800 \text{ seconds} = 277.7 \text{ hours} = 11.6 \text{ days}\end{aligned}$$



**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

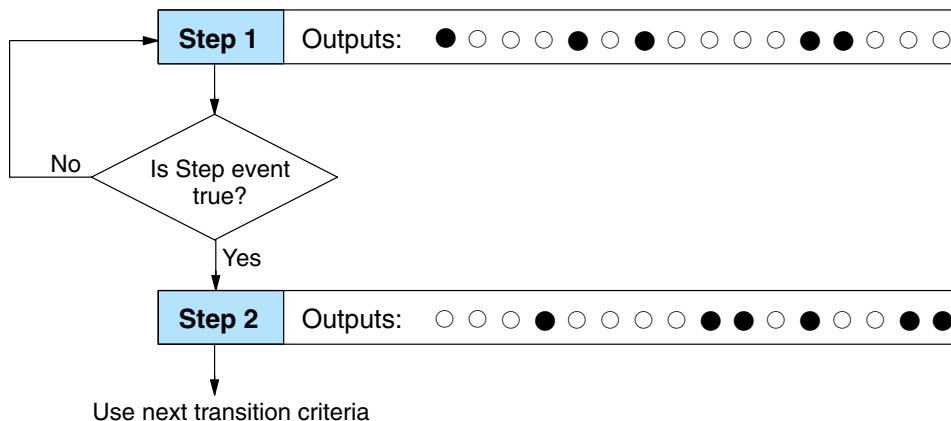
Step transitions may also occur based on time and/or external events. The figure below shows how step transitions work in these cases.



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**

<b>CT10</b>	Counts in step	V1010	1528
<b>CT11</b>	Timer Value	V1011	0200
<b>CT12</b>	Preset Step	V1012	0001
<b>CT13</b>	Current Step	V1013	0004

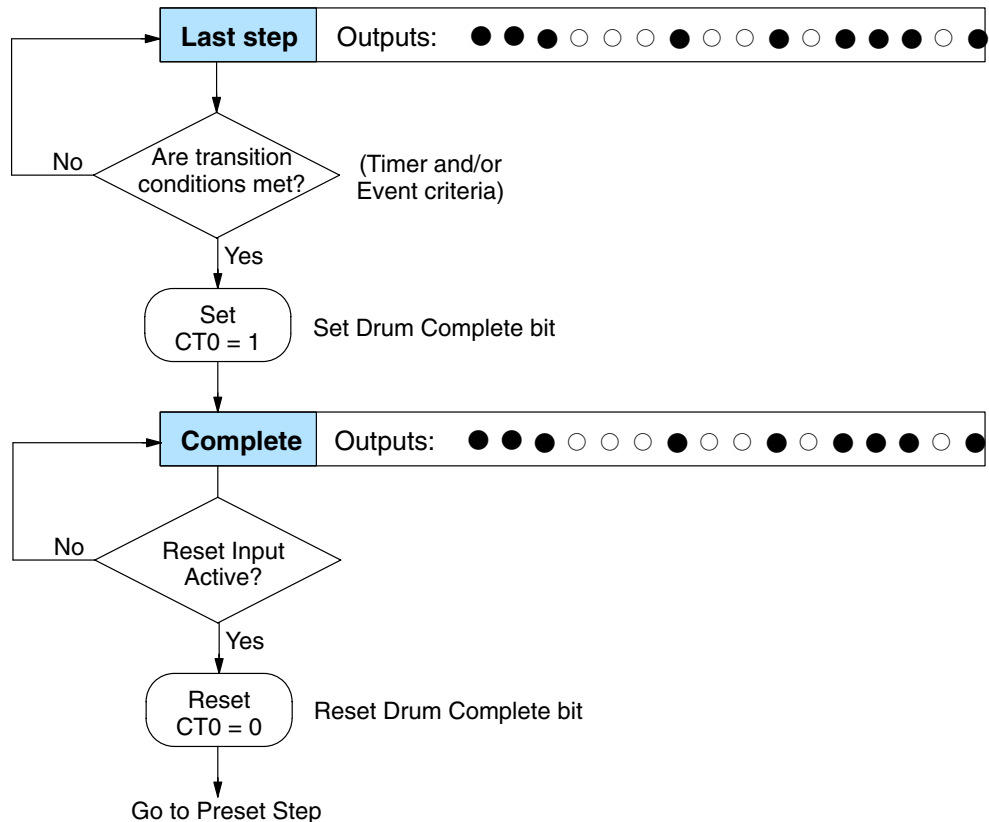
CT10 shows that we are at the 1528th count in the current step, which is step 4 (shown in CT13). If we have programmed step 4 to have 3000 counts, then the step is just over half completed. CT11 is the count timer, shown in units of 0.01 seconds. So, each least-significant-digit change represents 0.01 seconds. The value of 200 means that we have been in the current count (1528) for 2 seconds (0.01 x 100). Finally, CT12 holds the preset step value which was programmed into the drum instruction. When the drum's Reset input is active, it presets to step 1 in this case. The value of CT12 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 CT0). 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.

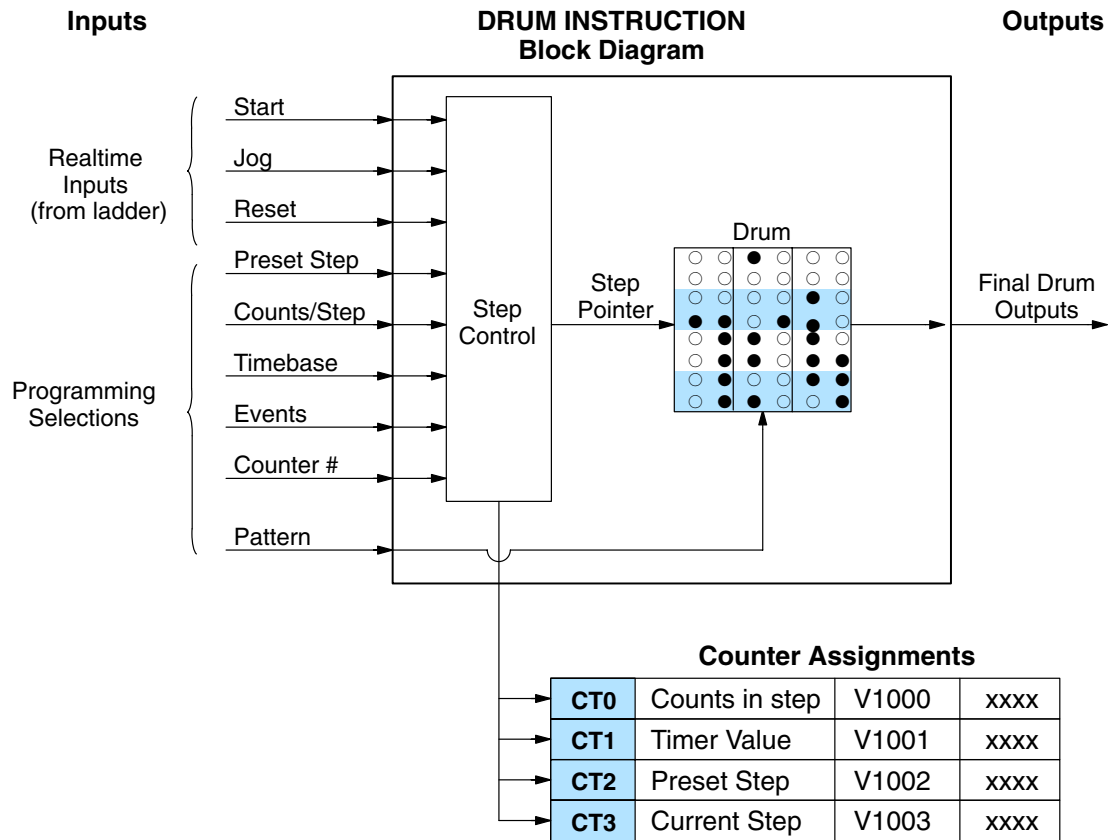
The drum leaves the “drum complete” state when the Reset input becomes active (or on a program-to-run mode transition). It resets the drum complete bit (such as CT0), and then goes directly to the appropriate step number defined as the preset step.



## Overview of Drum Operation

### Drum Instruction Block Diagram

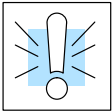
The drum instruction utilizes various inputs and outputs in addition to the drum pattern itself. Refer to the figure below.



The drum instruction accepts several inputs for step control, the main control of the drum. The inputs and their functions are:

- **Start** – The Start input is effective only when Reset is off. When Start is on, the drum timer runs if it is in a timed transition, and the drum looks for the input event during event transitions. When Start is off, the drum freezes in its current state (Reset must remain off), and the drum outputs maintain their current on/off pattern.
- **Jog** – The jog input is only effective when Reset is off (Start may be either on or off). The jog input increments the drum to the next step on each off-to-on transition (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 DL05 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.

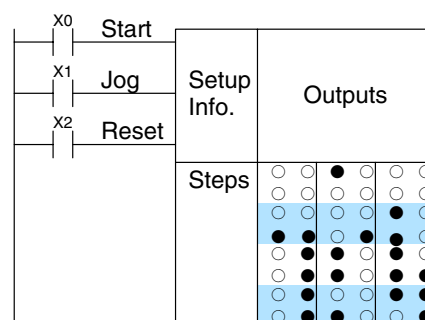
### Powerup State of Drum Registers

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
CT(n)	Current Step Count	Initialize = 0	Use Previous (no change)
CT(n + 1)	Counter Timer Value	Initialize = 0	Use Previous (no change)
CT(n + 2)	Preset Step	Initialize = Preset Step #	Use Previous (no change)
CT(n + 3)	Current Step #	Initialize = Preset Step #	Use Previous (no change)

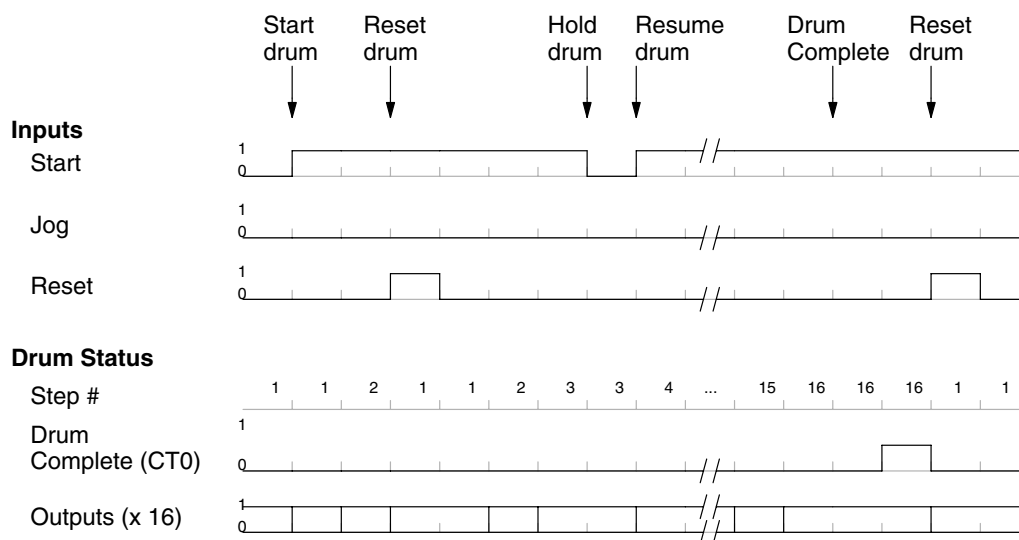
Applications with relatively fast drum cycle times typically will need to be reset on powerup, using the non-retentive option. Applications with relatively long drum cycle times may need to resume at the previous point where operations stopped, using the retentive case. The default option is the retentive case. This means that if you initialize scratchpad V-memory, the memory will be retentive.

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 (CT0, for example) indicates the drum cycle is done.



The timing diagram below shows an arbitrary timer drum input sequence and how the drum responds. As the CPU enters Run mode it initializes the step number to the preset step number (typically 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.

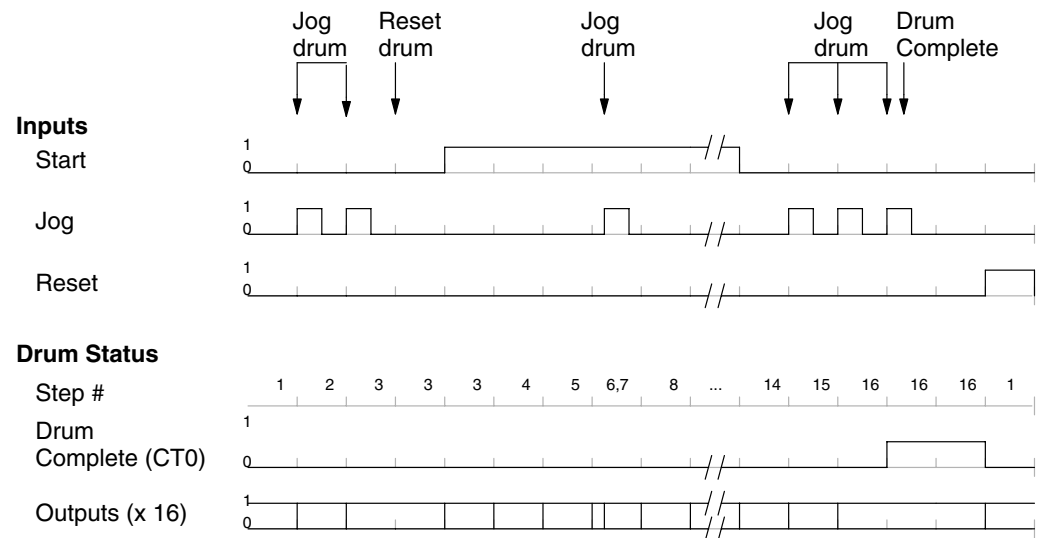


When the drum completes the last step (Step 16 in this example), the Drum Complete bit (CT0) turns on, and the step number remains at 16. When the Reset input turns on, it turns off the Drum Complete bit (CT0), and forces the drum to enter the preset step.

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

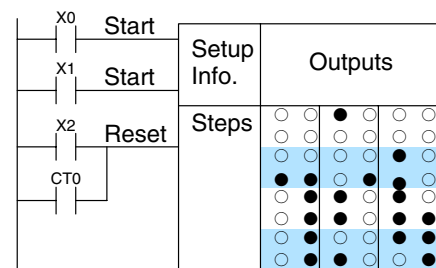
In the figure below, we focus on how the Jog input works on event drums. To the left of the diagram, note that the off-to-on transitions of the Jog input increments the step. Start may be either on or off (however, Reset must be off). Two jogs takes the drum to step three. Next, the Start input turns 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 CT0, so we logically OR the drum complete bit (CT0) with the Reset input. When the last step is done, the drum turns on CT0 which resets itself to the preset step, also resetting CT0. Contact 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

### Timed Drum with Discrete Outputs (DRUM)

The DL05 drum instructions may be programmed using **DirectSOFT** or for the EDRUM instruction only you can use a handheld programmer (firmware version v1.8 or later. This section covers entry using **DirectSOFT** for all instructions plus the handheld mnemonics for the EDRUM instruction.

The Timed Drum with Discrete Outputs is the most basic of the DL05's drum instructions. It operates according to the principles covered on the previous pages. Below is the instruction in chart form as displayed by **DirectSOFT**.

		Counter Number	Step Preset	Timebase	Discrete Output Assignment															
		DRUM	CT aaa	15	0															
Control Inputs	Start	Step Preset	K bb	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)
	Reset	0.01 sec/Count	K cccc	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)	(Ffff)
		Step #	Counts																	
Step Number	1	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	2	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Counts per Step	3	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	4	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Output Pattern	5	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	6	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	7	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	8	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	9	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	10	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	11	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	12	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	13	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	14	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	15	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	16	Kdddd		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

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 **DirectSOFT**.

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.

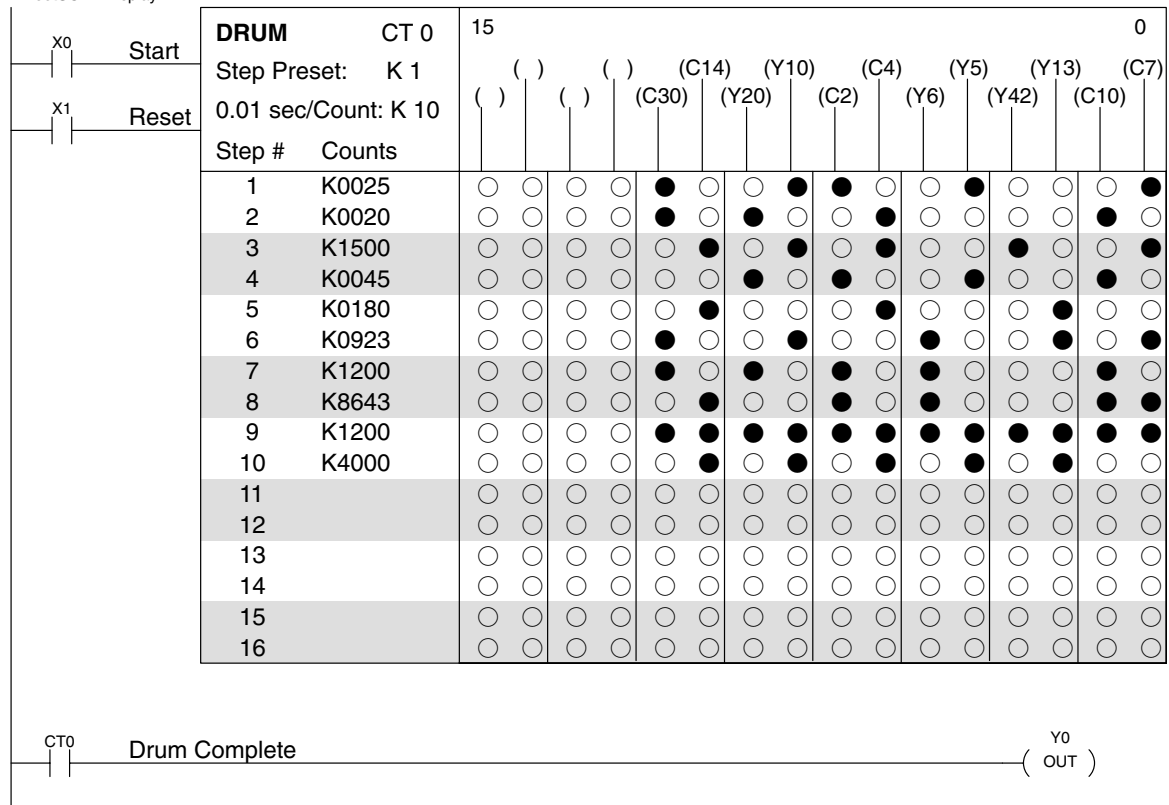
Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa	—	0 – 174
Preset Step	bb	K	1 – 16
Timer base	cccc	K	0 – 99.99 seconds
Counts per step	dddd	K	0 – 9999
Discrete Outputs	Ffff	X, Y, C	see page 4-28

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CT(n+2) at any time. However, the other counters are for monitoring purposes only.

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

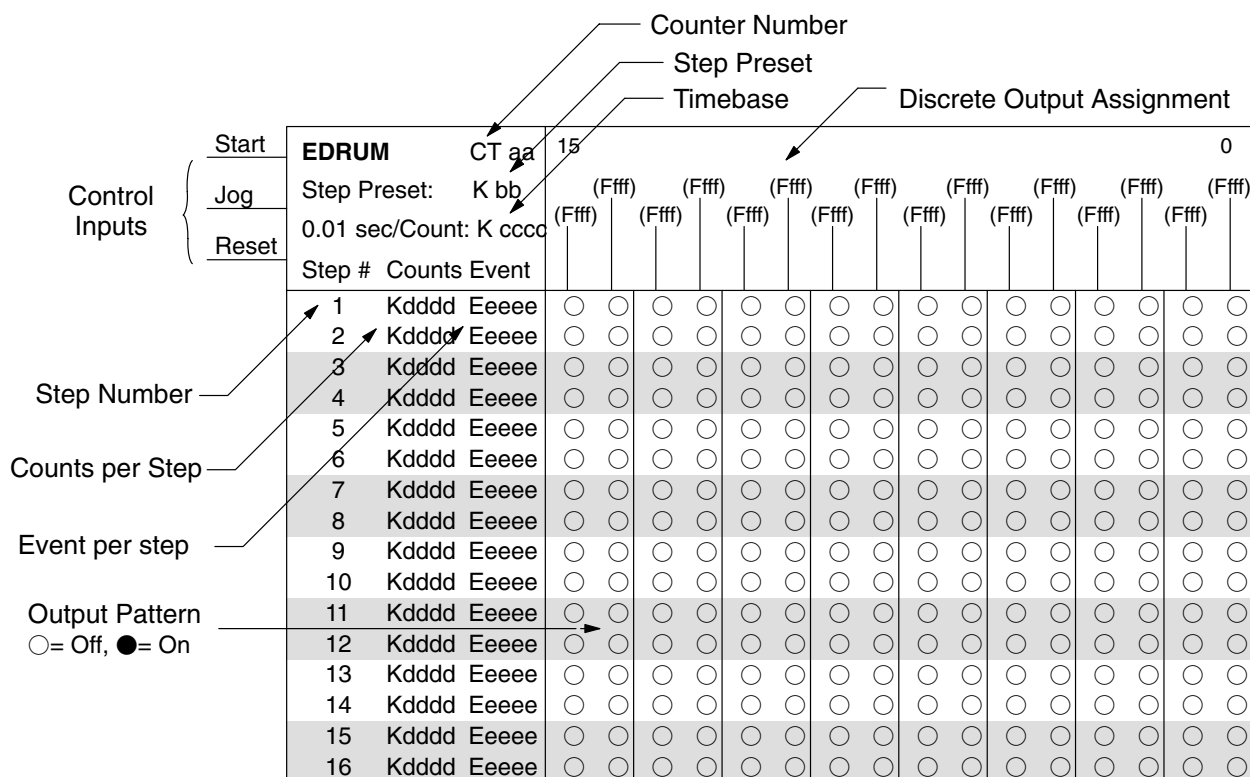
The following ladder program shows the DRUM instruction in a typical ladder program, as shown by **DirectSOFT**. Steps 1 through 10 are used, and twelve of the sixteen output points are used. The preset step is step 1. The timebase runs at  $(K10 \times 0.01) = 0.1$  second per count. Therefore, the duration of step 1 is  $(25 \times 0.1) = 2.5$  seconds. In the last rung, the Drum Complete bit (CT0) turns on output Y0 upon completion of the last step (step 10). A drum reset also resets CT0.

DirectSOFT Display



## Event Drum (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 **DirectSOFT**.



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.

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	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	eeee	X, Y, C, S, T, CT	see page 4-28
Discrete Outputs	ffff	X, Y, C	see page 4-28

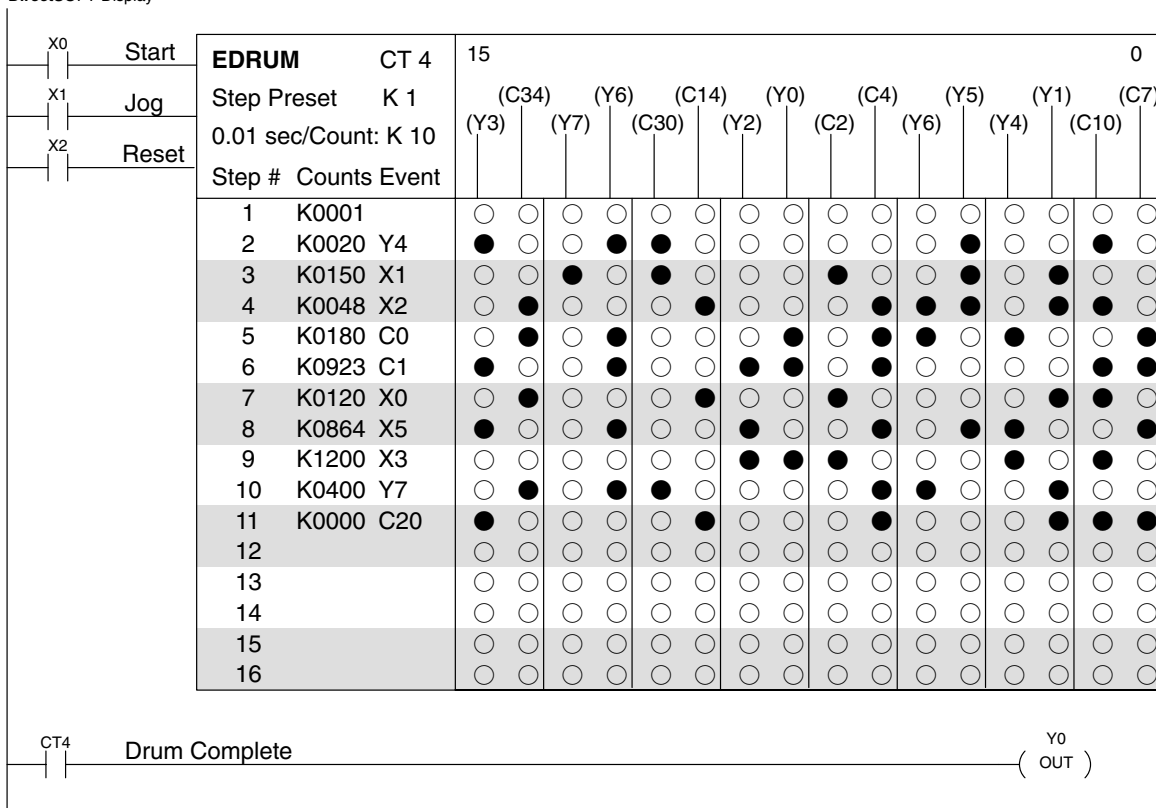


Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CT(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CT(n)	0 – 174	Counts in step	CTn = Drum Complete
C( n+1)	1 – 175	Timer value	CT(n+1) = (not used)
CT( n+2)	2 –176	Preset Step	CT(n+2) = (not used)
CT( 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 **DirectSOFT**. Steps 1 through 11 are used, and all sixteen output points are used. The preset step is step 1. The timebase runs at  $(K10 \times 0.01) = 0.1$  second per count. Therefore, the duration of step 1 is  $(1 \times 0.1) = 0.1$  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.

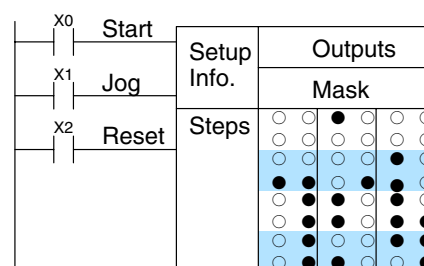
DirectSOFT Display



The EDRUM instruction may be programmed using either **DirectSOFT** or a handheld programmer. This section covers entry via the handheld programmer (Refer to the **DirectSOFT** 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    \$<sub>STR</sub>    →    A<sub>0</sub>    ENT

(Repeat for Store X1 and Store X2)

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

## Handheld Programmer Keystrokes

EDRUM CNT0	SHFT	E <sub>1</sub>	D <sub>3</sub>	R <sub>ORN</sub>	U <sub>ISG</sub>	M <sub>ORST</sub>	→	A <sub>0</sub>	ENT
------------	------	----------------	----------------	------------------	------------------	-------------------	---	----------------	-----

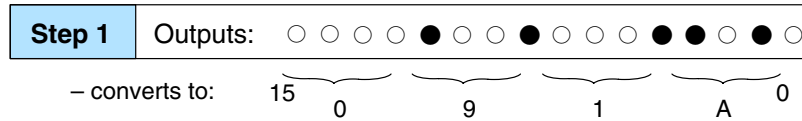
After entering the EDRUM mnemonic as above, the handheld programmer creates an input form for all the drum parameters. The input form consists of approximately fifty or more default mnemonic entries containing DEF (define) statements. The default mnemonics are already “input” for you, so they appear 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.

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 page 4–28
Counts per step	16	dddd	DEF K0000	K	0 – 9999
Events	16	dddd	DEF K0000	X, Y, C, S, T, CT	see page 4–28
Output pattern	16	gggg	DEF K0000	K	0 – FFFF

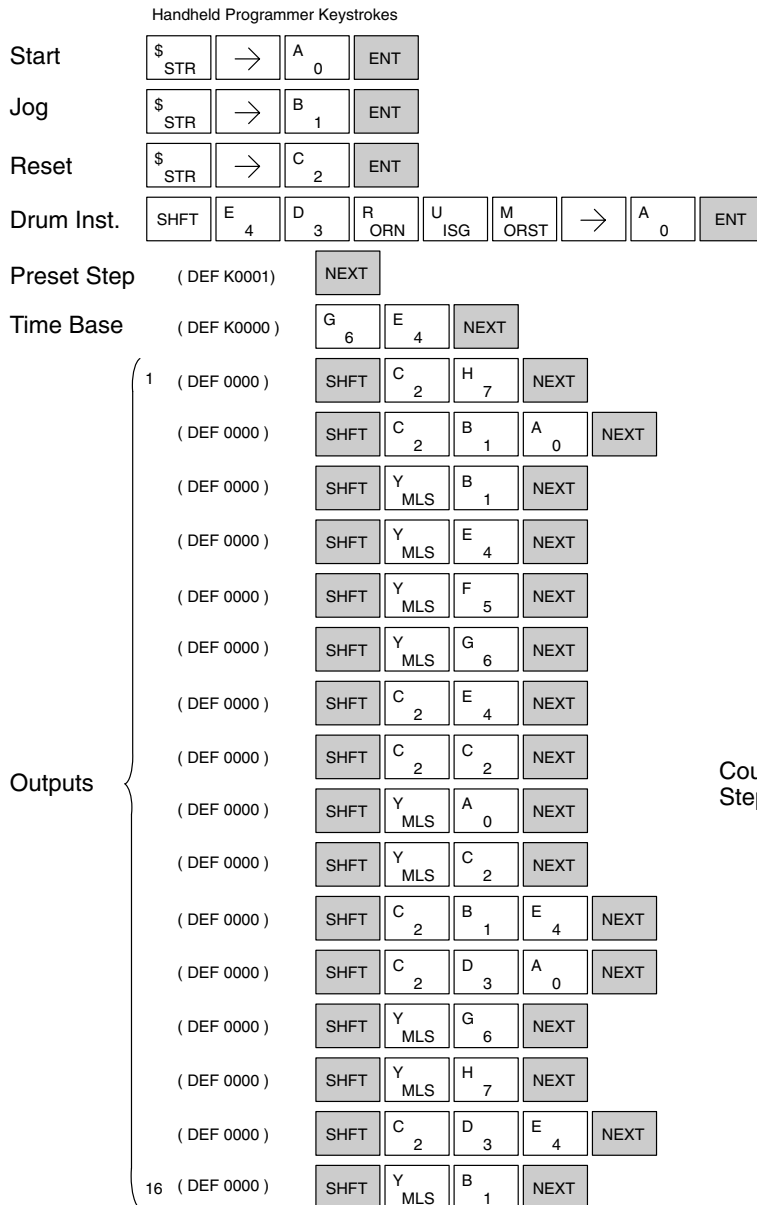
**NOTE:** Default entries for output points and events are “DEF 0000”, which means they are unassigned. If you need to go back and change an assigned output as unused again, enter “K0000”. The entry will again show as “DEF 0000”.



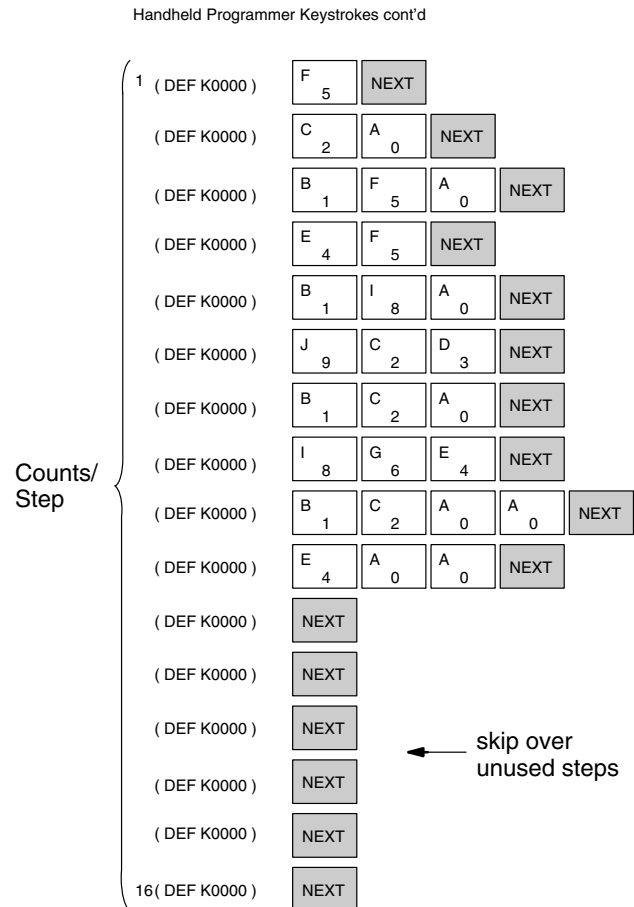
Using the DRUM entry chart (two pages before), we show the method of entry for the basic time/event drum instruction. First, we convert the output pattern for each step to the equivalent hex number, as shown in the following example.



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 1.7 or later.



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



(Continued on next page)

Handheld Programmer Keystrokes cont'd

Events

1	(DEF 0000)	NEXT	← skip over unused event
	(DEF 0000)	SHFT Y MLS E 4 NEXT	
	(DEF 0000)	SHFT X SET B 1 NEXT	
	(DEF 0000)	SHFT X SET C 2 NEXT	
	(DEF 0000)	SHFT C 2 A 0 NEXT	
	(DEF 0000)	SHFT C 2 B 1 NEXT	
	(DEF 0000)	SHFT X SET A 0 NEXT	
	(DEF 0000)	SHFT X SET F 5 NEXT	
	(DEF 0000)	SHFT X SET D 3 NEXT	
	(DEF 0000)	SHFT Y MLS H 7 NEXT	
	(DEF 0000)	SHFT C 2 C 2 A 0 NEXT	
	(DEF 0000)	NEXT	
	(DEF 0000)	NEXT	
	(DEF 0000)	NEXT	
	(DEF 0000)	NEXT	
16	(DEF 0000)	NEXT	

Handheld Programmer Keystrokes cont'd

Output Pattern

1	(DEF K0000)	NEXT	← step 1 pattern = 0000
	(DEF K0000)	J 9 I 8 B 1 C 2 NEXT	
	(DEF K0000)	C 2 I 8 J 9 E 4 NEXT	
	(DEF K0000)	E 4 E 4 H 7 G 6 NEXT	
	(DEF K0000)	F 5 B 1 G 6 J 9 NEXT	
	(DEF K0000)	J 9 D 3 E 4 D 3 NEXT	
	(DEF K0000)	E 4 E 4 I 8 G 6 NEXT	
	(DEF K0000)	J 9 E 4 F 5 J 9 NEXT	
	(DEF K0000)	D 3 I 8 SHFT A 0 NEXT	
	(DEF K0000)	F 5 I 8 G 6 E 4 NEXT	
	(DEF K0000)	I 8 E 4 E 4 H 7 NEXT	
	(DEF K0000)	NEXT	
	(DEF K0000)	NEXT	
	(DEF K0000)	NEXT	← unused steps
	(DEF K0000)	NEXT	
16	(DEF K0000)	NEXT	

Last rung

\$ STR	GY CNT	A 0	NEXT
SHFT	Y MLS	A 0	NEXT

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

**RLL *PLUS***

# Stage Programming

---

In This Chapter. . . .

- Introduction to Stage Programming
  - Learning to Draw State Transition Diagrams
  - Using the Stage Jump Instruction for State Transitions
  - Stage Program Example: Toggle On/Off Lamp Controller
  - Four Steps to Writing a Stage Program
  - Stage Program Example: A Garage Door Opener
  - Stage Program Design Considerations
  - RLL<sup>PLUS</sup> (Stage) Instructions
  - Questions and Answers about Stage Programs
-

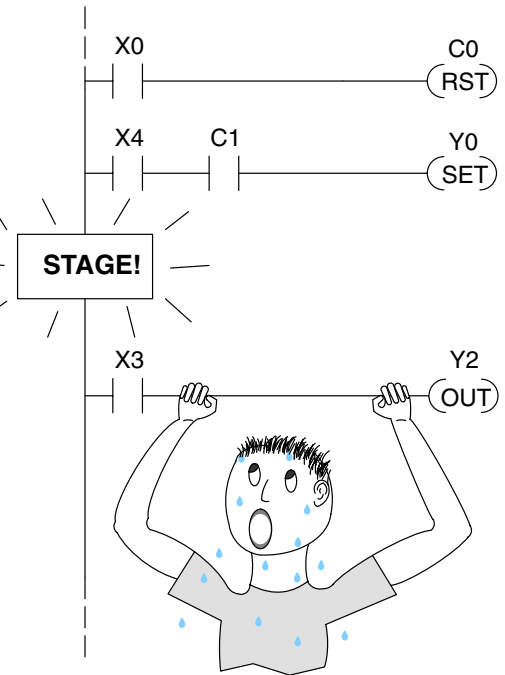
## Introduction to Stage Programming

### Overcoming “Stage Fright”

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 <sup>PLUS</sup>. 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.

Many PLC programmers in the industry have become comfortable using RLL for every PLC program they write... but often remain skeptical or even fearful of learning new techniques such as stage programming. While RLL is great at solving boolean logic relationships, it has disadvantages as well:

- Large programs can become almost unmanageable, because of a lack of structure.
- In RLL, latches must be tediously created from self-latching relays.
- When a process gets stuck, it is difficult to find the rung where the error occurred.
- Programs become difficult to modify later, because they do not intuitively resemble the application problem they are solving.



It's easy to see that these inefficiencies consume a lot of additional time, and time is money. *Stage programming overcomes these obstacles!* We believe a few moments of studying the stage concept is one of the greatest investments in programming speed and efficiency a PLC programmer can make!

So, we encourage you to study stage programming and add it to your “toolbox” of programming techniques. This chapter is designed as a self-paced tutorial on stage programming. For best results:

- Start at the beginning and do not skip over any sections.
- Study each stage 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

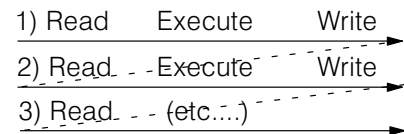
Those familiar with ladder program execution know that the CPU must scan the ladder program repeatedly, over and over. Its three basic steps are:

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.



PLC Scan



Most manufacturing processes consist of a series of activities or conditions, each lasting for several seconds, minutes, or even hours. We might call these “process states”, which are either active or inactive at any particular time. A challenge for RLL programs is that a particular input event may last for 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.

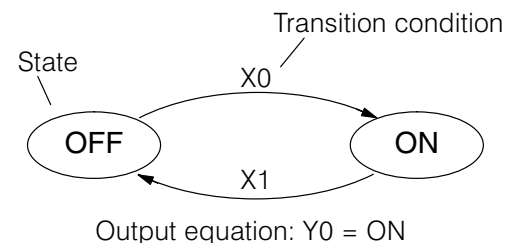
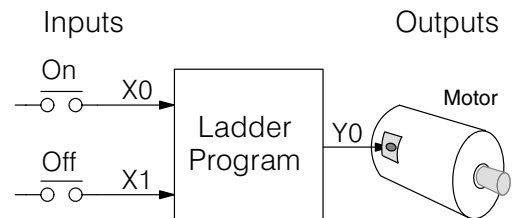
### The Need for State Diagrams

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

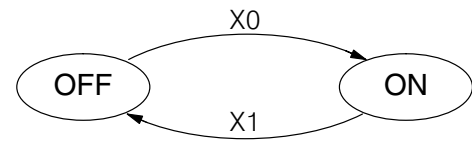
Consider the simple process shown to the right, which controls an industrial motor. We will use a green momentary SPST pushbutton to turn the motor on, and a red one to turn it off. The machine operator will press the appropriate pushbutton for just a second or so. The two states of our process are ON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



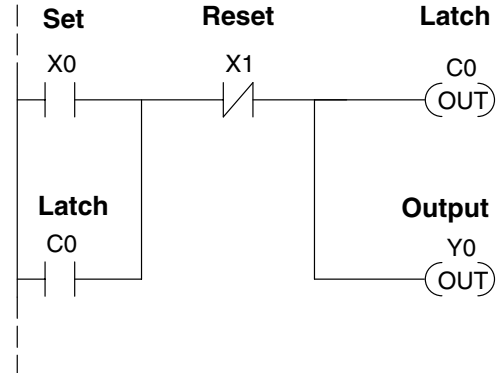
Output equation:  $Y0 = ON$

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.

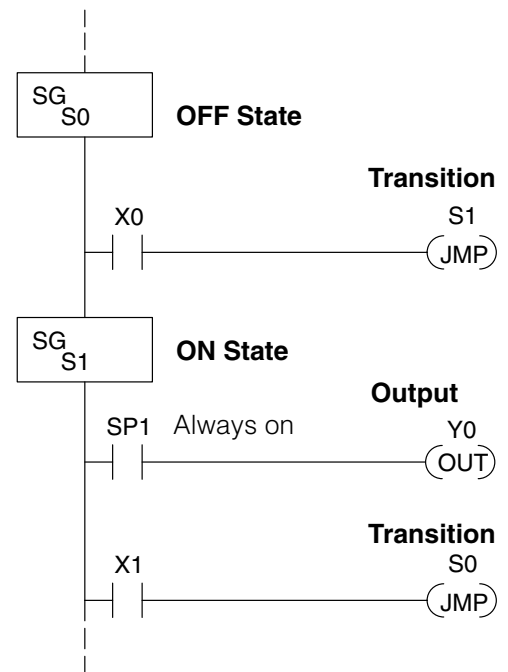


### Stage Equivalent

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.



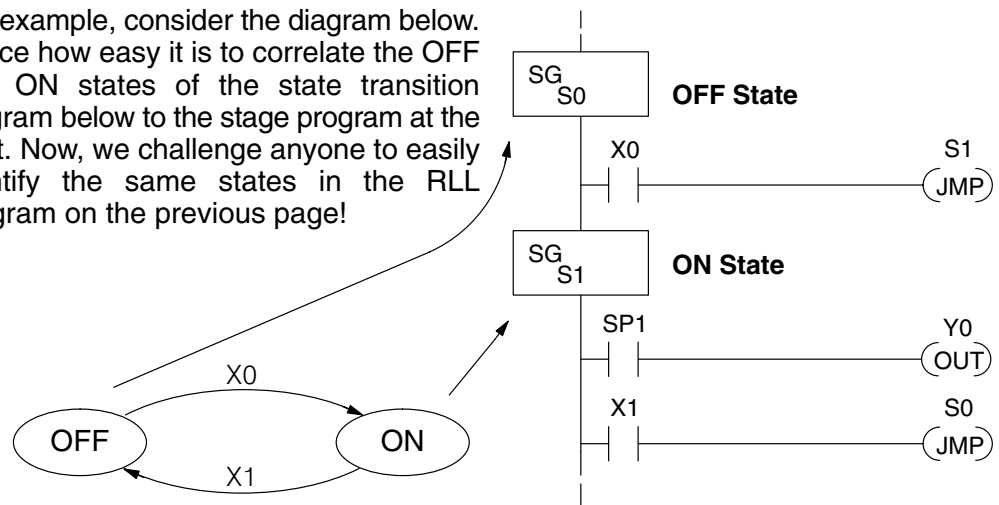
When the Off pushbutton (X1) is pressed, a transition back to the Off State occurs. The JMP S0 instruction executes, which simply turns off the Stage bit S1 and turns on Stage bit S0. On the next PLC scan, the CPU will not execute Stage S1, so the motor output Y0 will turn off. The Off state (Stage 0) 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!

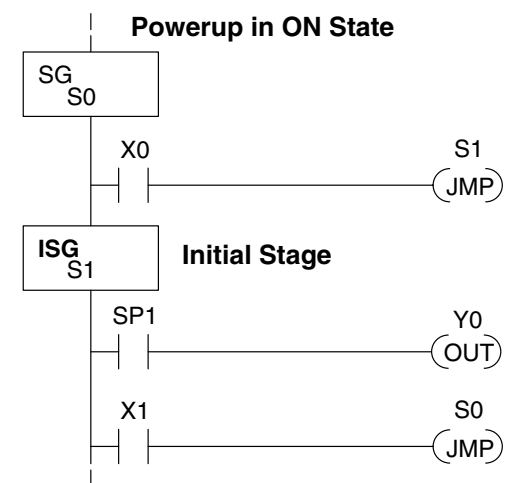
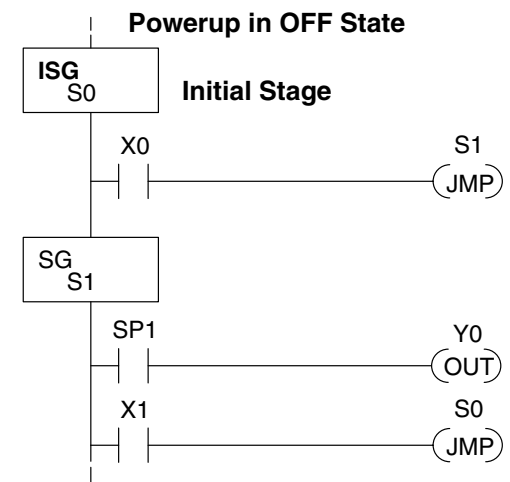
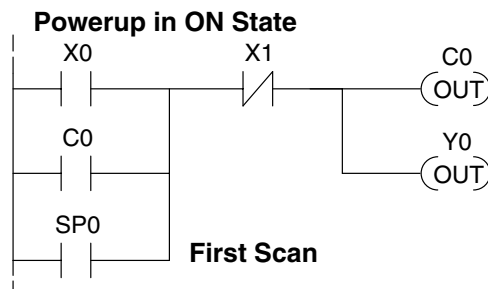


## Initial Stages

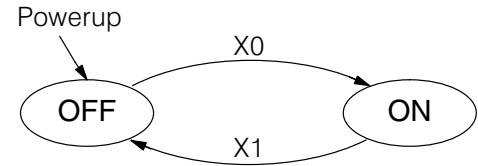
At powerup and Program-to-Run Mode transitions, the PLC always begins with all normal stages (SG) off. So, the stage programs shown so far have actually had no way to get started (because rungs are not scanned unless their stage is active).

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.



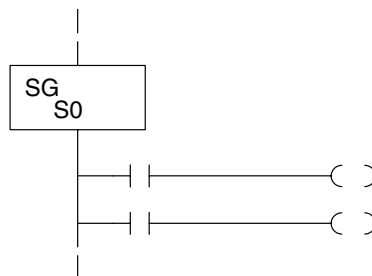
We can mark our desired powerup state as shown to the right, which helps us remember to use the appropriate Initial Stages when creating a stage program. It is permissible to have as many initial stages as the process requires.



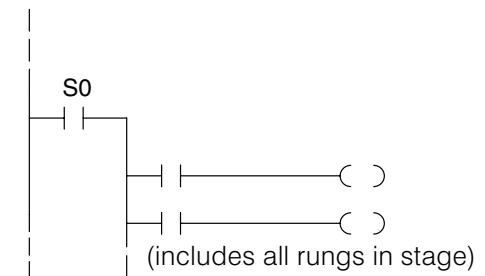
**What Stage Bits Do** You may recall that a stage is just a section of ladder program which is either active or inactive at a given moment. All stage bits (S0 to S377) reside in the PLC's image register as individual status bits. Each stage bit is either a boolean 0 or 1 at any time. Program execution 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).

#### Actual Program Appearance



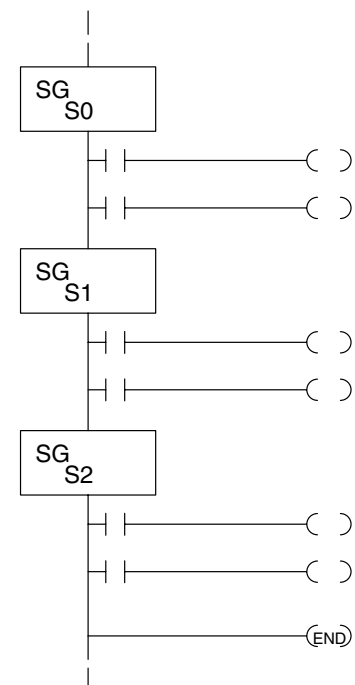
#### Functionally Equivalent Ladder



#### Stage Instruction Characteristics

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

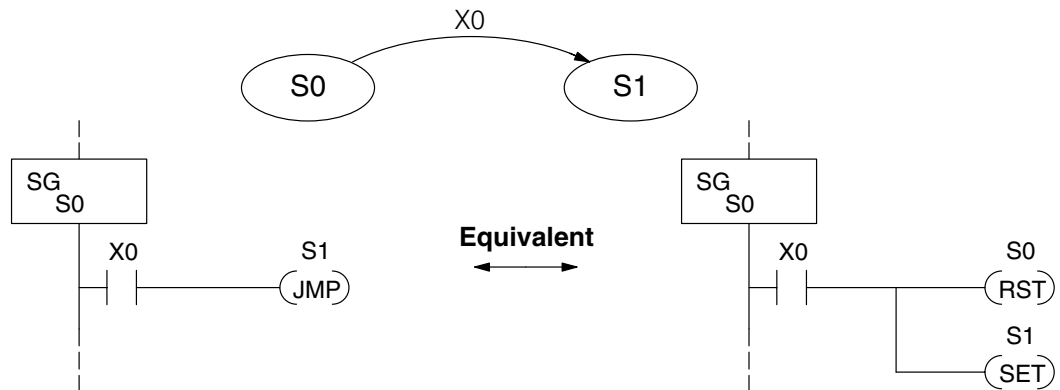
- **Execution** – Only logic in active stages are executed on any scan.
- **Transitions** – Stage transition instructions take effect on the next occurrence of the stages involved.
- **Octal numbering** – Stages are numbered in octal, like I/O points, etc. So “S8” is not valid.
- **Total Stages** – The DL05 offers up to 256 stages (S0 to S377 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

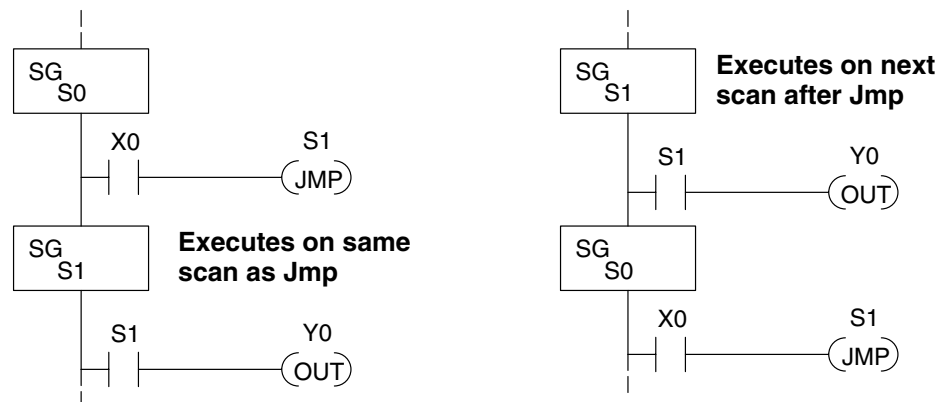
### Stage Jump, Set, and Reset Instructions

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.

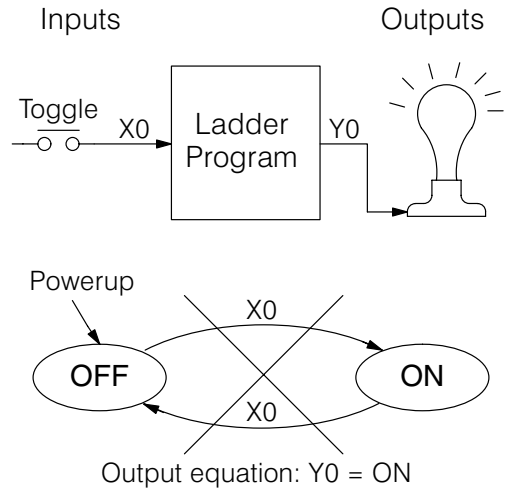


Note: Assume we start with Stage 0 active and stage 1 inactive for both examples.

## Stage Program Example: Toggle On/Off Lamp Controller

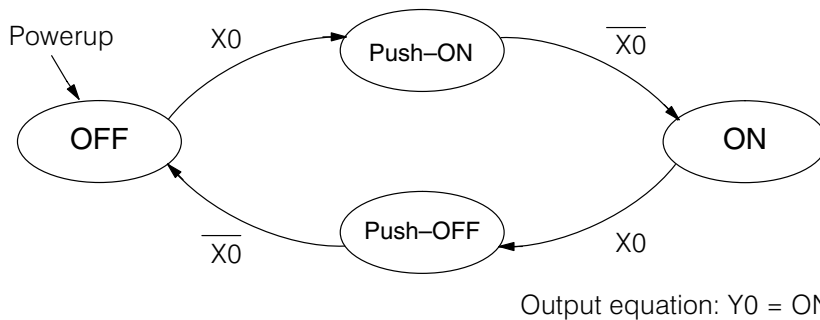
**A 4-State Process** In the process shown to the right, we use an ordinary momentary pushbutton to control a light bulb. The ladder program will latch the switch input, so that we will push and release to turn on the light, push and release again to turn it off (sometimes called toggle function). Sure, 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)!

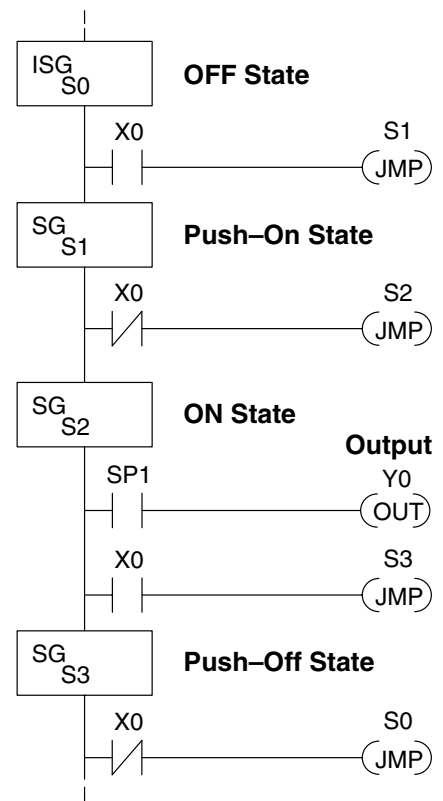
The solution is to make the the push and the release of the pushbutton separate events. Refer to the new state transition diagram below. At powerup we enter the OFF state. When switch X0 is pressed, we enter the Press-ON state. When it is released, we enter the ON state. Note that X0 with the bar above 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 256 total stages are available in the DL05, numbered 0 to 377 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!

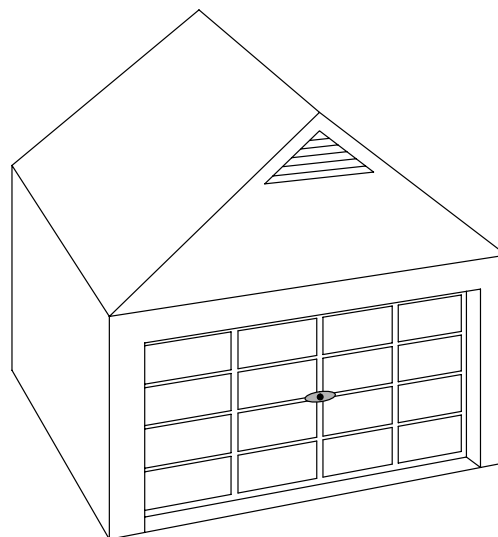
## Stage Program Example: A Garage Door Opener

### 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.

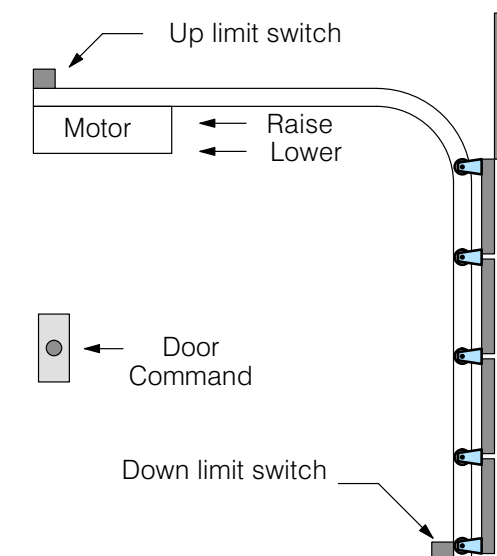
Our garage door controller has a motor which raises or lowers the door on command. The garage owner pushes and releases a momentary pushbutton once to raise the door. After the door is up, another push-release cycle will lower the door.



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

The motor has two command inputs: raise and lower. When neither input is active, the motor is stopped.

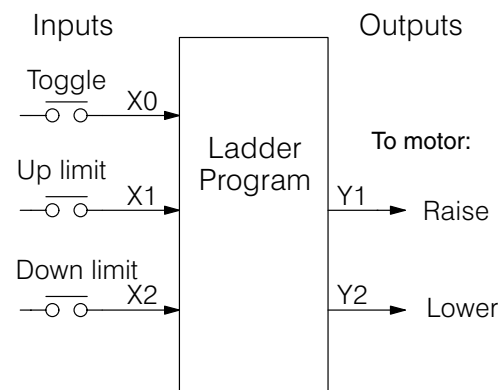
The door command is just a simple pushbutton. Whether wall-mounted as shown, or a radio-remote control, all door control commands logical OR together as one pair of switch contacts.



### 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.

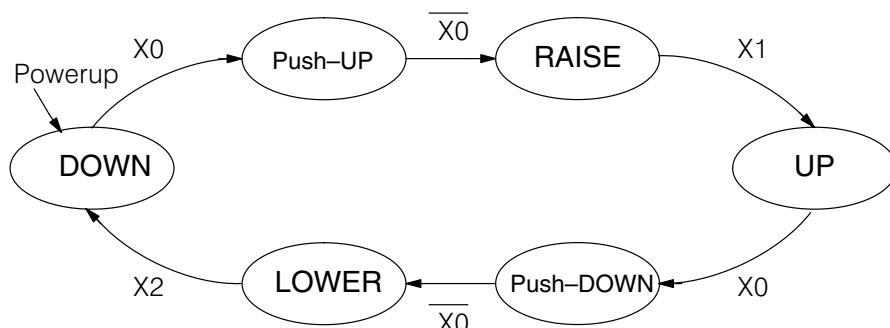
The controller has two outputs to drive the motor. Y1 is the up (raise the door) command, and Y2 is the down (lower the door) command.



## 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.



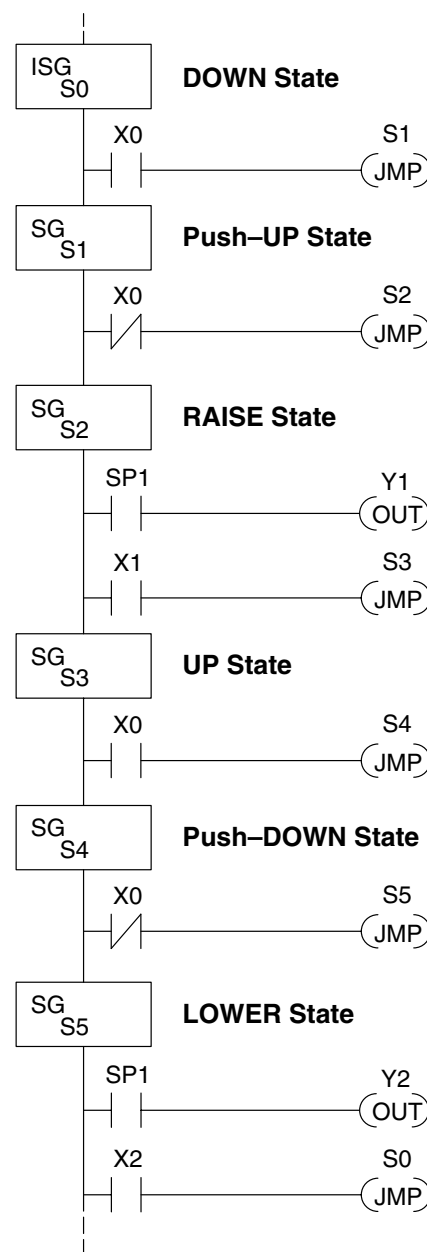
Output equations: Y1 = RAISE Y2 = LOWER

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.

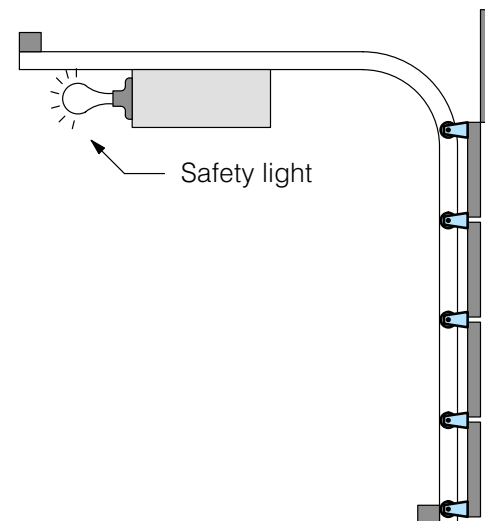


### Add Safety Light Feature

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.

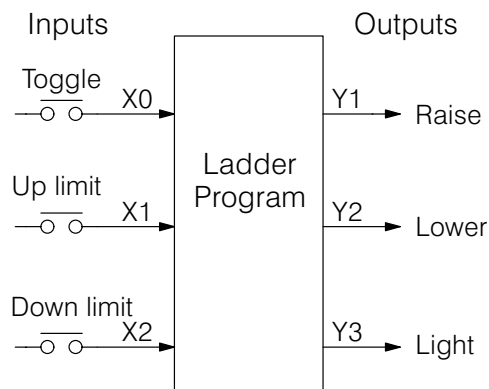
This part of the exercise will demonstrate the use of parallel states in our state diagram. Instead of using the JMP instruction, we'll use the set and reset commands.



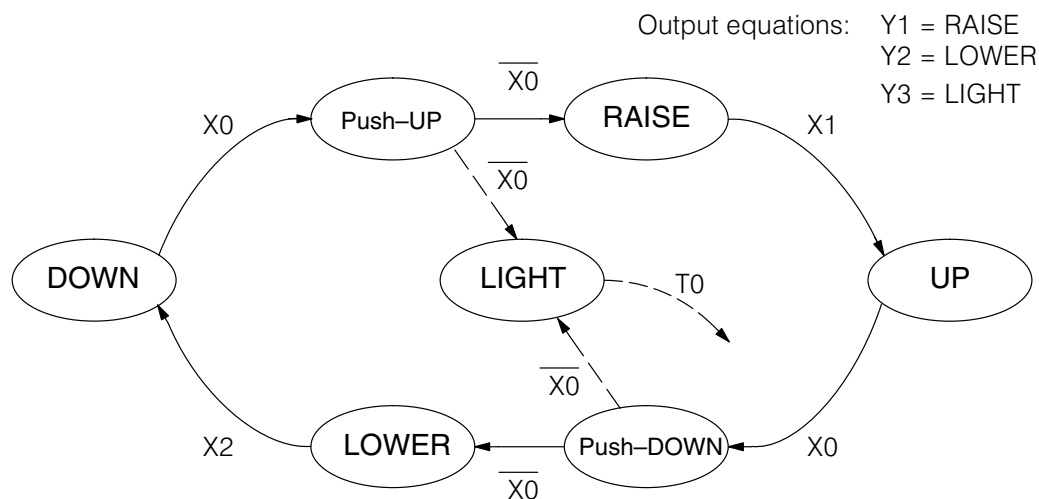
### Modify the Block Diagram and State Diagram

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 opens, we transition from S1 and go to two new active states: S2 and S6. In the Push-DOWN state S4, we make the same additions. So, any time someone presses the door control pushbutton, the light turns on.

Most new stage programmers would be concerned about where to place the Light Stage in the ladder, and how to number it. The good news is that it doesn't matter!

- 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.

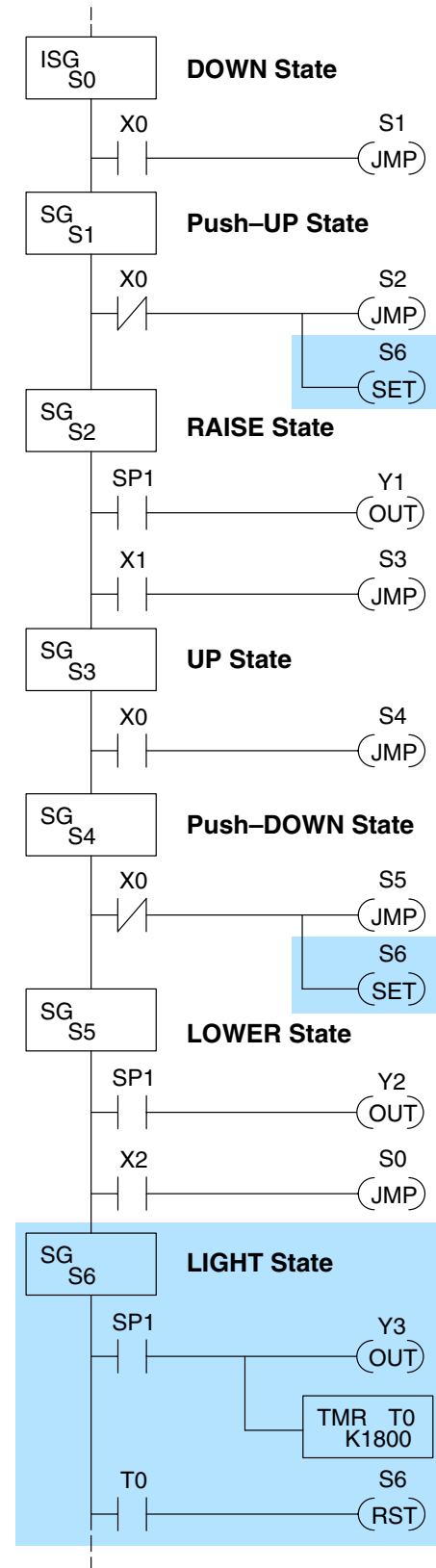
In stage S6, we turn on the safety light by energizing Y3. Special relay contact SP1 is always on. Timer T0 times at 0.1 second per count. To achieve 3 minutes time period, we calculate:

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

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

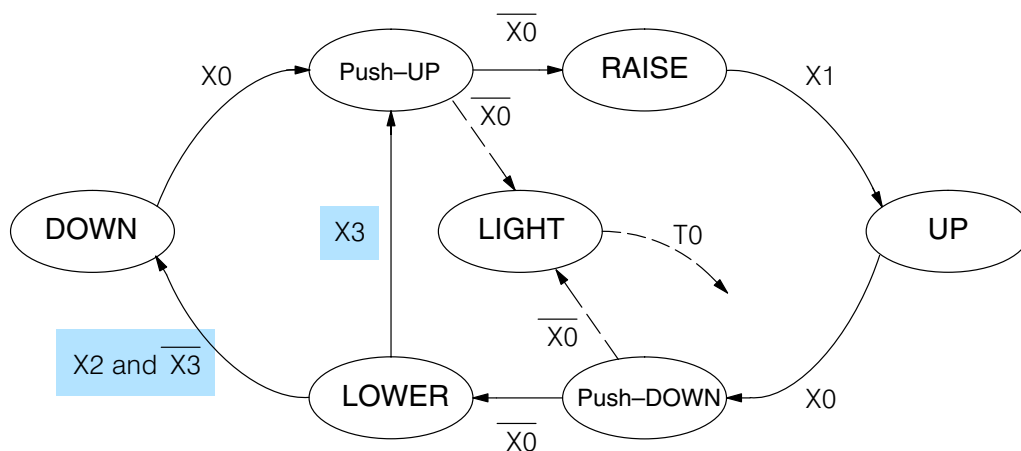
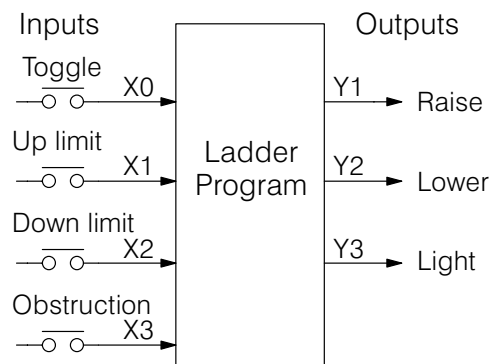
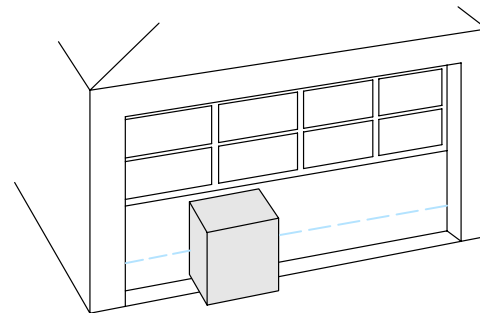
The timer has power flow whenever stage S6 is active. The corresponding timer bit T0 is set when the timer expires. So three minutes later, T0=1 and the instruction Reset S6 causes the stage to be inactive.

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

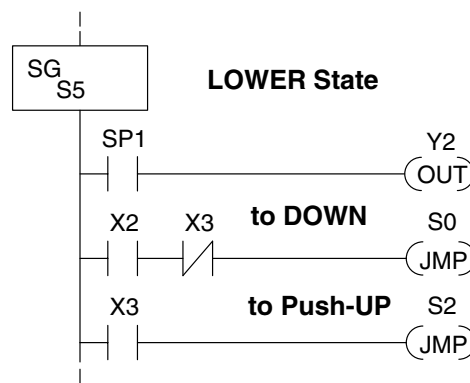
Some garage door openers today will detect an object under the door. This halts further lowering of the door. Usually implemented with a photocell (“electric-eye”), a door in the process of being lowered will halt and begin raising. We will define our safety feature to work in this way, adding the input 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.



### 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.

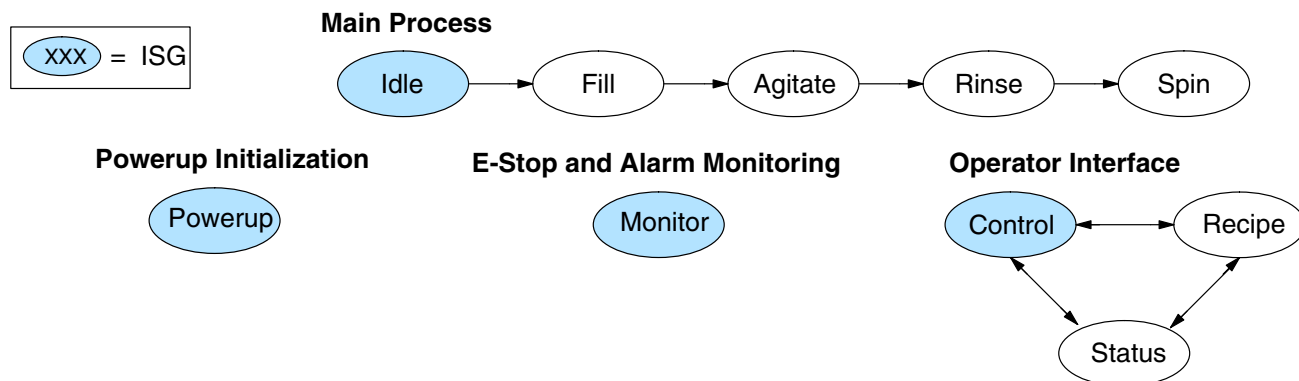


## Stage Program Design Considerations

### 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.

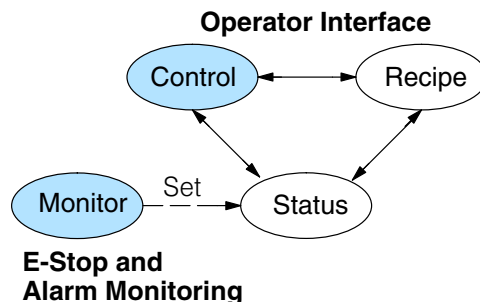
The following figure shows a typical application. During operation, the primary manufacturing activity Main Process, Powerup Initialization, E-Stop and Alarm Monitoring, and Operator Interface are all running. At powerup, three initial stages shown begin operation.



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

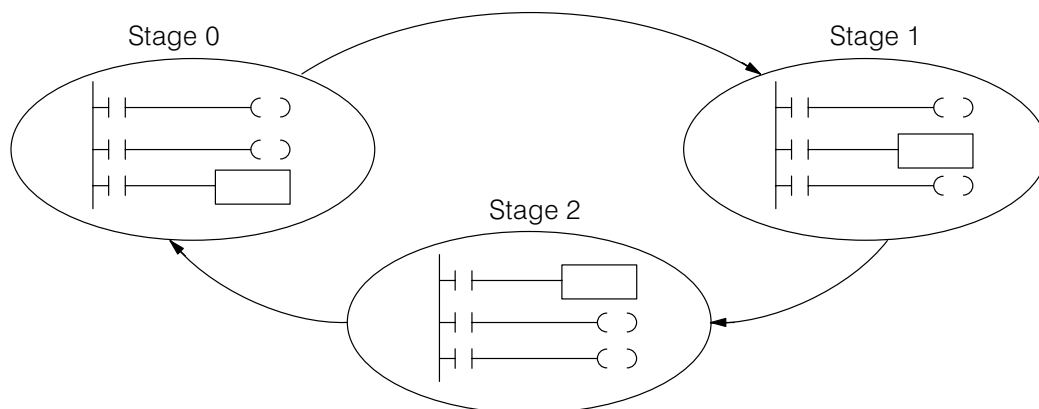
- **Powerup Initialization** – This stage contains ladder rung tasks done just once at powerup. Its last rung resets the stage, so this stage is only active for one scan (or only as many scans that are required).
- **Main Process** – this stage sequence controls the heart of the process or machine. One 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.

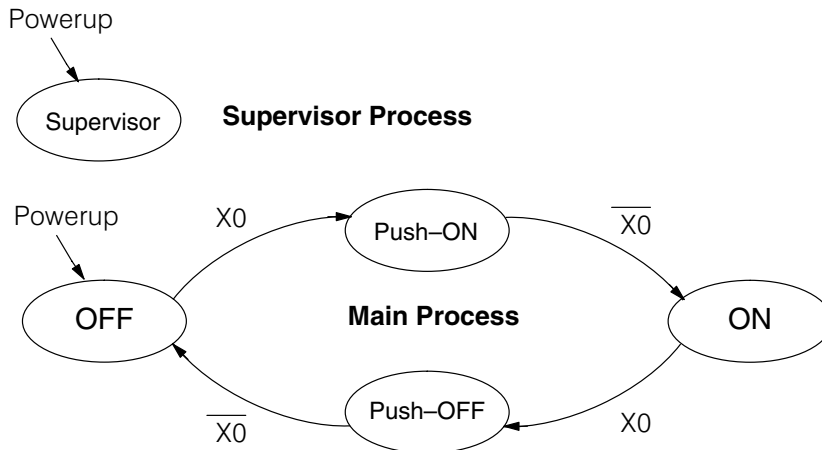
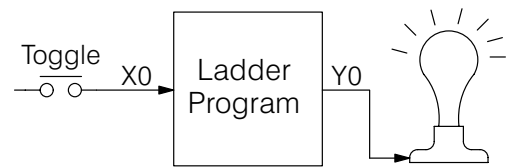
The ordinary Counter instruction does have a restriction inside stages: it may not be reset from other stages using the RST instruction for the counter bit. However, the special Stage counter provides a solution (see next paragraph).

**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

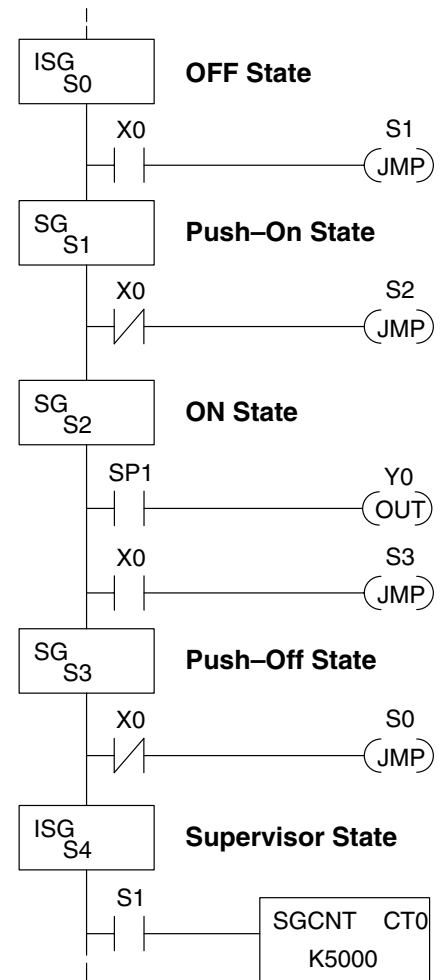
You may recall the light bulb on-off controller example from earlier in this chapter. For the purpose of illustration, suppose we want to monitor the “productivity” of the lamp process, by counting the number of on-off cycles which occurs. This application will require the addition of a simple counter, but the key decision is in where to put the counter.



New stage programming students will typically try to place the counter inside one of 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.



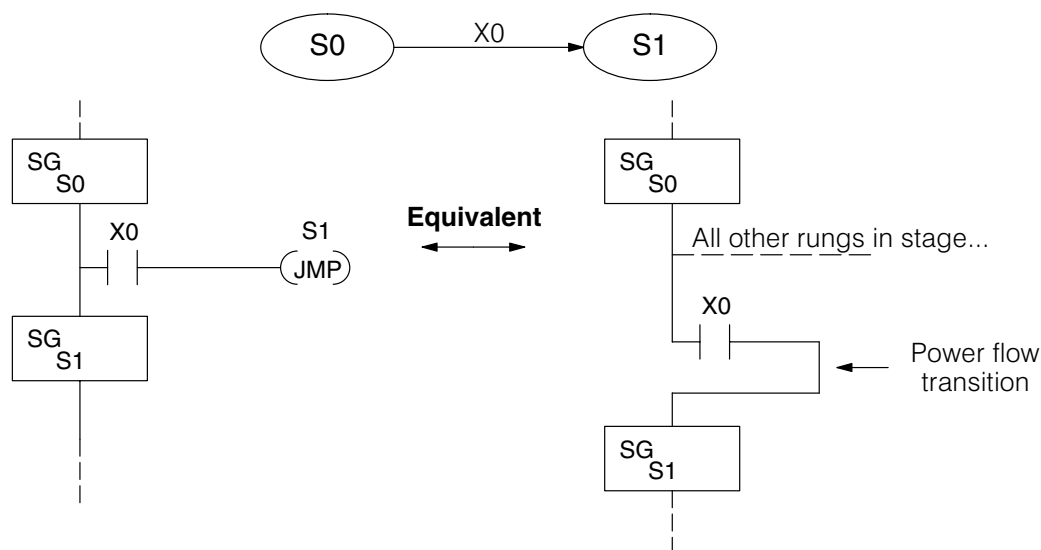
### Stage Counter

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 **DirectSOFT**, 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.

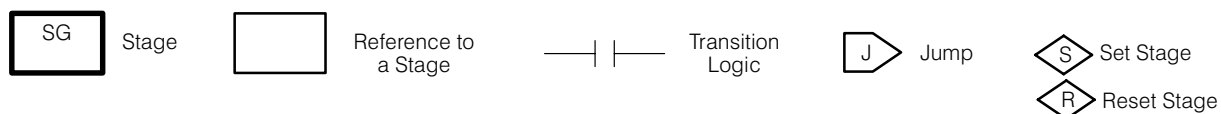


Recall that the Stage JMP instruction may occur anywhere in the current stage, and the result is the same. However, power flow transitions (shown above) must occur as the last rung in a stage. All other rungs in the stage will precede it. The power flow transition method is also achievable on the handheld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

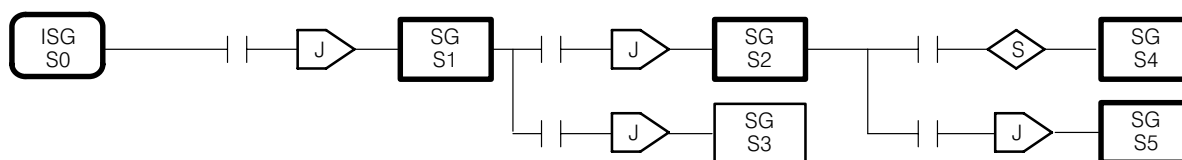
The power flow transition method does eliminate one Stage JMP instruction, its only advantage. However, it is not as easy to make program changes as using the Stage JMP. Therefore, we advise using Stage JMP transitions for most programmers.

### Stage View in DirectSOFT

The Stage View option in **DirectSOFT** 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.

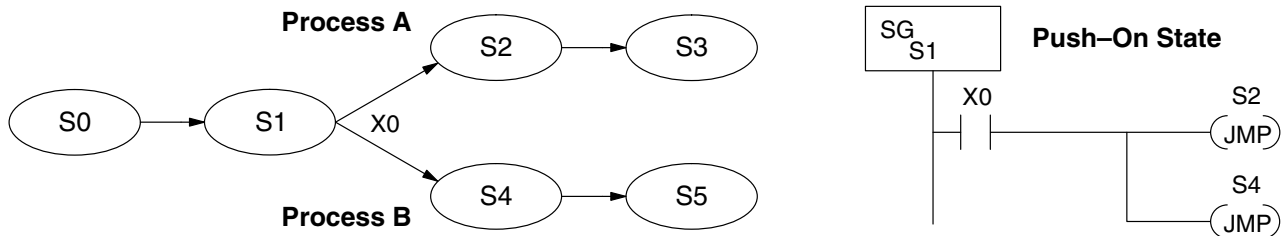


The following diagram is a typical stage view of a ladder program containing stages. Note the left-to-right direction of the flow chart.



## Parallel Processing Concepts

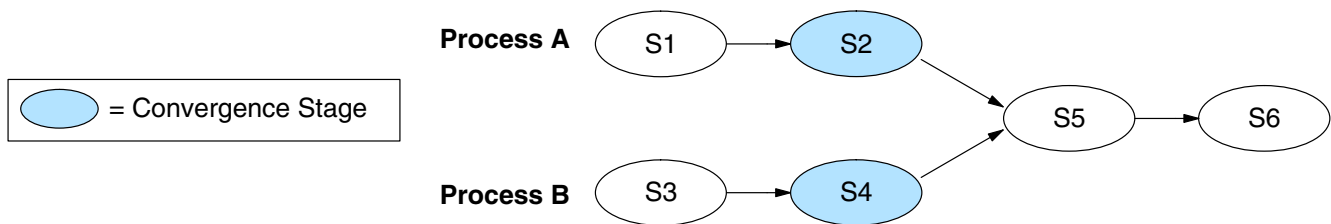
**Parallel Processes** Previously in this chapter we discussed how a state may transition to either one state or another, called an *exclusive transition*. In other cases, we may need to branch simultaneously to two or more parallel processes, as shown below. It is acceptable to use all JMP instructions 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!

### Converging Processes

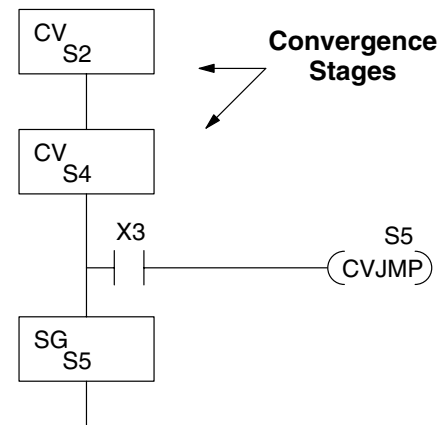
Now we consider the opposite case of parallel branching, which is *converging processes*. This simply means we stop doing multiple things and continue doing one thing at a time. In the figure below, processes A and B converge when stages S2 and S4 transition to S5 at some point in time. So, S2 and S4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



### Convergence Stages (CV)

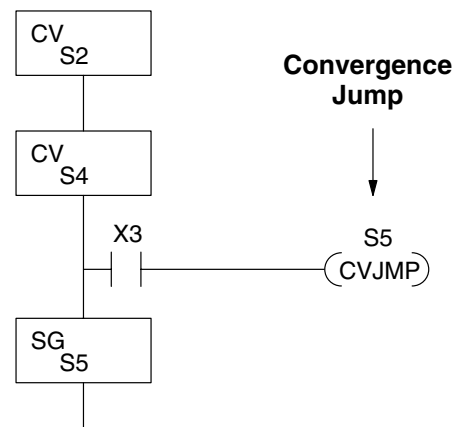
While the converging principle is simple enough, it brings a new complication. As parallel processing completes, the multiple processes almost never finish at the same time. In other words, how can we know whether Stage S2 or S4 will finish last? This is an important point, because we have to decide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

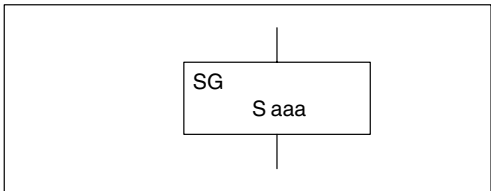
- A convergence stage is to be used as the last stage of a process which is running in parallel to another process or processes. A transition to the convergence stage means that a particular process is through, and represents a waiting point until all other parallel processes also finish.
- The maximum 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<sup>PLUS</sup>(Stage) Instructions

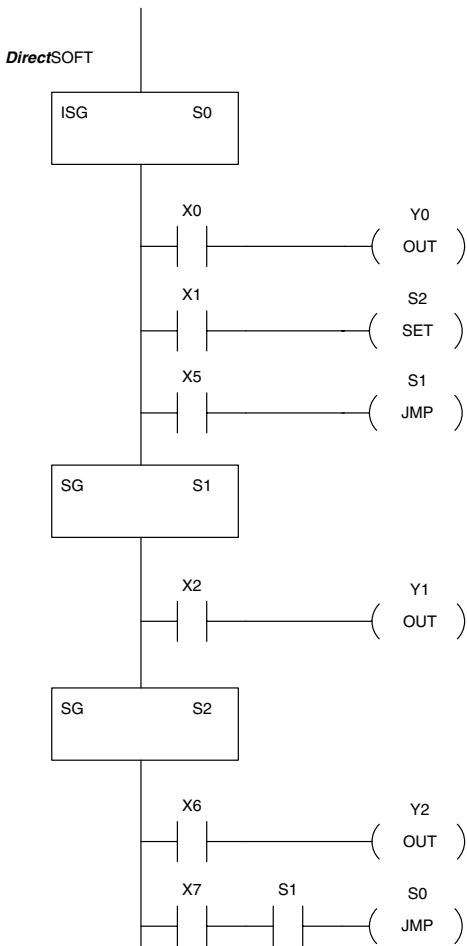
## Stage<sup>TM</sup> (SG)

The Stage<sup>TM</sup> instructions are used to create structured RLL<sup>PLUS</sup> 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		DL05 Range
		aaa
Stage	S	0-377

The following example is a simple RLL<sup>PLUS</sup> program. This program utilizes an initial stage, stage, and jump instructions to create a structured program.

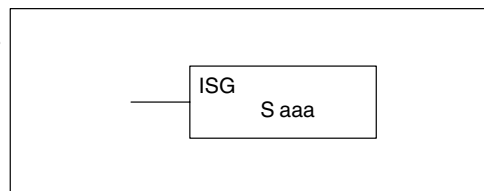


Handheld Programmer Keystrokes

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.

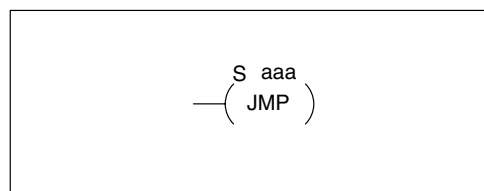


Operand Data Type	DL05 Range
	aaa
Stage S	0-377

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

**JUMP  
(JMP)**

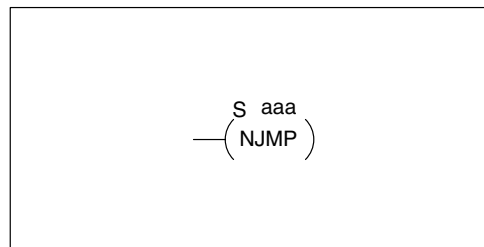
The Jump instruction allows the program to transition from an active stage containing the jump instruction to another stage (specified in the instruction). The jump occurs when the input logic is true. The active stage containing the Jump will deactivate 1 scan later.



Operand Data Type	DL05 Range
	aaa
Stage S	0-377

**Not Jump  
(NJMP)**

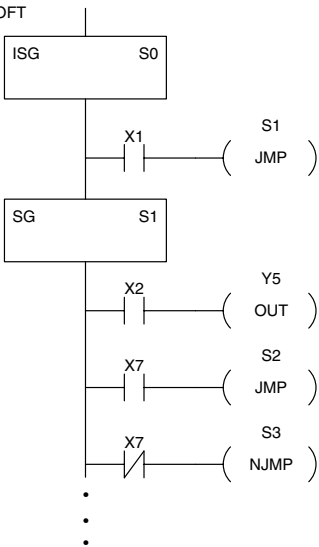
The Not Jump instruction allows the program to transition from an active stage which contains the jump instruction to another which is specified in the instruction. The jump will occur when the input logic is off. The active stage that contains the Not Jump will be deactivated 1 scan after the Not Jump instruction is executed.



Operand Data Type	DL05 Range
	aaa
Stage S	0-377

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.

DirectSOFT



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

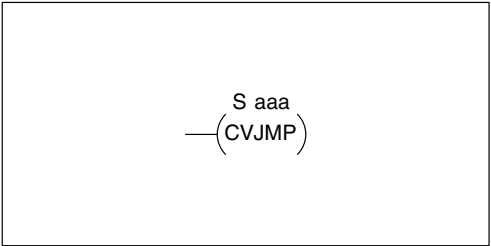
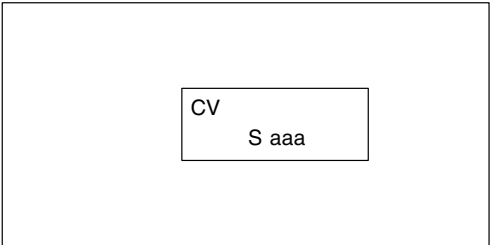
### Converge Stage (CV) and Converge Jump (CVJMP)

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

When all of the Converge Stages within a group become active, the CVJMP instruction (and any additional logic in the final CV stage) will be executed. All preceding CV stages *must* be active before the final CV stage logic can be executed. All Converge Stages are deactivated one scan after the CVJMP instruction is executed.

Additional logic instructions are only allowed following the last Converge Stage instruction and before the CVJMP instruction. Multiple CVJUMP instructions are allowed.

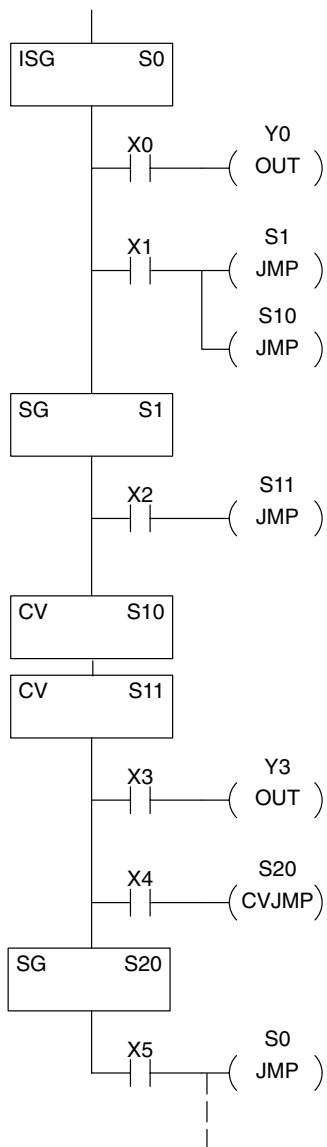
Converge Stages must be programmed in the main body of the application program. This means they cannot be programmed in Subroutines or Interrupt Routines.



Operand Data Type	DL05 Range
	aaa
Stage S	0-377

In the following example, when Converge Stages S10 and S11 are *both* active the CVJMP instruction will be executed when X4 is on. The CVJMP will deactivate S10 and S11, and activate S20. Then, if X5 is on, the program execution will jump back to the initial stage, S0.

DirectSOFT Display



Handheld Programmer Keystrokes

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					

## 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 **DirectSOFT**, 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. . . .

- DL05 PID Loop Features
  - Loop Setup Parameters
  - Loop Sample Rate and Scheduling
  - Ten Steps to Successful Process Control
  - Basic Loop Operation
  - PID Loop Data Configuration
  - PID Algorithms
  - Loop Tuning Procedure
  - PV Analog Filter
  - Feedforward Control
  - Time Proportioning Control
  - Cascade Control
  - Process Alarms
  - Ramp/Soak Generator
  - Troubleshooting Tips
  - Bibliography
  - Glossary of PID Loop Terminology
-

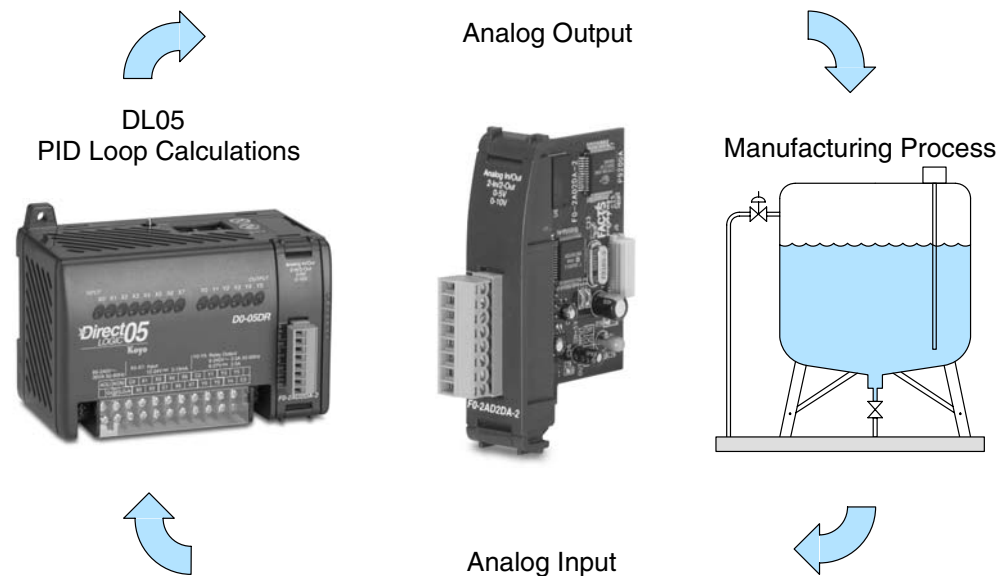
## DL05 PID Loop Features

### Main Features

The DL05 process loop control offers a sophisticated set of features to address many application needs. The main features are:

- Up to 4 loops, individual programmable sample rates
- Manual/ Automatic/Cascaded loop capability available
- Two types of bumpless transfer available
- Full-featured alarms
- Ramp/soak generator with up to 16 segments
- Auto Tuning

The DL05 CPU has process control loop capability in addition to ladder program execution. You can select and configure up to four loops. All sensor and actuator wiring connects directly to DL05 analog modules. All process variables, gain values, alarm levels, etc., associated with each loop reside in a Loop Variable Table in the CPU. The DL05 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 DL05 is the **DirectSOFT32** programming software, release 3.0c, or later. **DirectSOFT32** uses dialog boxes to help you set up the individual loops. After completing the setup, you can use **DirectSOFT32**'s PID Trend View to tune each loop. The configuration and tuning selections you make are stored in the DL05's FLASH memory, which is retentive. The loop parameters also may be saved to disk for recall later.

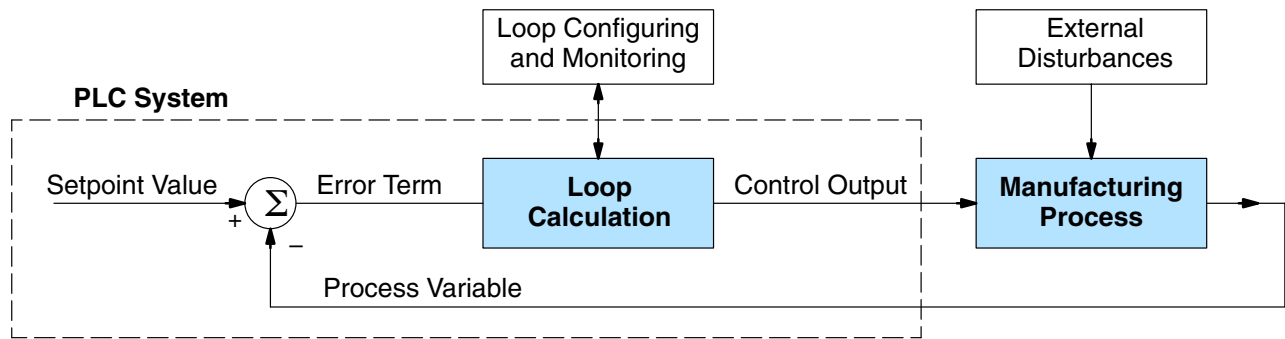


PID Loop Feature	Specifications
Number of loops	Selectable, 4 maximum
CPU V-memory needed	32 words (V locations) per loop selected, 64 words if using ramp/soak
PID algorithm	Position or Velocity form of the PID equation
Control Output polarity	Selectable direct-acting or reverse-acting
Error term curves	Selectable as linear, square root of error, and error squared
Loop update rate (time between PID calculation)	0.05 to 99.99 seconds, user programmable
Minimum loop update rate	0.05 seconds for 1 to 4 loops,
Loop modes	Automatic, Manual (operator control), or Cascade control
Ramp/Soak Generator	Up to 8 ramp/soak steps (16 segments) per loop with indication of ramp/soak step number
PV curves	Select standard linear, or square-root extract (for flow meter input)
Set Point Limits	Specify minimum and maximum setpoint values
Process Variable Limits	Specify minimum and maximum Process Variable values
Proportional Gain	Specify gains of 0.01 to 99.99
Integrator (Reset)	Specify reset time of 0.1 to 999.8 in units of seconds or minutes
Derivative (Rate)	Specify the derivative time from 0.01 to 99.99 seconds
Rate Limits	Specify derivative gain limiting from 1 to 20
Bumpless Transfer I	Automatically initialized bias and setpoint when control switches from manual to automatic
Bumpless Transfer II	Automatically set the bias equal to the control output when control switches from manual to automatic
Step Bias	Provides proportional bias adjustment for large setpoint changes
Anti-windup	For position form of PID, this inhibits integrator action when the control output reaches 0% or 100 % (speeds up loop recovery when output recovers from saturation)
Error Deadband	Specify a tolerance (plus and minus) for the error term (SP-PV), so that no change in control output value is made

Alarm Feature	Specifications
Deadband	Specify 0.1% to 5% alarm deadband on all alarms
PV Alarm Points	Select PV alarm settings for Low-low, Low, High, and High-high conditions
PV Deviation	Specify alarms for two ranges of PV deviation from the setpoint value
Rate of Change	Detect when PV exceeds a rate of change limit you specify

## 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 DL05 uses the Proportional-Integral-Derivative (PID) algorithm (more on this later).

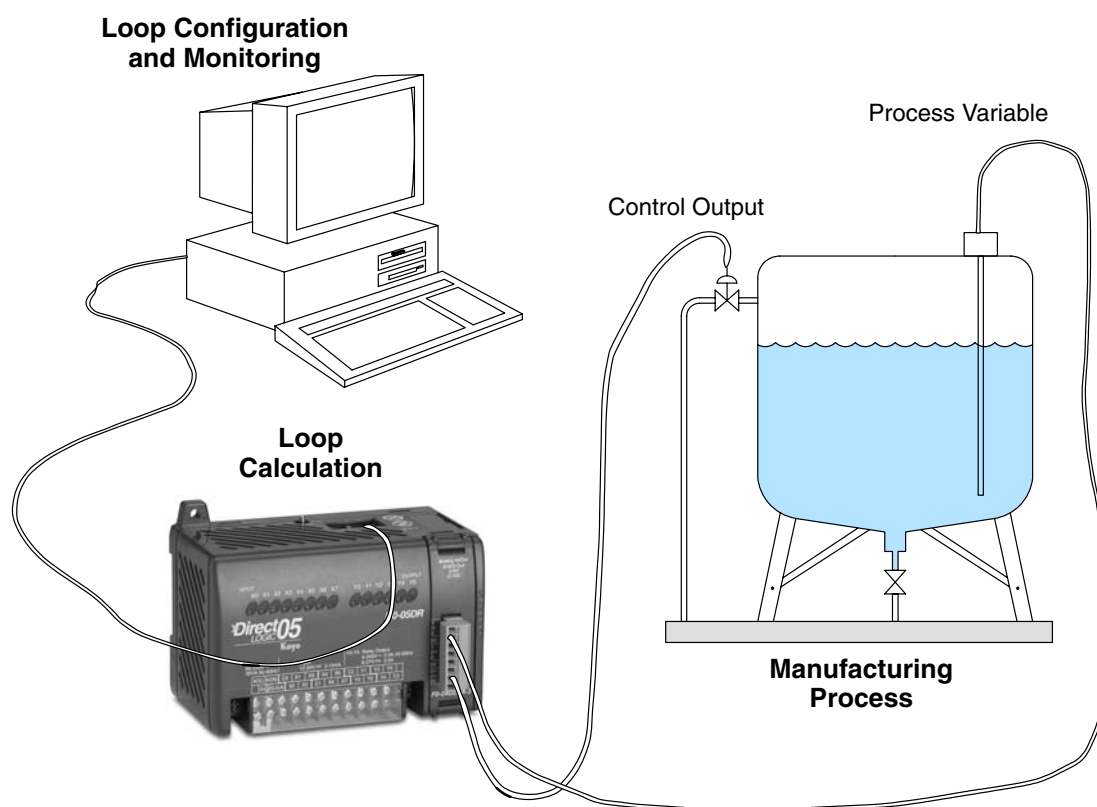
**Control Output** – the result of the loop calculation, which becomes a command for the process (such as the heater level in an oven).

**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.

Using an analog I/O combination 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 **DirectSOFT32**, the PLC programming software for **DirectLOGIC** programmable controllers. **DirectSOFT32**, release 3.0c or later, can program the DL05 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 **DirectSOFT32 Manual**.

## Loop Setup Parameters

### Loop Table and Number of Loops

The DL05 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 – V7377	write
V7641	Number of Loops	BCD	0 – 4	write
V7642	Loop Error Flags	Binary	0 or 1	read

If the number of loops is “0”, the loop controller task is turned off during the ladder program scan. The loop controller will allow use of loops in ascending order, beginning with 1. For example, you cannot use loop 1 and 4 while skipping 2 and 3. The loop controller attempts to control the full number of loops specified in V7641.

### 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.



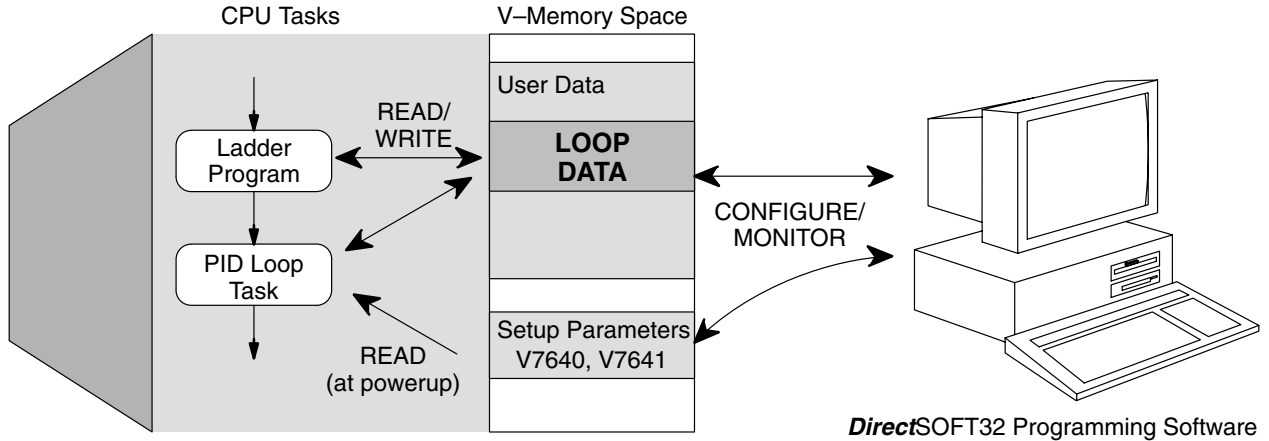
If you use the **Direct**SOFT32 loop setup dialog box, its automatic range checking prohibits possible setup errors. However, the setup parameters may be written using other methods such as RLL, so the error flag register may be helpful in those cases. The following table lists the errors reported in 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 4.
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.

V-Memory		User Data
▲	V2000	<b>LOOP #1</b> 32 words
▼	V2037	
▲	V2040	<b>LOOP #2</b> 32 words
▼	V2077	
•		
•		
•		
		<b>LOOP #3</b> 32 words
		<b>LOOP #4</b> 32 words



**NOTE:** The DL05 CPU's PID algorithm requires **DirectSOFT32** Version 3.0c (or later) and firmware version 2.1 (or later).

### 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
1	Addr + 0	PID Loop Mode Setting 1	bits
2	Addr + 1	PID Loop Mode Setting 2	bits
3	Addr + 2	Setpoint Value (SP)	word/binary
4	Addr + 3	Process Variable (PV)	word/binary
5	Addr + 4	Bias (Integrator) Value	word/binary
6	Addr + 5	Control Output Value	word/binary
7	Addr + 6	Loop Mode and Alarm Status	bits
8	Addr + 7	Sample Rate Setting	word/BCD
9	Addr + 10	Gain (Proportional) Setting	word/BCD
10	Addr + 11	Reset (Integral) Time Setting	word/BCD
11	Addr + 12	Rate (Derivative) Time Setting	word/BCD
12	Addr + 13	PV Value, Low-low Alarm	word/binary
13	Addr + 14	PV Value, Low Alarm	word/binary
14	Addr + 15	PV Value, High Alarm	word/binary
15	Addr + 16	PV Value, High-high Alarm	word/binary
16	Addr + 17	PV Value, deviation alarm (YELLOW)	word/binary
17	Addr + 20	PV Value, deviation alarm (RED)	word/binary
18	Addr + 21	PV Value, rate-of-change alarm	word/binary
19	Addr + 22	PV Value, alarm hysteresis setting	word/binary
20	Addr + 23	PV Value, error deadband setting	word/binary
21	Addr + 24	PV low-pass filter constant	word/BCD
22	Addr + 25	Loop derivative gain limiting factor setting	word/BCD
23	Addr + 26	SP value lower limit setting	word/binary
24	Addr + 27	SP value upper limit setting	word/binary
25	Addr + 30	Control output value lower limit setting	word/binary
26	Addr + 31	Control output value upper limit setting	word/binary
27	Addr + 32	Remote SP Value V-Memory Address Pointer	word/hex
28	Addr + 33	Ramp/Soak Setting Flag	bit
29	Addr + 34	Ramp/Soak Programming Table Starting Address	word/hex
30	Addr + 35	Ramp/Soak Programming Table Error Flags	bits
31	Addr + 36	PV direct access, channel number	word/hex
32	Addr + 37	Control output direct access, channel number	word/hex

**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	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	select 16 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	–	0→1 Start
1	Hold Ramp / Soak Profile	write	–	0→1 Hold
2	Resume Ramp / soak Profile	write	–	0→1 Resume
3	Jog Ramp / Soak Profile	write	–	0→1 Jog
4	Ramp / Soak Profile Complete	read	–	Complete
5	PV Input Ramp / Soak Deviation	read	Off	On
6	Ramp / Soak Profile in Hold	read	Off	On
7	Reserved	read	–	–
8–15	Current Step in R/S Profile	read	decode as byte (hex)	

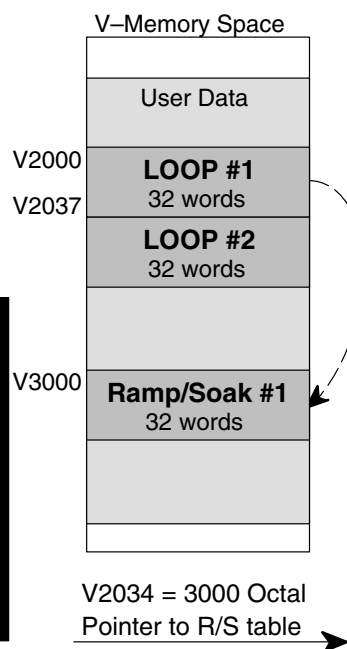
Bits 8–15 must be read as a byte to indicate the current segment number of the Ramp/Soak generator in the profile. This byte will have the values 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and 10, which represent segments 1 to 16 respectively. If the byte=0, then the Ramp/Soak table is not active.

### Ramp/Soak Table Location (Addr + 34)

Each loop that you configure has the option of using a built-in Ramp/Soak generator dedicated to that loop. This feature generates SP values that follow a profile. To use the Ramp Soak feature, you must program a separate table of 32 words with appropriate values. A **DirectSOFT32** 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

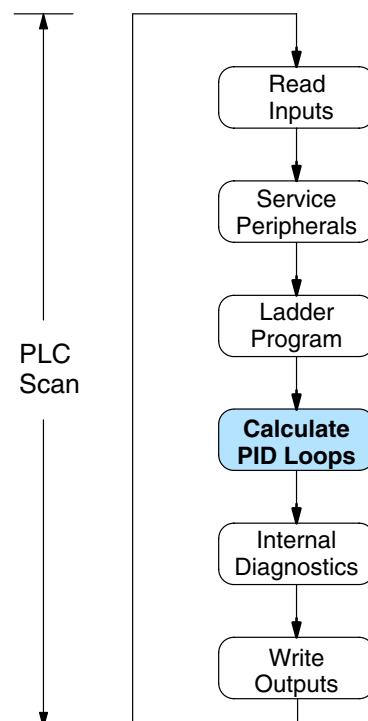
### Loop Sample Rates Addr + 07

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 DL05 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

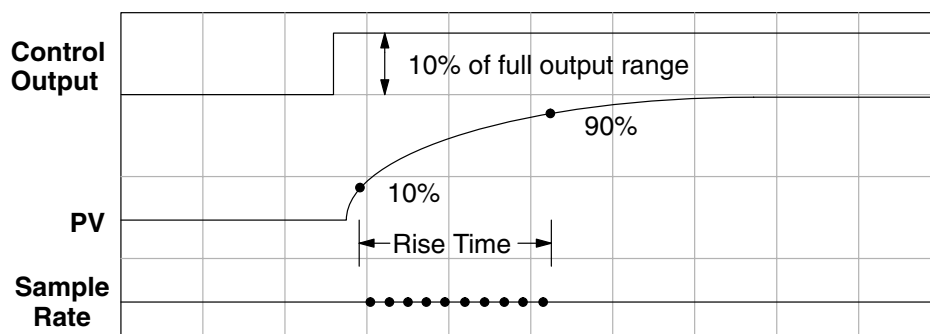
For any particular control loop, there is no single perfect sample rate to use. A good sample rate is a compromise that simultaneously satisfies various guidelines:

- 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.

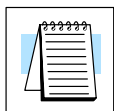
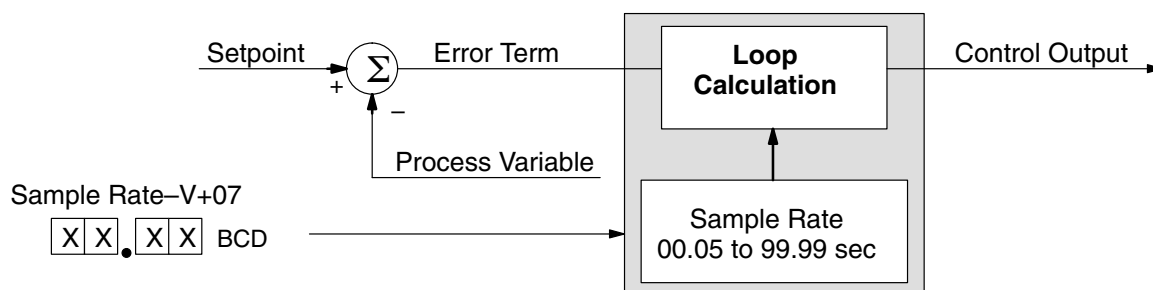


In the figure above, suppose the measured rise time response of the PV was 25 seconds. The suggested sample rate from this measurement will be 2.5 seconds. For illustration, the sample rate time line shows ten samples within the rise time period. These show the frequency of PID calculations as the PV 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 V+07 contains a BCD number from 00.05 to 99.99 (with an implied decimal point). This represents 50 mS to 99.99 seconds. This number may be programmed using **DirectSOFT32'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 $\mu$ S
Typical	250 $\mu$ S
Maximum	350 $\mu$ S

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:

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

For example, suppose the estimated scan time without loop calculations is 50 mS, and the loop sample time is 3 seconds. Now, calculate the new scan time:

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

As the calculation shows, the addition of only one loop with a slow sample rate has a very small effect on scan time. Next, 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 DL05 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 $\mu$ S
2	Water bath temperature	30 sec	0.42 $\mu$ S
3	Dye level, main tank	10 sec	1.25 $\mu$ S
4	Steam Pressure, Autoclave	1.5 sec	8.3 $\mu$ S

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

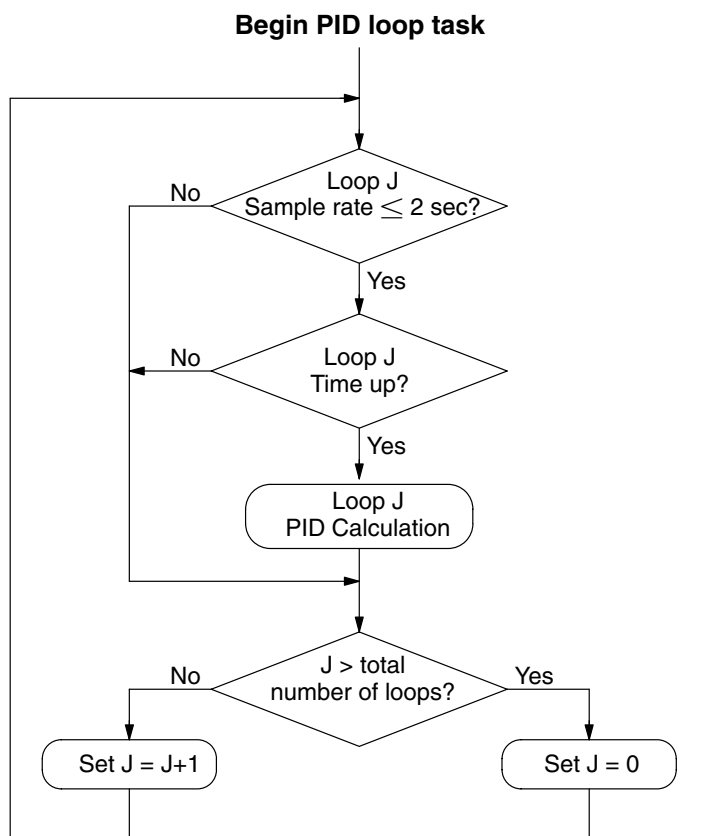
$$\text{Avg. Scan Time with PID loops} = \left[ 50 \mu\text{S} + 0.42 \mu\text{S} + 1.25 \mu\text{S} + 8.3 \mu\text{S} \right] + 50 \text{ mS} = 50.06 \text{ mS}$$

The DL05 CPU only does PID calculation on a particular scan for the loop(s) which have sample time periods that are due for an update (calculation). The built-in loop scheduler applies the following rules:

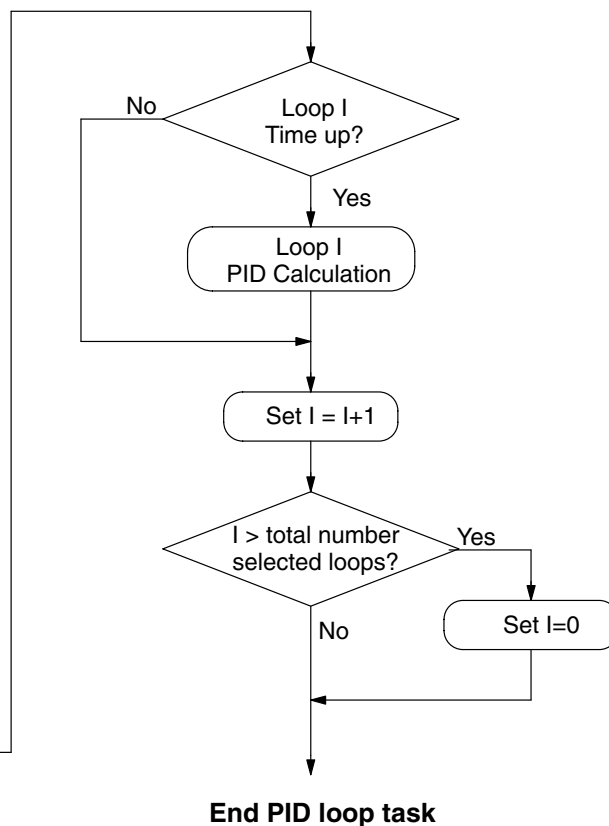
- Loops with sample rates  $\leq 2$  seconds are processed at the rate of as many loops per scan as is required to maintain each loop's sample rate. Specifying loops with fast sample rates will increase the PLC scan time. *So, use this capability only if you need it!*
- Loops with sample rates  $> 2$  seconds are processed at the rate of one or 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 ( $> 2$  sec) and fast ( $\leq 2$  sec) loops, respectively. The flow chart allows the J pointer to increment from loop 1 to the last loop, if there are any fast loops specified. The I pointer increments only once per scan, and then only when the next slow loop is due for an update. In this way, both I and J pointers cycle from 1 to the highest loop number used, except at different rates. Their combined activity keeps all loops properly updated.

Loop Sample Times  $\leq 2$  seconds:



Loop Sample Times  $> 2$  seconds:



## Ten Steps to Successful Process Control

Modern electronic controllers such as the DL05 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

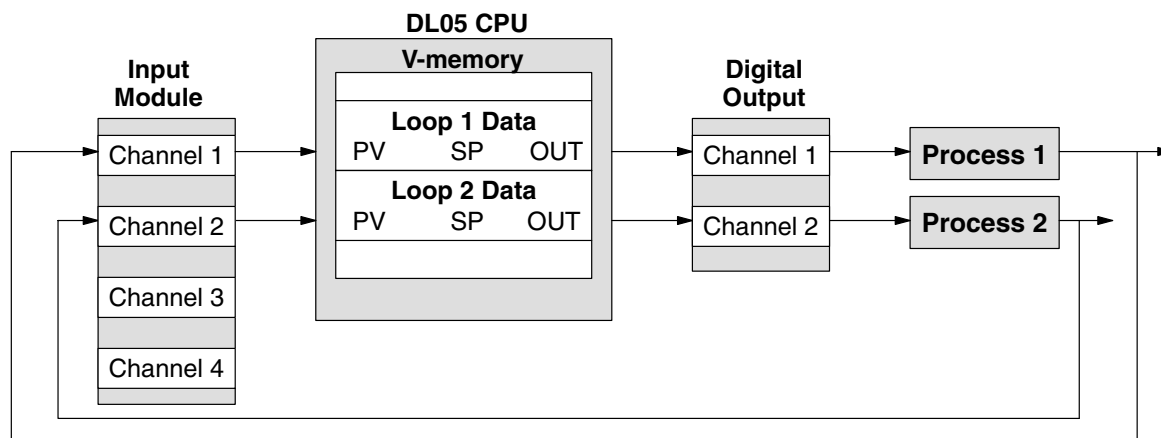
Assuming the control strategy is sound, it is still crucial to *properly size the actuators and properly scale the sensors*.

- 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 DL05 provides 12-bit and 15-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.

We offer DL05 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.



### 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 **DO-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 **DirectSOFT32** (3.0c 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.

### 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).

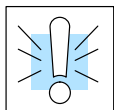
- Verify that the PV value from the sensor is correct.
- If it is safe to do so, gradually increase the control output up above 0%, and see if the PV responds (*and moves in the correct direction!*).

### 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.



**WARNING:** Be sure the Emergency Stop and power-down provision is readily accessible, in case the process goes out of control. Damage to equipment and/or serious injury to personnel can result from loss of control of some processes.

### Step 10: Save Parameters

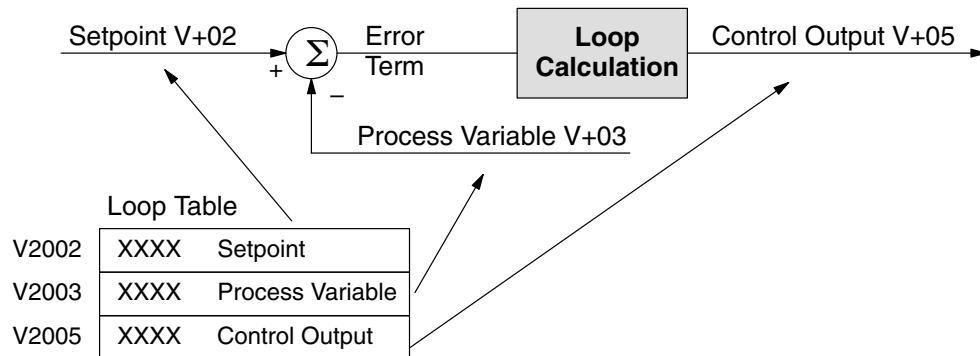
When the loop tests and tuning sessions are complete, be sure to save all loop setup parameters to disk.



## Basic Loop Operation

### Data Locations

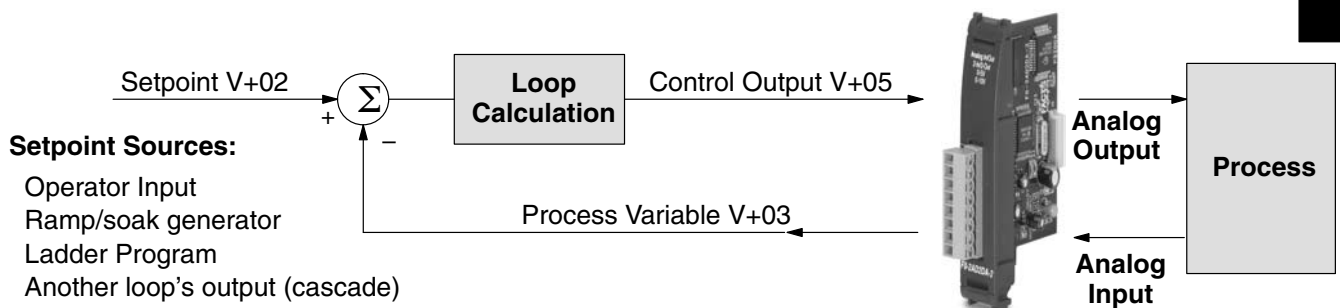
Each PID loop is dependent on the instructions and data values in its respective loop table. The following diagram shows an example of the loop table locations corresponding to the main three loop variables: SP, PV, and Control Output. The example below begins at V2000 (you can use any memory location 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 V+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 V+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.

Direct Access  
to Analog I/O

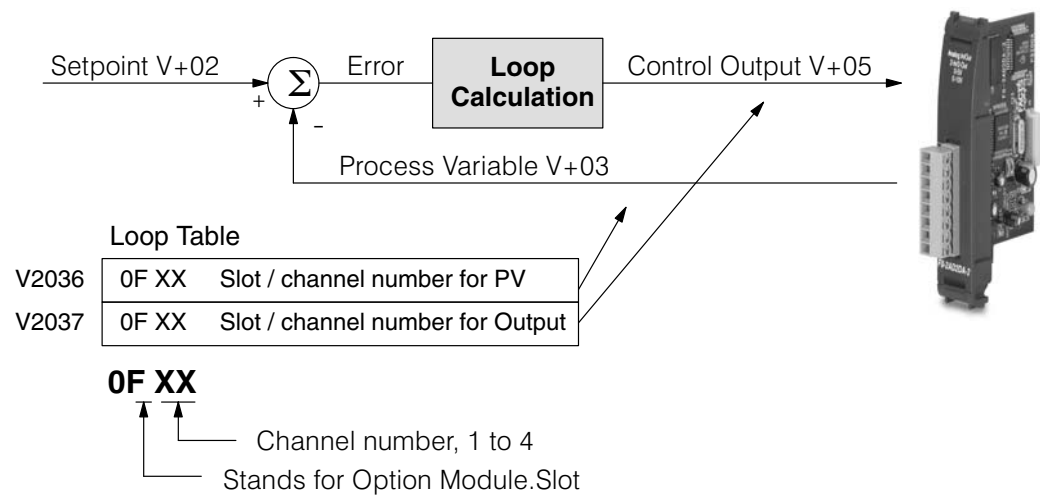


The loop controller in the DL05 PLC has the ability to directly access analog input and output values independent of the ladder logic scan. These values represent the process variable (PV) and the control output. The Direct Access feature makes it possible for the loop controller to perform closed-loop control while the CPU is in Program Mode.

The loop controller can read the analog PV value in the selected data format directly from the desired analog input module and write the control output value in the same or a different data format to the desired analog output. The Direct Access feature, when enabled, accesses the analog values only once per PID calculation for each respective loop. The ladder logic, however, may simultaneously access the same analog input data by the standard method (LD instruction) or by the pointer method whenever the CPU is in Run Mode.

**NOTE:** If PID direct access is used, do not use the analog output pointer logic to send data to the outputs.

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 figure below shows the loop table parameters at V+36 and V+37 and their role in direct access to the analog values.



You may program these loop table parameters directly, or use the appropriate **DirectSOFT32** dialog box for easy configuring. For example, a value of “0F02” in register V2036 directs the loop controller to read the PV data from channel 1 of the analog input module. 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 analog input module.

If the PV or control output values require some math manipulation by ladder logic, then it will not be possible to use the direct access function of the loop controller. In this case, ladder logic will have to perform the math and transfer the data from the analog module as required.

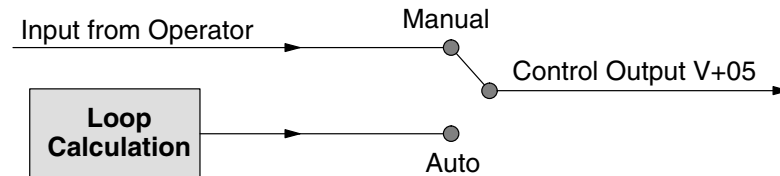


**NOTE:** The loop controller restricts the transfer of analog data to or from the module to one method. In other words, you must designate the analog module for direct access or ladder logic access.

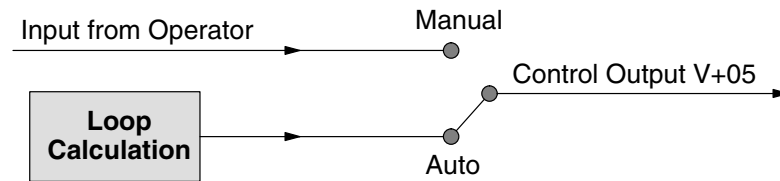
## Loop Modes

The DL05 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.

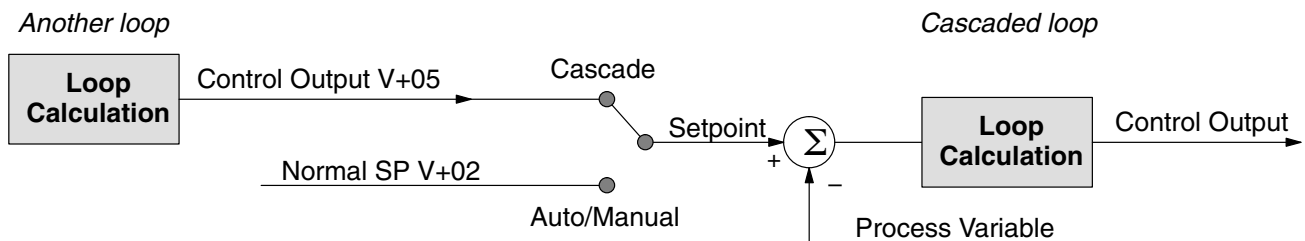
In **Manual Mode**, the loop is not executing PID calculations (however, loop alarms are still active). With regard to the loop table, the CPU stops writing values to location V+05 for that loop. It is expected that an operator or other intelligent source is manually controlling the output, by observing the PV and writing data to V+05 as necessary to keep the process under control. The drawing below shows the equivalent schematic diagram of manual mode operation.



In **Automatic Mode**, the loop operates normally and generates new control output values. It calculates the PID equation and writes the result in location V+05 every sample period of that loop. The equivalent schematic diagram is shown below.



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



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.



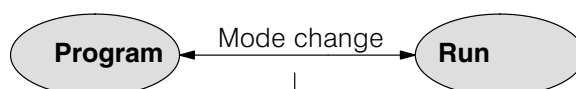
## CPU Modes and Loop Modes

The DL05 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 DL05 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).

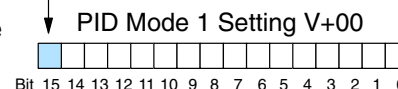
The relationship between CPU modes and loop modes is depicted in the figure below. The vertical dashed line shows the optional relationship between the mode changes. Bit 15 of PID Mode 1 setting word (V+00) determines the selection. If set to zero so the loop follows the CPU mode, then placing the CPU in Program Mode will force all loops into Manual Mode. Similarly, placing the CPU in Run mode will allow each loop to return to the mode it was in previously (which includes Manual, Automatic, and Cascade). With this selection you automatically affect the modes of the loops by changing the CPU mode.

### CPU Modes:

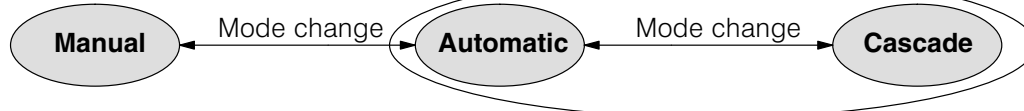


### Loop Mode Linking

0 = loop follows PLC mode  
1 = loop is independent of PLC mode

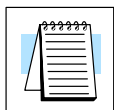


### Loop Modes:



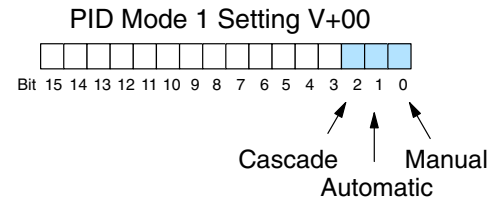
If Bit 15 is set to one, then the loops will run independent of the CPU mode. It is like having two independent processors in the CPU... one is running ladders and the other is running the process loops.

**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 **DirectSOFT32**) 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

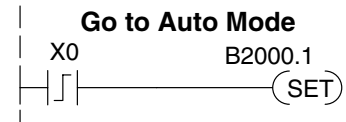
The first three bits of the PID Mode 1 word (V+00) request the operating mode of the corresponding loop. Note: these bits are mode change *requests*, not commands (certain conditions can prohibit a particular mode change – see next page).



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:

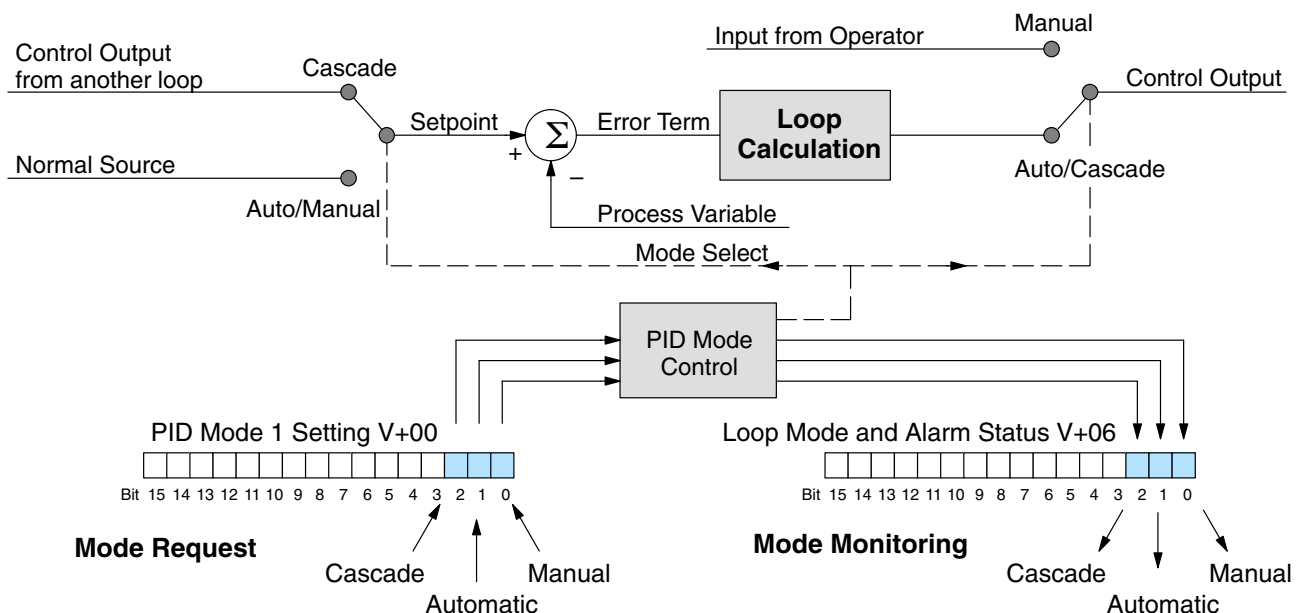
- **DirectSOFT32’s PID View** – this is the easiest method. Click on one of the radio buttons, and **DirectSOFT32** sets the appropriate bit.
- **HPP** – Use Word Status (WD ST) to monitor the contents of V+00, which will be a 4-digit BCD/hex value. You must calculate and enter a new value for V+00 that ORs the correct mode bit with its current value.
- **Ladder program**– ladder logic can request any loop mode when the PLC is in Run Mode. This will be necessary after application startup.

Use the program shown to the right to SET the mode bit on (do not use an out coil). On a 0–1 transition of X0, the rung sets the Auto bit = 1. The loop controller resets it.



- **Operator panel** – interface the operator’s panel to ladder logic using standard methods, then use the technique above to set the mode bit.

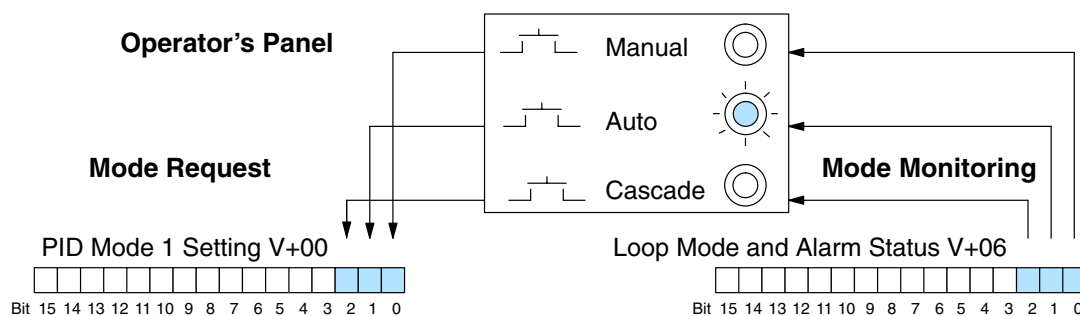
Since we can only *request* mode changes, the PID loop controller decides when to permit mode changes and provides the loop mode status. It reports the current mode on bits 0, 1, and 2 of the Loop Mode and Alarm Status word, location V+06 in the loop table. The parallel request / monitoring functions are shown in the figure below. The figure also shows the 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 16 loops as the desired mode of operation. If power failure and restoration occurs during PLC Run Mode, the CPU returns all loops to their prior mode (which could be Manual, Auto, or Cascade).
- On a Program-to-Run mode transition, the CPU forces each loop to return to its prior mode recorded during the last PLC Run Mode.
- You can add and configure new loops only when the PLC is in Program Mode. New loops automatically begin in Manual Mode.

### Loop Mode Override

In normal conditions the mode of a loop is determined by the request to V+00, bits 0, 1, and 2. However, some conditions exist which will prevent a requested mode change from occurring:

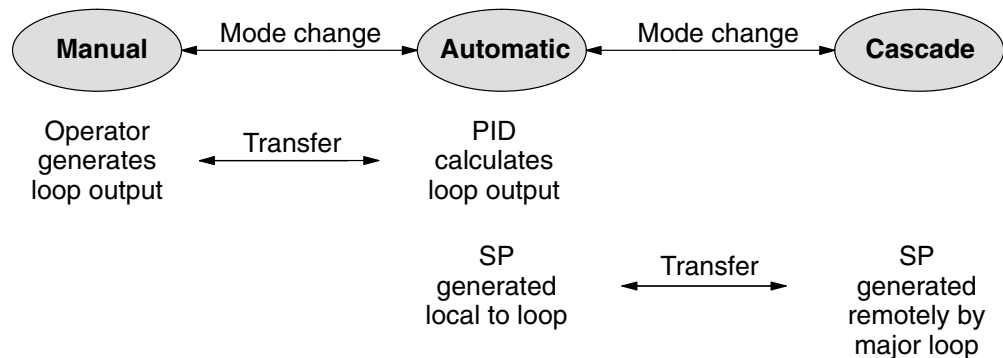
- A loop that is not set independent of PLC mode cannot change modes when the PLC is in Program mode.
- A major loop of a cascaded pair of loops cannot go from Manual to Auto until its minor loop is in Cascade mode.

In other situations, the PID loop controller will automatically change the mode of the loop to ensure safe operation:

- A loop which develops an error condition automatically goes to Manual.
- If the minor loop of a cascaded pair of loops leaves Cascade Mode for any reason, its major loop automatically goes to Manual Mode.

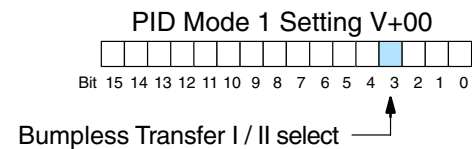
## Bumpless Transfers

In process control, the word “transfer” has a particular meaning. A loop transfer occurs when we change its mode of operation, as shown below. When we change loop modes, what we are really doing is causing a transfer of control of some loop parameter from one source to another. For example, when a loop changes from Manual Mode to Automatic Mode, control of the output changes from the operator to the loop controller. When a loop changes from Automatic Mode to Cascade Mode, control of the SP changes from its original source in Auto Mode to the output of another loop (the major loop).



The basic problem of loop transfers is the two different sources of the loop parameter being transferred will have different numerical values. This causes the PID calculation to generate an undesirable step change, or “bump” on the control output, thereby upsetting the loop to some degree. The “bumpless transfer” feature arbitrarily forces one parameter equal to another at the moment of loop mode change, so the transfer is smooth (no bump on the control output).

The bumpless transfer feature of the DL05 loop controller is available in two types: Bumpless I, and Bumpless II. Use **DirectSOFT32**’s PID Setup dialog box to select transfer type. Or, you can use bit 3 of PID Mode 1 V+00 setting as shown.



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

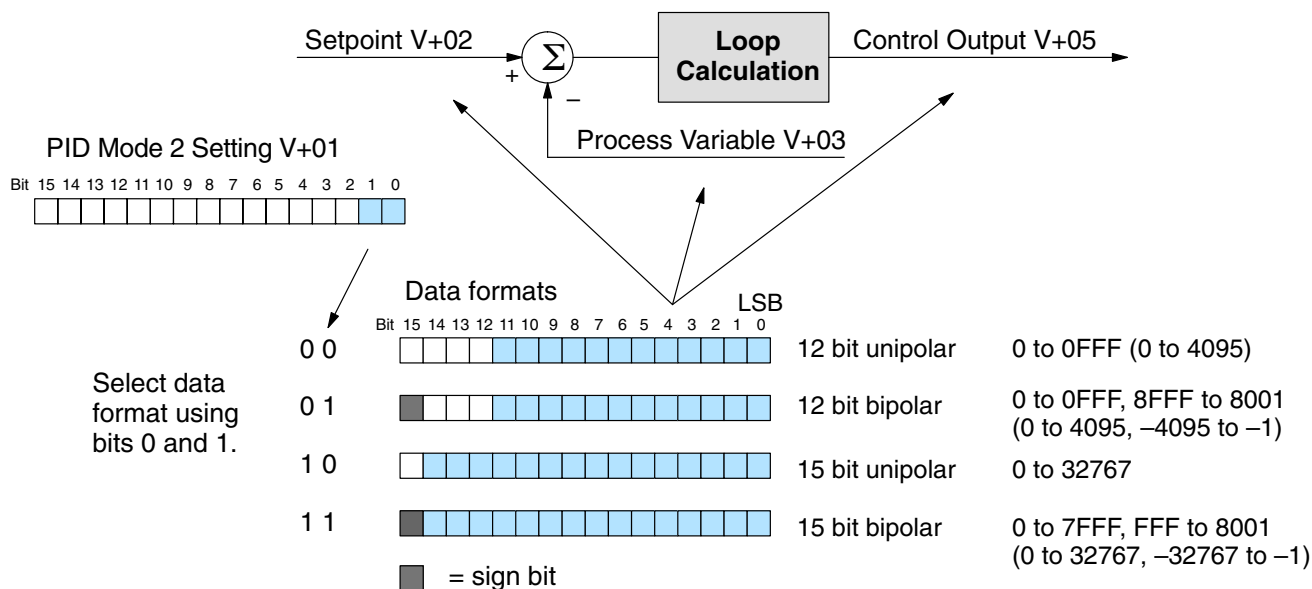
Transfer Type	Transfer Select Bit	PID Algorithm	Manual-to-Auto Transfer Action	Auto-to-Cascade Transfer Action
Bumpless Transfer I	0	Position	Forces Bias = Control Output Forces SP = PV	Forces Major Loop Output = Minor Loop PV
		Velocity	Forces SP = PV	Forces Major Loop Output = Minor Loop PV
Bumpless Transfer II	1	Position	Forces Bias = Control Output	none
		Velocity	none	none



## PID Loop Data Configuration

### Loop Parameter Data Formats

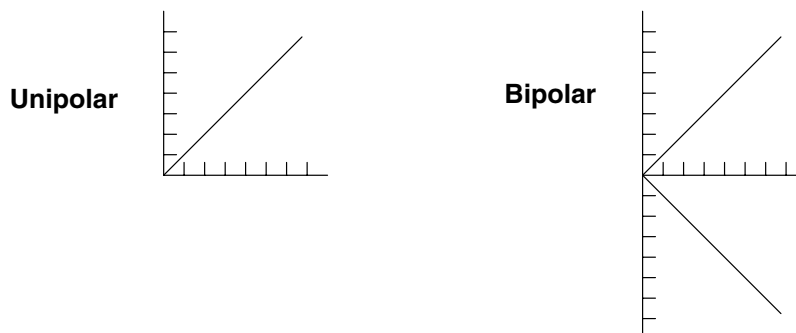
In choosing the Process Variable range and resolution, a related choice to make is the data format of the three main loop variables: SP, PV, and Control Output (the Integrator sum in V+04 also uses this data format). The four data formats available are 12 or 15 bit (right justified), signed or unsigned (MSB is sign bit in bipolar formats). The four binary combinations of bits 0 and 1 of PID Mode 2 word V+01 choose the format. The **DirectSOFT32** 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

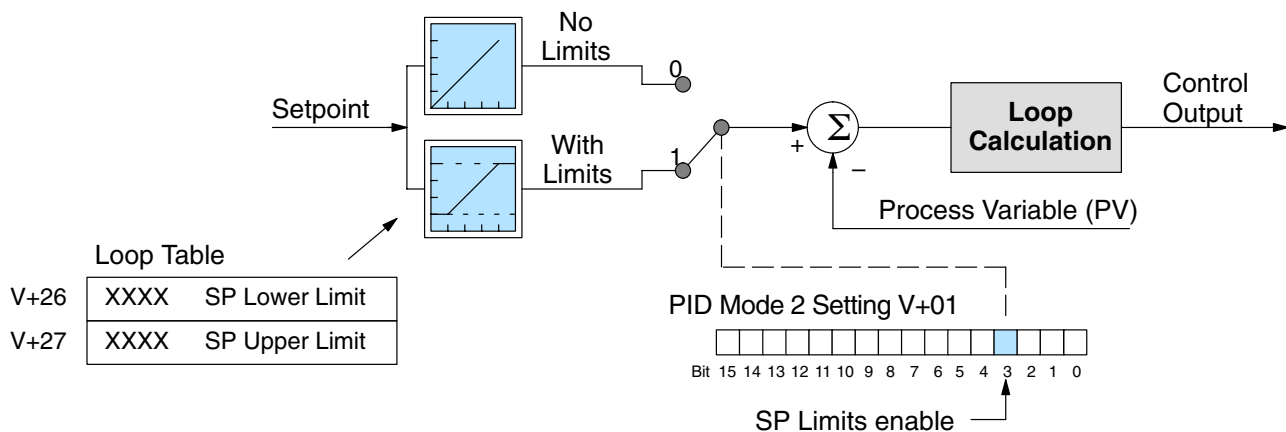
In many batch process applications, sensors or actuators interface to DL05 analog modules using 4–20 mA signals. This signal type has a built-in 20% offset, because the zero-point is a 4 mA instead of 0 mA. However, remember the analog modules convert the signals into data *and remove the offset at the 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 by 1.25.
- **Control Output** – In the event the Control Output is going to a device with 20% offset, all you need to do is have the ladder program write a value equivalent to the offset to the integrator register (V+04), before transitioning from Manual to Auto mode. The loop will then see this offset as a part of the process, taking care of it for you automatically.

## Setpoint (SP) Limits

The Setpoint in loop table location V+02 represents the desired value of the process variable. After selecting the data format for these variables, you can set limits on the range of SP values which the loop calculation will use. Many loops have two or more possible sources writing the Setpoint at various times, and the limits you set will help safeguard the process from the effects of a bad SP value.

In the figure below, the SP has a selectable limit function, enabled by PID Mode 2 Setting V+01 word, bit 3. If enabled, then locations V+26 and V+27 determine the lower and upper SP limits, respectively. The loop calculation applies this limit internally, so it is always possible to write any value to V+02.

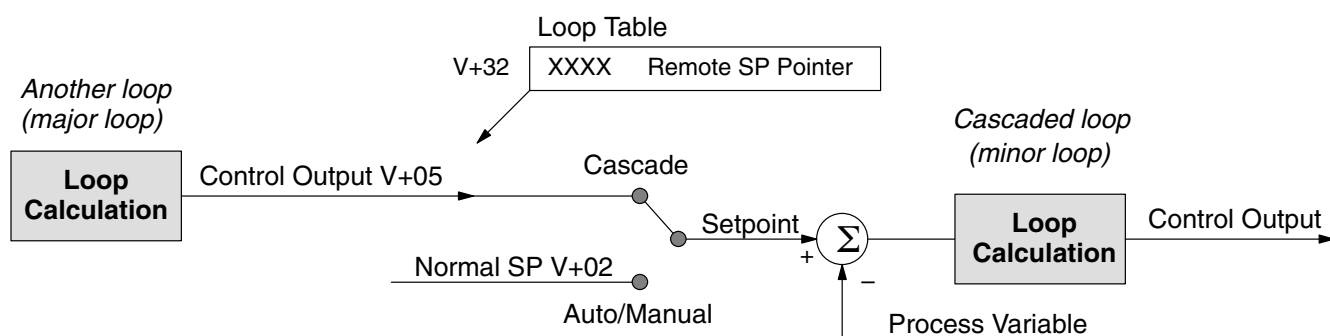


The loop calculation checks these SP upper and lower limits before each calculation. This means ladder logic can change the limit settings while a process is in progress, allowing you to keep a tighter guard band on the SP input value.

### Remote Setpoint (SP) Location

You may recall there are generally several possible data sources for the SP value. The PID loop controller has the built-in ability to select between two sources according to the current loop mode. Refer to the figure below. A loop reads its setpoint from table location V+02 in Auto or Manual modes. If you plan to use Cascade Mode for the loop at any time, then you must program its loop parameter table with a *remote setpoint pointer*.

The Remote SP pointer resides in location V+32 in the loop table. For loops that will be cascaded (made a minor loop), you will need to program this location with the address of the major loop's Control Output address. Find the starting location of the major loop's parameter table and add offset +05 to it.

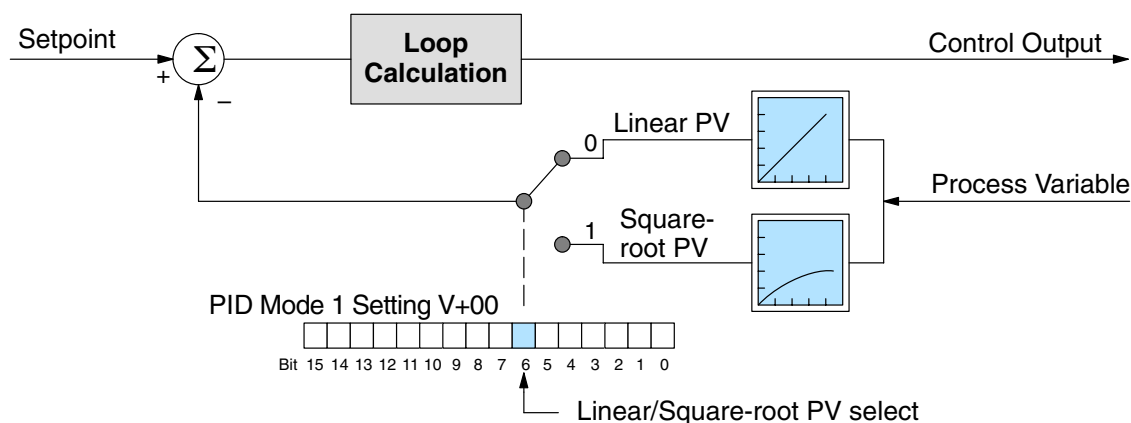


A **DirectSOFT32** Loop Setup dialog box will allow you to enter the Remote SP pointer if you know the address. Otherwise, you can enter it with a HPP or program it through ladder logic using the LDA instruction.

### Process Variable (PV) Configuration

The process variable input to each loop is the value the loop is ultimately trying to control, to make it equal to the setpoint and follow setpoint changes as quickly as possible. Most sensors for process variables have a primarily linear response curve. Most temperature sensors are mostly linear across their sensing range. However, flow sensing using an orifice plate technique gives a signal representing (approximately) the square of the flow. Therefore, a square-root extract function is necessary before using the signal in a linear control system (such as PID).

Some flow transducers are available which will do the square-root extract, but they add cost to the sensor package. The PID loop PV input has a selectable square-root extract function, pictured below. You can select between normal (linear) PV data, and data needing a square-root extract by using PID Mode setting V+00 word, bit 6.



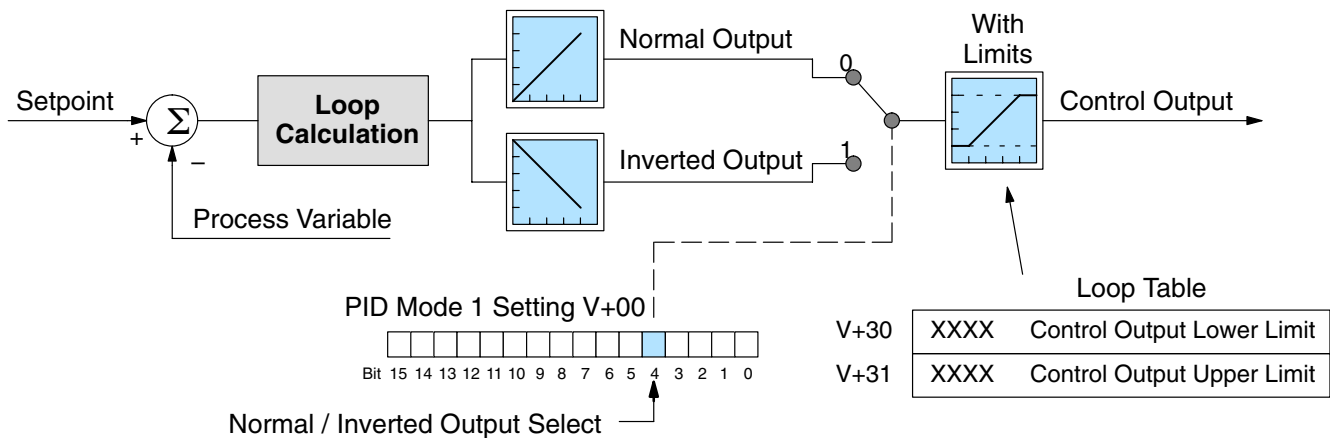
**IMPORTANT:** The scaling of the SP must be adjusted if you use PV square-root extract, because the loop drives the output so the *square root* of the PV is equal to the PV input. Divide the desired SP value by the square root of the analog span, and use the result in the V+02 location for the SP. This does reduce the resolution of the SP, but most flow control loops do not require a lot of precision (the recipient of the flow is integrating the errors). Use one of the following formulas for the SP according to the data format you are using. It's a good idea to set the SP upper limit to the top of the allowed range.

Data Format	SP Scaling	SP Range	PV range
12-bit	$SP = PV \text{ input} / 64$	0 – 64	0 – 4095
15-bit	$SP = PV \text{ input} / 181.02$	0 – 181	0 – 32767

### Control Output Configuration

The Control Output is the numerical result of the PID calculation. All of the other parameter choices ultimately influence the value of a loop's Control Output for each calculation. Some final processing selections dedicated to the Control Output are available, shown below. At the far right of the figure, the final output may be restricted by lower and upper limits that you program. The values for V+30 and V+31 may be set once using **DirectSOFT32'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 **DirectSOFT**). Use bit 4 of the PID Mode 1 Setting V+00 word to configure the output. Independently of unipolar or bipolar format, a normal output goes upward on positive errors and downward on negative errors (where  $Error = (SP - PV)$ ). The inverted output reverses the direction of the output change.

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

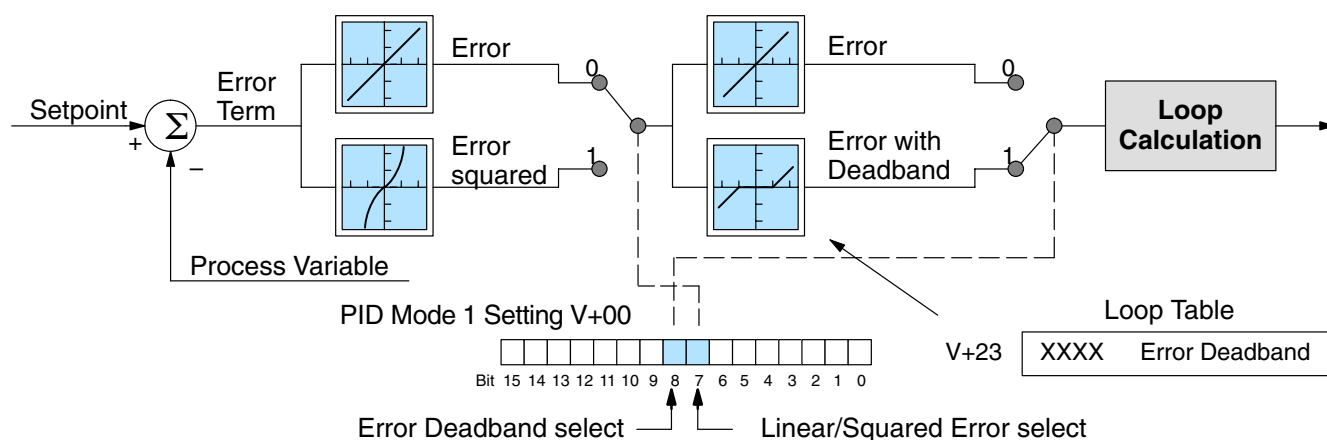
## Error Term Configuration

The Error term is internal to the CPU's 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:  $\text{Error} = (\text{SP} - \text{PV})$ . If the PV square-root extract is enabled, then  $\text{Error} = (\text{SP} - (\text{sqrt}(\text{PV})))$ . In any case, the size of the error and algebraic sign determine the next change of the control output for each PID calculation.

Now we will superimpose some "special effects" on to the error term as described. Refer to the diagram below. Bit 7 of the PID Mode Setting 1 V+00 word lets you select a linear or squared error term, and bit 8 enables or disables the error deadband.



**NOTE:** When first configuring a loop, it's best to use the standard error term. After the loop is tuned, then you will be able to tell if these functions will enhance control.



**Error Squared** – When selected, the squared error function simply squares the error term (but preserves the original algebraic sign), which is used in the calculation. This affects the Control Output by diminishing its response to smaller error values, but maintaining its response to larger errors. Some situations in which the error squared term might be useful:

- Noisy PV signal – using a squared error term can reduce the effect of low-frequency electrical noise on the PV, which will make the control system jittery. A squared error maintains the response to larger errors.
- Non-linear process – some processes (such as chemical pH control) require non-linear controllers for best results. Another application is surge tank control, where the Control Output signal must be smooth.

**Error Deadband** – When selected, the error deadband function takes a range of small error values near zero, and simply substitutes zero as the value of the error. If the error is larger than the deadband range, then the error value is used normally.

Loop parameter location V+23 must be programmed with a desired deadband amount. Units are the same as the SP and PV units (0 to FFF in 12-bit mode, and 0 to 7FFF in 15-bit mode). The PID loop controller automatically applies the deadband symmetrically about the zero-error point.

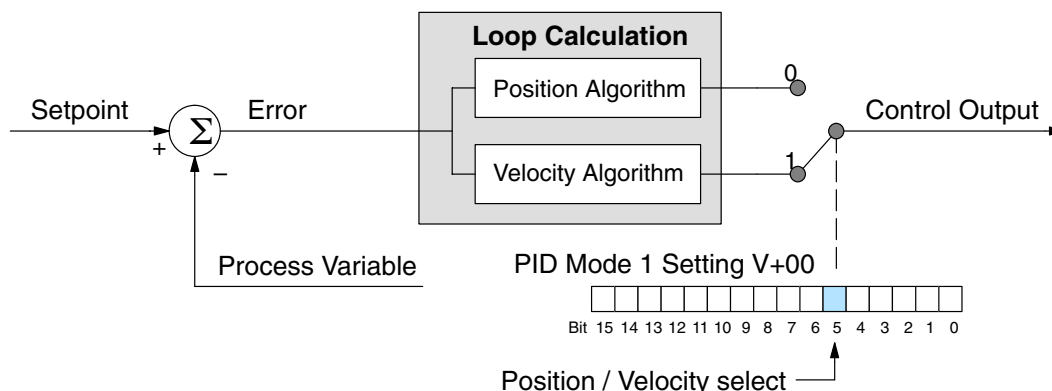
## PID Algorithms

The Proportional–Integral–Derivative (PID) algorithm is widely used in process control. The PID method of control adapts well to electronic solutions, whether implemented in analog or digital (CPU) components. The DL05 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 vice-versa).

The DL05 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 **DirectSOFT32's** PID View Setup dialog box to select the desired algorithm. Or, use bit 5 of PID Mode 1 Setting V+00 word as shown below to select the algorithm.



**NOTE:** The selection of a PID algorithm is very fundamental to control loop operation, and is normally never changed after the initial configuration of a loop.

**Position Algorithm** The Position Algorithm causes the PID equation to calculate the Control Output  $M_n$ :

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

In the formula above, the sum of the integral terms and the initial output are combined into the “Bias” term,  $M_x$ . Using the bias term, we define formulas for the Bias and Control Output as a function of sampling time:

$$M_{x0} = M_o$$

$$M_{xn} = K_i * e_n + M_{xn-1}$$

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

$$M_n = K_c * e_n + K_r * (e_n - e_{n-1}) + M_{xn} \dots \text{Output for sampling time “n”}$$

The position algorithm variables and related variables are:

$T_s$  = Sample rate  
 $K_c$  = Proportional gain  
 $K_i = K_c * (T_s/T_i)$  coefficient of integral term  
 $K_r = K_c * (T_d/T_s)$  coefficient of derivative term  
 $T_i$  = Reset time (integral time)  
 $T_d$  = Rate time (derivative time)  
 $SP_n$  = Set Point for sampling time "n" (SP value)  
 $PV_n$  = Process variable for sampling time "n" (PV)  
 $e_n = SP_n - PV_n$  = Error term for sampling time "n"  
 $M_0$  = Control Output for sampling time "0"  
 $M_n$  = Control Output for sampling time "n"

Analysis of these equations will be found in most good text books on process control. At a glance, we can isolate the parts of the PID Position Algorithm which correspond to the P, I, and D terms, and the Bias as shown below.

$$M_n = K_c * e_n + K_i * \sum_{i=1}^n e_i + K_r * (e_n - e_{n-1}) + M_0$$

The initial output is the output value assumed from Manual mode control when the loop transitioned to Auto Mode. The sum of the initial output and the integral term is the bias term, which holds the "position" of the output. Accordingly, the Velocity Algorithm discussed next does not have a bias component.

**Velocity Algorithm** The Velocity Algorithm form of the PID equation can be obtained by transforming Position Algorithm formula with subtraction of the equation of (n-1)th degree from the equation of nth degree.

The velocity algorithm variables and related variables are:

$T_s$  = Sample rate  
 $K_c$  = Proportional gain  
 $K_i = K_c * (T_s/T_i)$  = coefficient of integral term  
 $K_r = K_c * (T_d/T_s)$  = coefficient of derivative term  
 $T_i$  = Reset time (integral time)  
 $T_d$  = Rate time (derivative time)  
 $SP_n$  = Set Point for sampling time "n" (SP value)  
 $PV_n$  = Process variable for sampling time "n" (PV)  
 $e_n = SP_n - PV_n$  = Error term for sampling time "n"  
 $M_n$  = Control Output for sampling time "n"

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

$$\Delta M_n = M_n - M_{n-1}$$

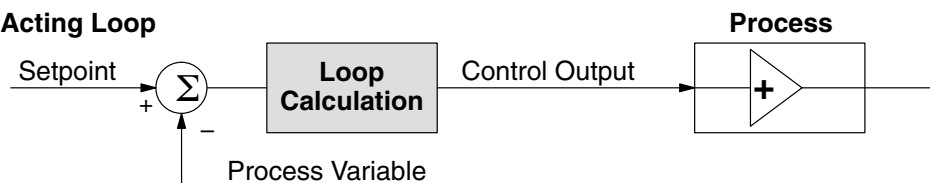
$$\Delta M_n = K_c * (e_n - e_{n-1}) + K_i * e_n + K_r * (e_n - 2 * e_{n-1} + e_{n-2})$$

## Direct-Acting and Reverse-Acting Loops

The gain of a process determines, in part, how it must be controlled. The process shown in the diagram below has a positive gain, which we call “direct-acting”. This means that when the control output increases, the process variable also eventually increases. Of course, a true process is usually a complex transfer function that includes time delays. Here, we are only interested in the direction of change of the process variable in response to a control output change.

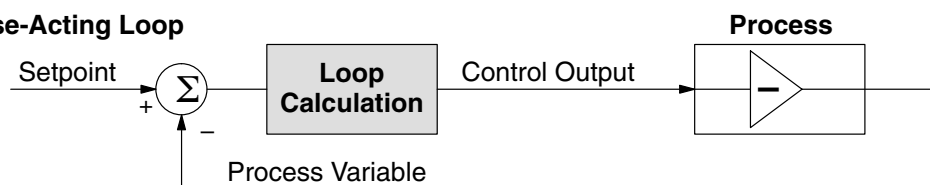
Most process loops will be direct-acting, such as a temperature loop. An increase in the heat applied increases the PV (temperature). Accordingly, direct-acting loops are sometimes called *heating loops*.

### Direct-Acting Loop



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*.

### Reverse-Acting Loop



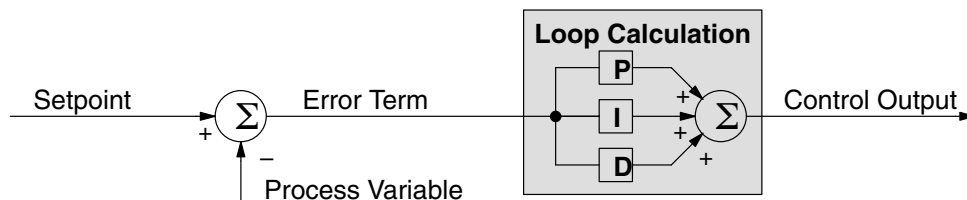
### It is crucial to know whether a particular loop is direct or reverse-acting!

Unless you are controlling temperature, there is no obvious answer. In a flow control loop, a valve positioning circuit can be configured and wired reverse-acting as easily as direct-acting. One easy way to find out is to run the loop in Manual Mode, where you must manually generate control output values. Observe whether the PV goes up or down in response to a step increase in the control output.

To run a loop in Auto or Cascade Mode, the control output must be correctly programmed (refer to the previous section on Control Output Configuration). Use “normal output” for direct-acting loops, and “inverted output” for reverse-acting loops. To compensate for a reverse-acting loop, the PID controller must know to invert the control output. If you have a choice, configure and wire the loop to be direct-acting. This will make it easier to view and interpret loop data during the loop tuning process.

## P-I-D Loop Terms

You may recall the introduction of the position and velocity forms of the PID loop equations. The equations basically show the three components of the PID calculation. The following figure shows a schematic form of the PID calculation, in which the control output is the sum of the proportional, integral and derivative terms. On each calculation of the loop, each term receives the same error signal value.

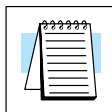
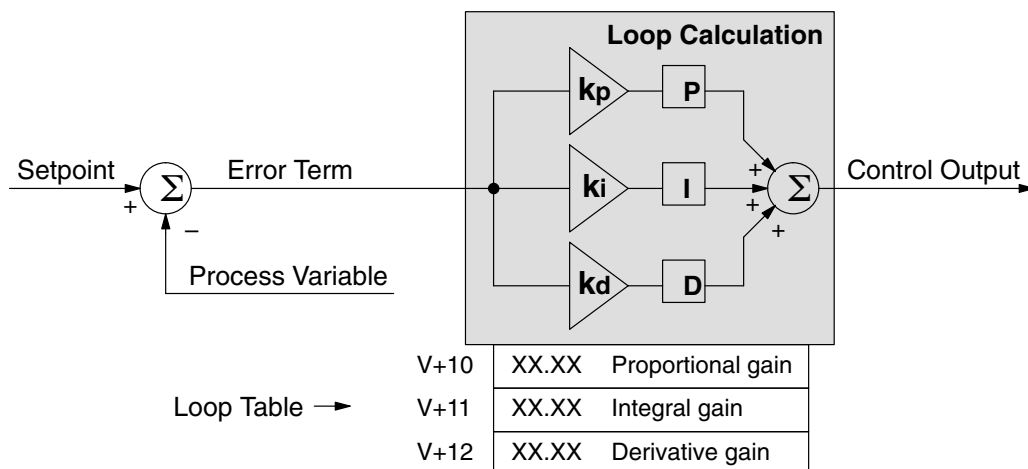


The role of the P, I, and D terms in the control task are as follows:

- **Proportional** – the proportional term simply responds proportionally to the current size of the error. This loop controller calculates a proportional term value for each PID calculation. When the error is zero, the proportional term is also zero.
- **Integral** – the integrator (or reset) term integrates (sums) the error values. Starting from the first PID calculation after entering Auto Mode, the integrator keeps a running total of the error values. For the position form of the PID equation, when the loop reaches equilibrium and there is no error, the running total represents the constant output required to hold the current position of the PV.
- **Derivative** – the derivative (or rate) term responds to change in the current error value from the error used in the previous PID calculation. Its job is to anticipate the probable growth of the error and generate a contribution to the output in advance.

The P, I, and D terms work together as a team. To do that effectively, they will need some additional instructions from us. The figure below shows the P, I, and D terms contain programmable **gain** values  $k_p$ ,  $k_i$ , and  $k_d$  respectively. The values reside in the loop table in the locations shown. The goal of the loop tuning process (covered later) is to derive gain values that result in good overall loop performance.

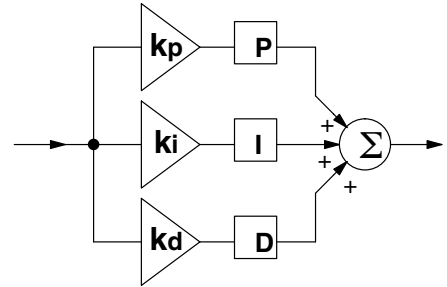
**NOTE:** The proportional gain is also simply called “gain”, in PID loop terminology.



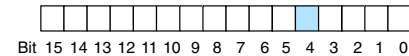


The P, I and D gains are 4-digit BCD numbers with values from 0000 to 9999. They contain an implied decimal point in the middle, so the values are actually 00.00 to 99.99. Some gain values have units – Integral gain may be in units of seconds or minutes, by programming the bit shown. Derivative gain is in seconds.

V+10	XX.XX P gain	–
V+11	XX.XX I gain	0=sec, 1=min.
V+12	XX.XX D gain	sec.



PID Mode 2 Setting V+01



In **DirectSOFT32'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.

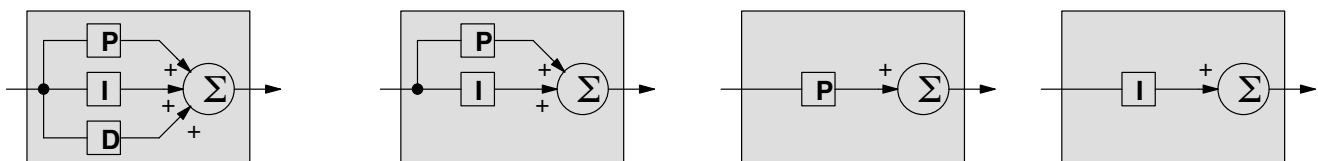
**Derivative Gain** – Values range from 0001 to 9999, but they are used internally as xx.xx. An entry of “0000” allows removal of the derivative term from the PID equation (a common practice). This accommodates applications which need proportional and/or integral-only loops. The derivative term has an 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( $1/T_s$ ), so smaller numbers produce more gain and larger numbers produce less gain. *This is very important to know during loop tuning.*

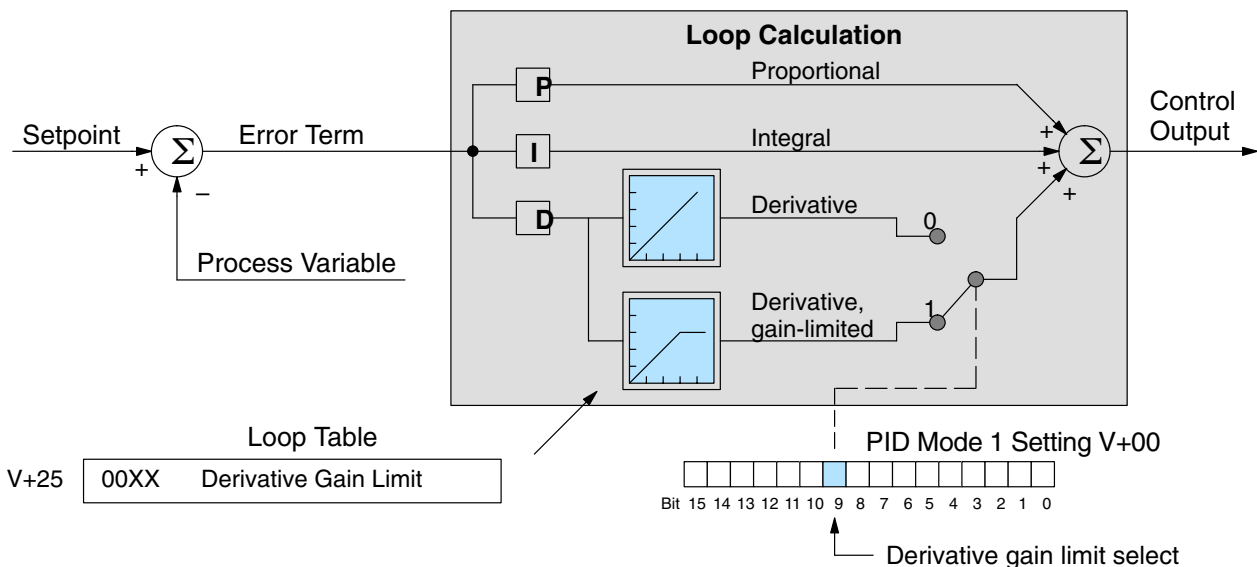
### Using a Subset of PID Control

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 DL05. We do not recommend using any other combination of control, because most of them are inherently unstable.



### Derivative Gain Limiting

The derivative term is unique in that it has an optional gain-limiting feature. This is provided because the derivative term reacts badly to PV signal noise or other causes of sudden PV fluctuations. The function of the gain-limiting is shown in the diagram below. Use bit 9 of PID Mode 1 Setting V+00 word to enable the gain limit.



The derivative gain limit in location V+25 must have a value between 0 and 20, in BCD format. This setting is operational only when the enable bit = 1.

The gain limit can be particularly useful during loop tuning. Most loops can tolerate only a little derivative gain without going into wild oscillations.

### Bias Term

In the widely-used *position* form of the PID equation, an important component of the control output value is the bias term shown below. Its location in the loop table is in V+04. the loop controller writes a new bias term after each loop calculation.

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

Control Output

Proportional Term

Integral Term

Derivative Term

Initial Output

V+04 XXXX Bias term

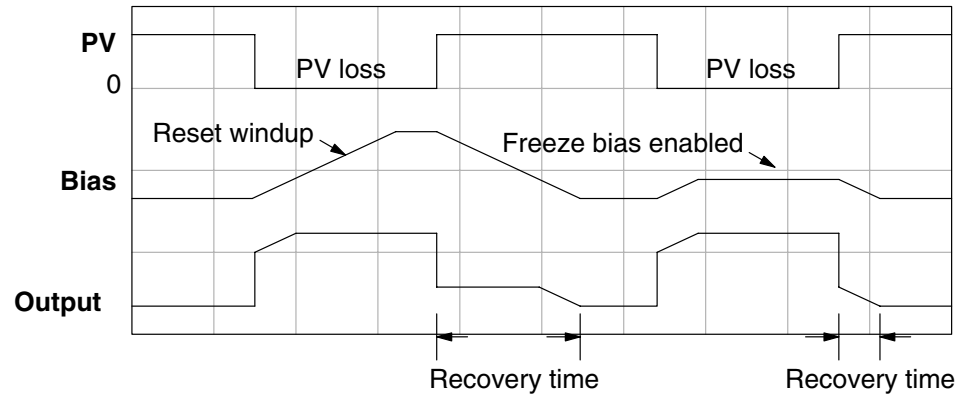
Bias Term

If we cause the error ( $e_n$ ) to go to zero for two or more sample periods, the proportional and derivative terms cancel. The bias term is the sum of the integral term and the initial output ( $M_o$ ). It represents the steady, constant part of the control output value, and is similar to the DC component of a complex signal waveform.

The bias term value establishes a “working region” for the control output. *When the error fluctuates around its zero point, the output fluctuates around the bias value.* This concept is very important, because it shows us why the integrator term must respond more slowly to errors than either the proportional or derivative terms.

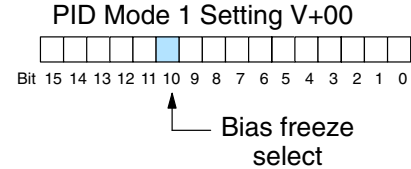
## Bias Freeze

The term “reset windup” refers to an undesirable characteristic of integrator behavior which occurs naturally under certain conditions. Refer to the figure below. Suppose the PV signal becomes disconnected, and the PV value goes to zero. While this is a serious loop fault, it is made worse by *reset windup*. Notice the bias (reset) term keeps integrating normally during the PV disconnect, until its upper limit is reached. When the PV signal returns, the bias value is saturated (windup) and takes a long time to return to normal. The loop output consequently has an extended recovery time. Until recovery, the output level is wrong and causes further problems.



In the second PV signal loss episode in the figure, the freeze bias feature is enabled. It causes the bias value to freeze when the control output goes out of bounds. Much of the reset windup is thus avoided, and the output recovery time is much less.

For most applications, the freeze bias feature will work with the loop as described above. You may enable the feature using the **DirectSOFT32** PID View setup dialog, or set bit 10 of PID Mode 1 Setting word as shown to the right.



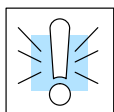
**NOTE:** The bias freeze feature stops the bias term from changing when the control output reaches the end of the data range. If you have set limits on the control output other than the range (i.e., 0–4095 for a unipolar/12bit loop), the bias term still uses the end of range for the stopping point and 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 DL05 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 DL05 is not intended to perform as a replacement for your process knowledge.

### Open-Loop Test

Whether you use manual or auto tuning, it is very important to verify basic characteristics of a newly-installed process before attempting to tune it. With the loop in Manual Mode, verify the following items for each new loop.

- **Setpoint** – verify the source which is to generate the setpoint can do so. You can put the PLC in Run Mode, but leave the loop in Manual Mode. Then monitor the loop table location V+02 to see the SP value(s). The ramp/soak generator (if you are using it) should be tested now.
- **Process Variable** – verify the PV value is an accurate measurement, and the PV data arriving in the loop table location V+03 is correct. If the PV signal is very noisy, consider filtering the input either through hardware (RC low-pass filter), or using a digital S/W filter.
- **Control Output** – if it is safe to do so, manually change the output a small amount (perhaps 10%) and observe its affect on the process variable. Verify the process is direct-acting or reverse acting, and check the setting for the control output (inverted or non-inverted). Make sure the control output upper and lower limits are not equal to each other.
- **Sample Rate** – while operating open-loop, this is a good time to find the ideal sample rate (procedure give earlier in this chapter). However, if you are going to use auto tuning, note the auto tuning procedure will automatically calculate the sample rate in addition to the PID gains.

The discussion beginning on the following page covers the manual tuning procedure. If want to perform only auto tuning, please skip the next section and proceed directly to the section on auto tuning.

## Manual Tuning Procedure

Now comes the exciting moment when we actually close the loop (go to Auto Mode) for the first time. Use the following checklist **before** switching to Auto mode:

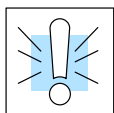
- Monitor the loop parameters with a loop trending instrument. We recommend using the PID view feature of **DirectSOFT**.



**NOTE:** We recommend using the PID trend view setup menu to select the vertical scale feature to *manual*, for both SP/PV area and Bias/Control Output areas. The auto scaling feature will otherwise change the vertical scale on the process parameters and add confusion to the loop tuning process.

- Adjust the gains so the Proportional Gain = 10, Integrator Gain = 9999, and Derivative Gain = 0000. This disables the integrator and derivative terms, and provides a little proportional gain.
- Check the bias term value in the loop parameter table (V+04). If it is not zero, then write it to zero using **DirectSOFT32** or HPP, etc.

**Now we can transition the loop to Auto Mode.** Check the mode monitoring bits to verify its true mode. If the loop will not stay in Auto Mode, check the troubleshooting tips at the end of this chapter.



**CAUTION:** If the PV and Control Output values begin to oscillate, reduce the gain values immediately. If the loop does not stabilize immediately, then transfer the loop back to Manual Mode and manually write a safe value to the control output. **During the loop tuning procedure, always be near the Emergency Stop switch which controls power to the loop actuator in case a shutdown is necessary.**

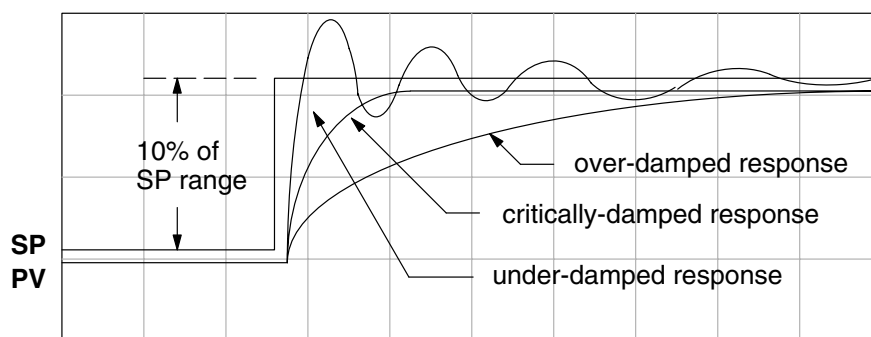
- At this point, the SP should = PV because of the bumpless transfer feature. Increase the SP a little, in order to develop an error value. With only the proportional gain active and the bias term=0, we can easily check the control output value:

$$\text{Control Output} = (\text{SP} - \text{PV}) \times \text{proportional gain}$$

- 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 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.



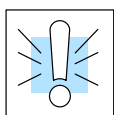
Now you may want to add a little derivative gain to further improve the critically-damped response above. Note the proportional and integral gains will be very close to their final values at this point. Adding some derivative action will allow you to increase the proportional gain slightly without causing loop oscillations. The derivative action tends to tame the proportional response slightly, so adjust these gains together.

Autotuning is initiated within **DirectSOFT32**. 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.

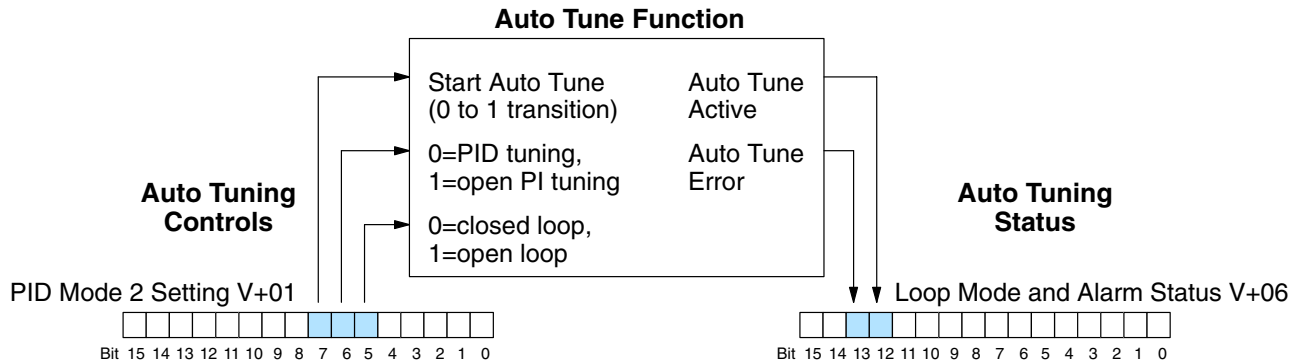
**WARNING:** Only authorized personnel fully familiar with all aspects of the process should make changes that affect the loop tuning constants. Using the loop auto tuning procedures will affect the process, including inducing large changes in the control output value. Make sure you thoroughly consider the impact of any changes to minimize the risk of injury to personnel or damage to equipment. The auto tune in the DL05 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.

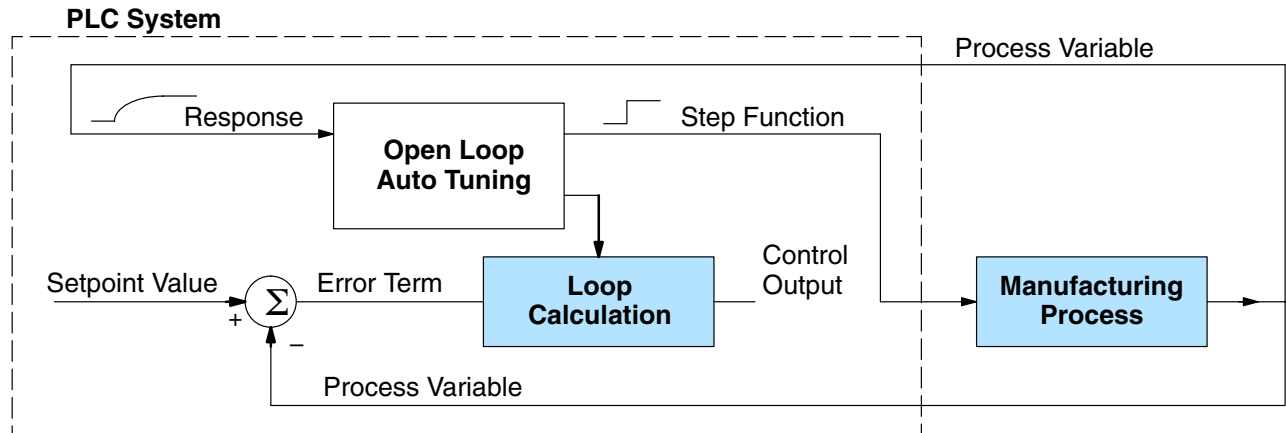
### Auto Tuning Procedure



The controls for the auto tuning function use three bits in the PID Mode 2 word V+01, as shown below. **DirectSOFT32** will manipulate these bits automatically when you use the auto tune feature within **DirectSOFT**. Or, you may have ladder logic access these bits directly for allowing control from another source such as a dedicated operator interface. The individual control bits let you to start the auto tune procedure, select PID or PI tuning, and select closed-loop or open-loop tuning. If you select PI tuning, the auto tune procedure leaves the derivative gain at 0. The Loop Mode and Alarm Status word V+06 reports the auto tune status as shown. Bit 12 will be on (1) when during the auto tuning cycle, automatically returning to off (0) when done.



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



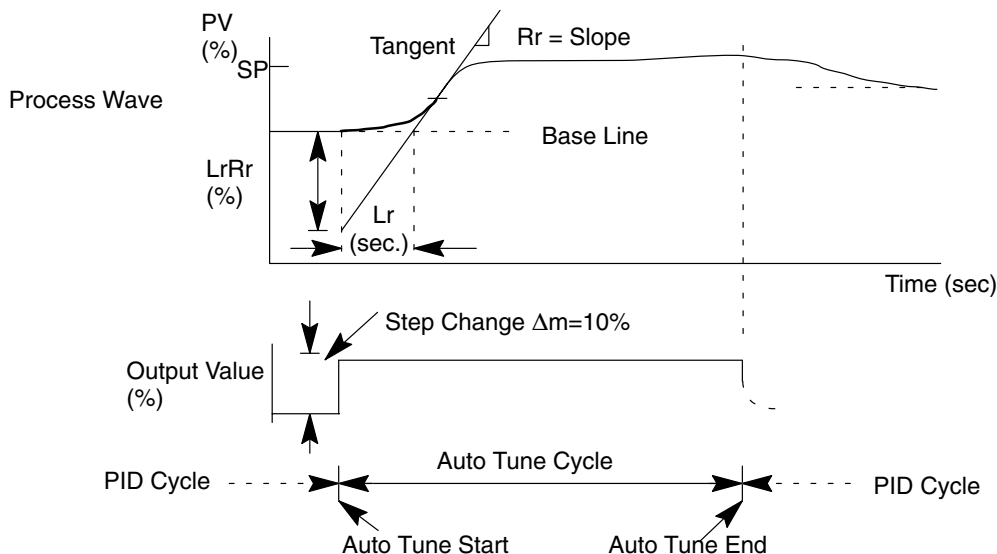
**NOTE:** In theory, the SP value does not matter in this case, because the loop is not closed. However, the firmware requires that the SP value be more than 205 counts away from the PV value before starting the auto tune cycle (205 counts or more below the SP for forward-acting loops, or 205 counts or more above the SP for reverse-acting loops).

When auto tuning, the loop controller induces a step change on the output and simply observes the response of the PV. From the PV response, the auto tune function calculates the gains and the sample time. It automatically places the results in the corresponding registers in the loop table.



The following timing diagram shows the events which occur in the open-loop auto tuning cycle. The auto tune function takes control of the control output and induces a 10%-of-span step change. If the PV change which the loop controller observes is less than 2%, then the step change on the output is increased to 20%-of-span.

### Open Loop Auto Tune Cycle Wave: Step Response Method



- \* When Auto Tune starts, step change output  $\Delta m = 10\%$
- \* During Auto Tune, the controller output reached the full scale positive limit. Auto Tune stopped and the Auto Tune Error bit in the Alarm word bit turned on.
- \* When PV change is under 2%, output is changed at 20%.

When the loop tuning observations are complete, the loop controller computes  $R_r$  (maximum slope in %/sec.) and  $L_r$  (dead time in sec). The auto tune function computes the gains according to the Ziegler-Nichols equations, shown below:

#### PID tuning:

$$\begin{aligned}
 P &= 1.2 * \Delta m / L_r R_r \\
 I &= 2.0 * L_r \\
 D &= 0.5 * L_r \\
 \text{Sample Rate} &= 0.056 * L_r
 \end{aligned}$$

#### PI tuning:

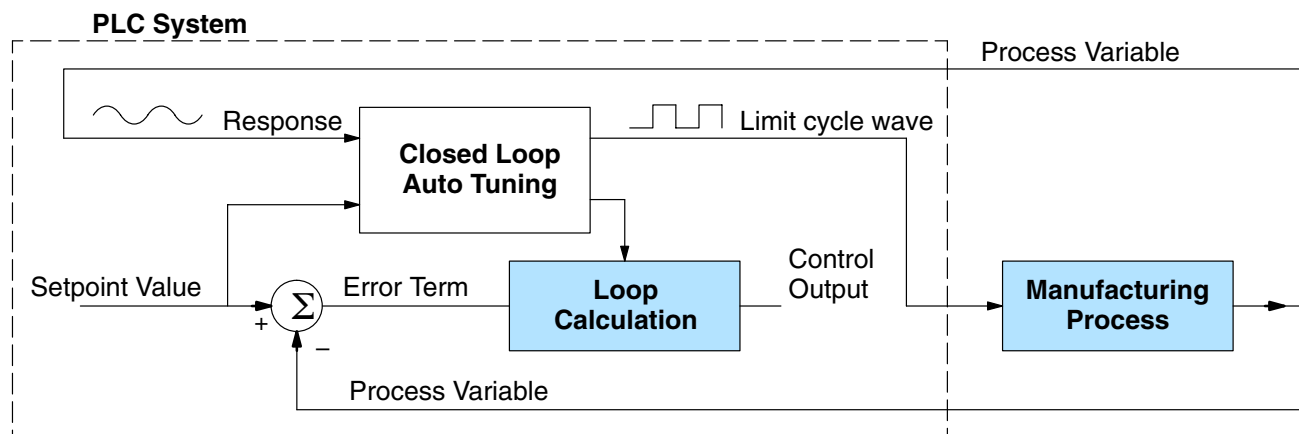
$$\begin{aligned}
 P &= 0.9 * \Delta m / L_r R_r \\
 I &= 3.33 * L_r \\
 D &= 0 \\
 \text{Sample Rate} &= 0.12 * L_r
 \end{aligned}$$

$$\Delta m = \text{Output step change (10\% = 0.1, 20\% = 0.2)}$$

We highly recommend using **DirectSOFT32** for the auto tuning interface. The duration of each auto tuning cycle will depend on the mass of our process. A slowly-changing PV will result in a longer auto tune cycle time. When the auto tuning is complete, the proportional, integral, and derivative gain values are automatically updated in loop table locations V+10, V+11, and V+12 respectively. The sample time in V+07 is also updated automatically. You can test the validity of the values the auto tuning procedure yields by measuring the closed-loop response of the PV to a step change in the output. The instructions on how to do this are in the section on the manual tuning procedure (located prior to this section on auto tuning).



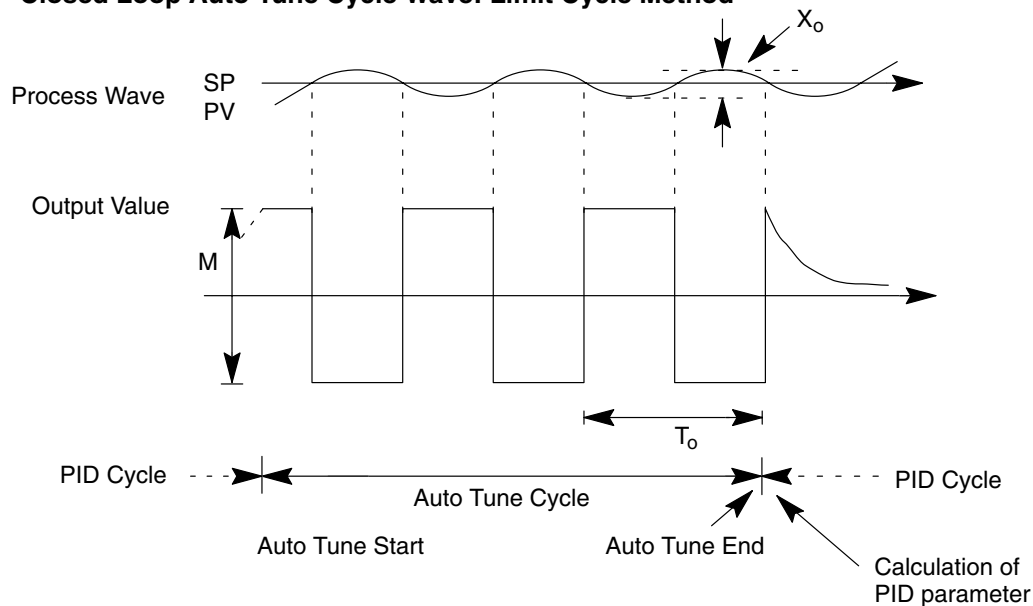
**Closed-Loop Auto Tuning** – During a closed-loop auto tuning cycle, the loop controller operates as shown in the diagram below.



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.

#### Closed Loop Auto Tune Cycle Wave: Limit Cycle Method



\* $M_{\max}$  = Output Value upper limit setting  $M_{\min}$  = 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  $K_{pc}$  (sensitive limit) and  $T_{pc}$  (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:

$$K_{pc} = 4M / (\pi * X_o) \quad T_{pc} = 0$$

$M$  = amplitude of output

#### PID tuning:

$$P = 0.45 * K_{pc}$$

$$I = 0.60 * T_{pc}$$

$$D = 0.10 * T_{pc}$$

$$\text{Sample Rate} = 0.014 * T_{pc}$$

#### PI tuning:

$$P = 0.30 * K_{pc}$$

$$I = 1.00 * T_{pc}$$

$$D = 0$$

$$\text{Sample Rate} = 0.03 * T_{pc}$$

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



**NOTE:** If your PV fluctuates rapidly, you probably need to use the built-in analog filter (see page 8-45) or create a filter in ladder logic (see example on page 8-46).

### Tuning Cascaded Loops

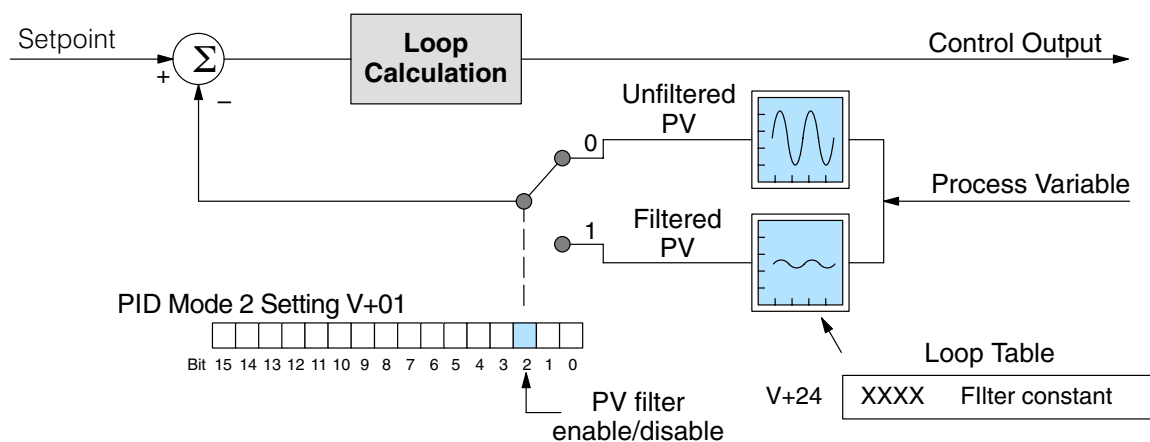
In tuning cascaded loops, we will need to de-couple the cascade relationship and tune the loops individually, using one of the loop tuning procedures previously covered.

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.

## The DL05 Built-in 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 DL05's built-in filter. The second method achieves a similar result using ladder logic.

The DL05 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 V+24 in the loop parameter table, the filter constant. The data format of the filter constant value is BCD, with an implied decimal point 00X.X, as follows:

- The filter constant has a 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
- 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 **DirectSOFT32** 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 V+10, V+11, and V+12 respectively. The sample time in V+07 is also updated automatically. You can test the validity of the values the auto tuning procedure yields by measuring the closed-loop response of the PV to a step change in the output. The instructions on how to do this are in the section on the manual tuning procedure.

The algorithm which the built-in filter follows is:

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

$y_i$  is the current output of the filter

$x_i$  is the current input to the filter

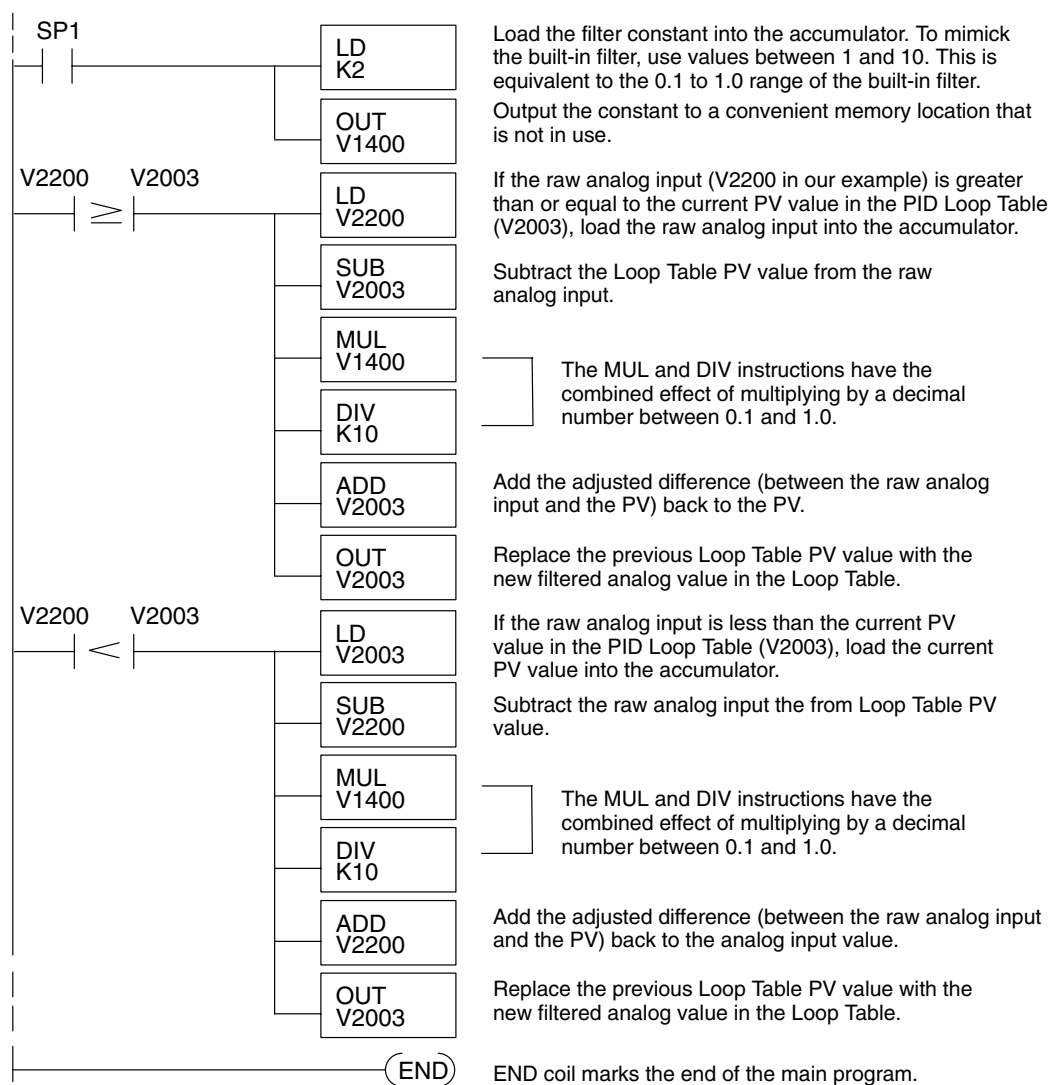
$y_{i-1}$  is the previous output of the filter

$k$  is the PV Analog Input Filter Factor

### Creating an Analog Filter in Ladder Logic

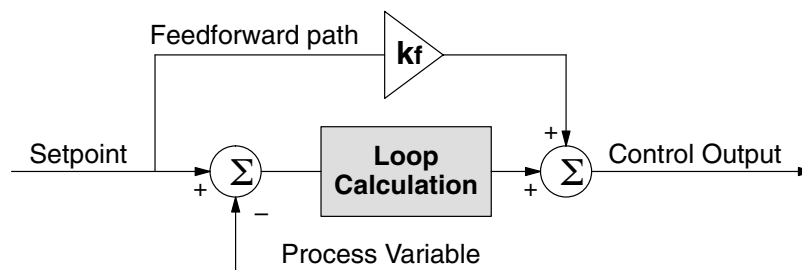
A similar algorithm can be built in your ladder program. Your analog inputs can be filtered effectively using either method. The following programming example describes the ladder logic you will need. Be sure to change the example memory locations to those that fit your application.

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 DL05. 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.

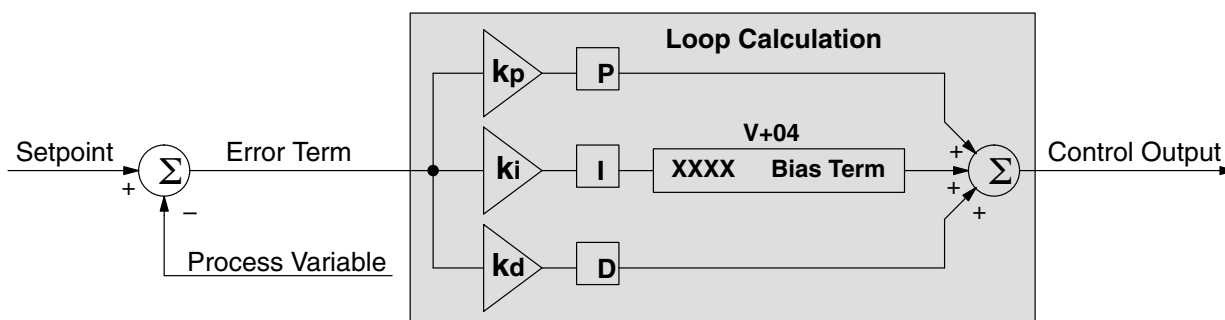


In the previous section on the bias term, we said that “the bias term value establishes a “working region” or operating point for the control output. *When the error fluctuates around its zero point, the output fluctuates around the bias value.*” Now, when there is a change in setpoint, an error is generated 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 DL05 loop controller, as shown below. The bias term has been made available to the user in a special read/write location, at PID Parameter Table location V+04.



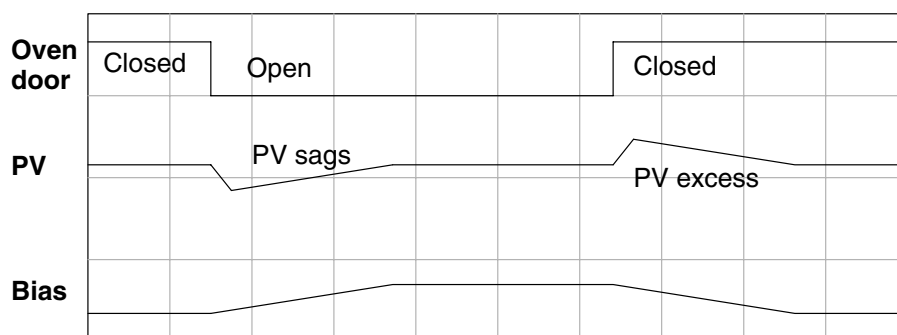
To change the bias (operating point), ladder logic only has to write the desired value to V+04. The PID loop calculation first reads the bias value from V+04 and modifies the value based on the current integrator calculation. Then it writes the result back to location V+04. This arrangement creates a sort of “transparent” bias term. All you have to do to implement feed forward control is write the correct value to the bias term at the right time (the example below shows you how).



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

### Feedforward Example

How do we know when to write to the bias term, and what value to write? Suppose we have an oven temperature control loop, and we have already tuned the loop for optimal performance. Refer to the figure below. We notice that when the operator opens the oven door, the temperature sags a bit while the loop bias adjusts to the heat loss. Then when the door closes, the temperature rises above the SP until the loop adjusts again. Feedforward control can help diminish this effect.



First, we record the amount of bias change the loop controller generates when the door opens or closes. Then, we write a ladder program to monitor the position of an oven door limit switch. When the door opens, our ladder program reads the current bias value from V+04, adds the desired change amount, and writes it back to V+04. When the door closes, we duplicate the procedure, but subtracting desired change amount instead. The following figure shows the results.



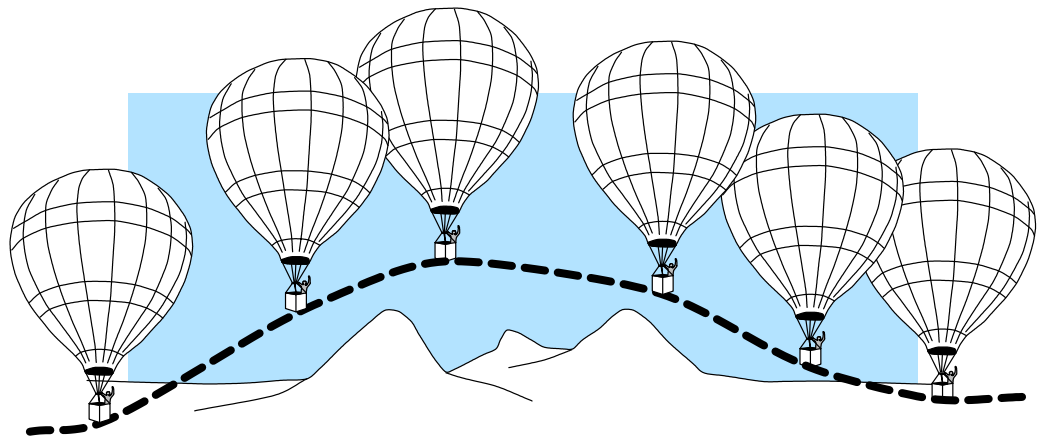
The step changes in the bias are the result of our two feed-forward writes to the bias term. We can see the PV variations are greatly reduced. The same technique may be applied for changes in setpoint.

## Time-Proportioning Control

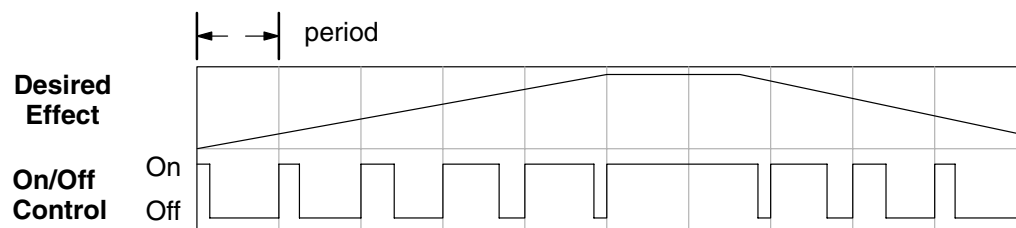
The PID loop controller in the DL05 CPU generates a smooth control output signal across a numerical range. The control output value is suitable to drive an analog output module, which connects to the process. In the process control field, this is called *continuous control*, because the output is on (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*.



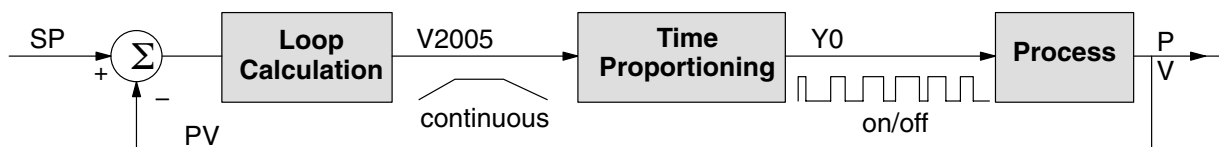
Time-proportioning control approximates continuous control by virtue of its duty-cycle – the ratio of ON time to OFF time. The following figure shows an example of how duty cycle approximates a continuous level when it is averaged by a large process mass.



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.

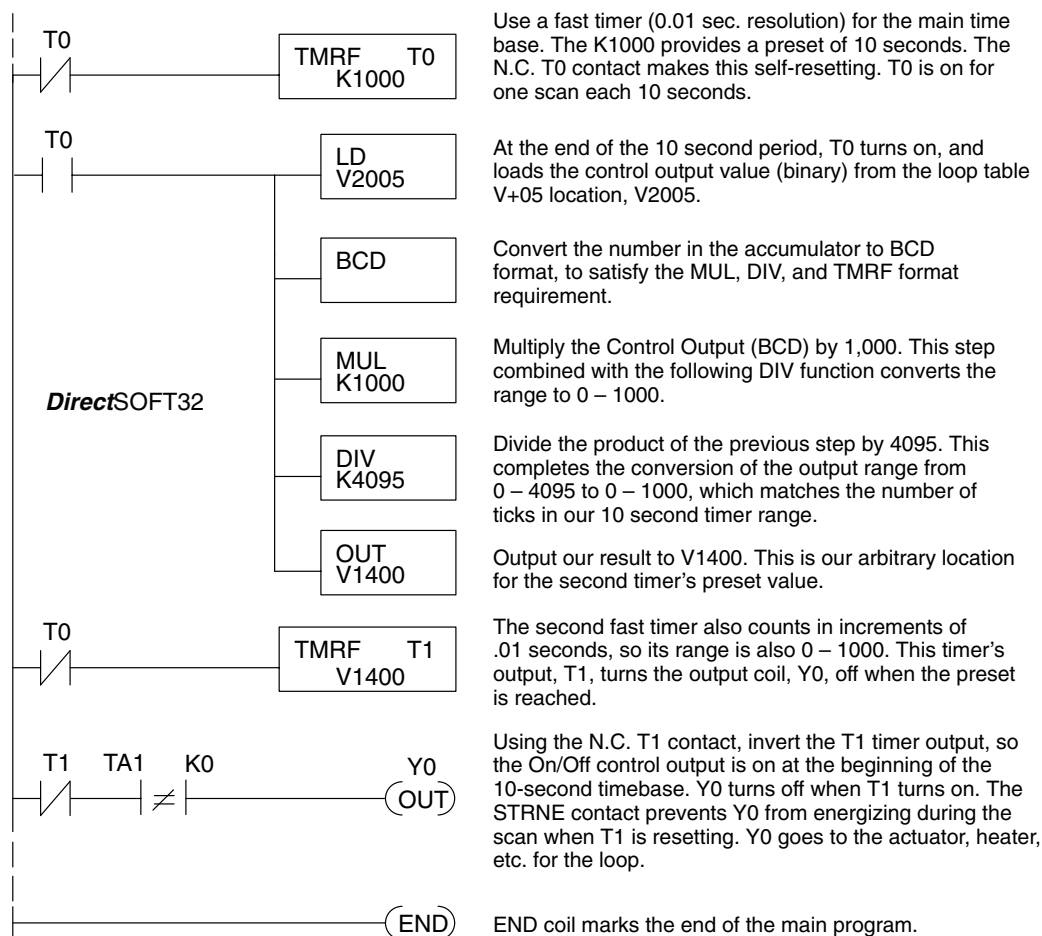


The example program uses two timers to generate On/Off control. It makes the following **assumptions**, which you can alter to fit your application:

- 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).
- The time base (one full cycle) for the On/Off waveform is 10 seconds. We use a fast timer (0.01 sec/tick), counting to 1000 ticks (10 seconds).
- 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).

**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

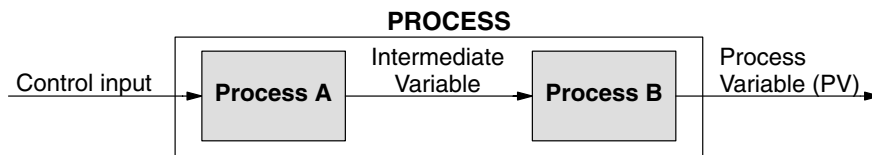
### Introduction



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 DL05 also provides Cascaded Mode.

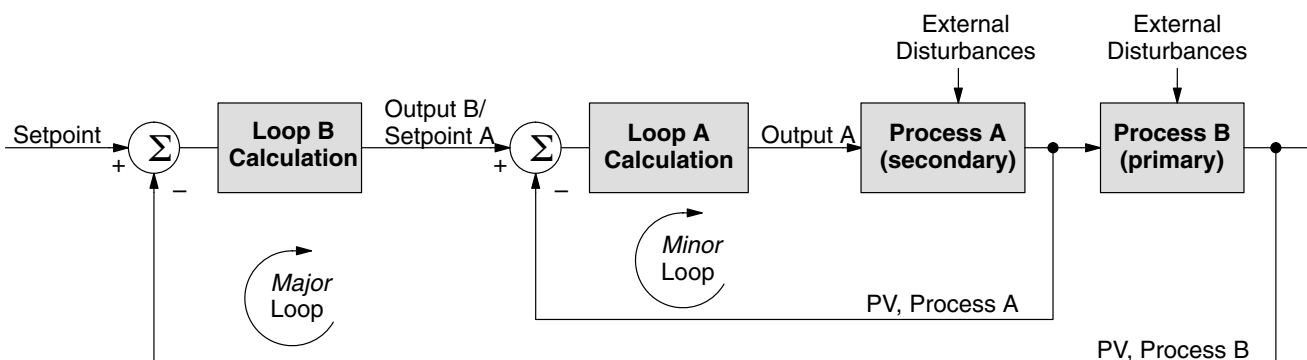
**NOTE:** Cascaded loops are an advanced process control technique. Therefore we recommend their use only for experienced process control engineers.

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.



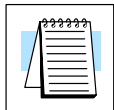
*The principle of cascaded loops is simply that we add another process loop to more precisely control the intermediate variable!* This separates the source of the control lag into two parts, as well.

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 DL05 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.

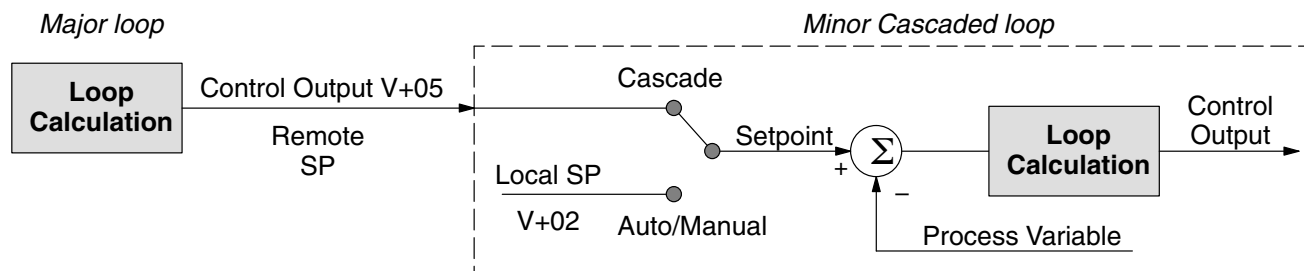
You can cascade together as many loops as necessary on the DL05, and you may have multiple groups of cascaded loops. For proper operation on cascaded loops you must use the same data range (12/15 bit) and unipolar/bipolar settings on the major and minor loop.

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

Major Loop (Auto mode)		Minor Loop (Cascade Mode)	
Loop Table		Loop Table	
V+02	XXXX SP	V+02	<del>XXXX SP</del>
V+03	XXXX PV	V+03	XXXX PV
V+05	XXXX Control Output	V+05	XXXX Control Output
		V+32	XXXX Remote SP Pointer

When using **DirectSOFT32**’s PID View to watch the SP value of the minor loop, **DirectSOFT32** automatically reads the major loop’s control output and displays it for the minor loop’s SP. The minor loop’s normal SP location, V+02, remains unchanged.

Now, we use the loop parameter arrangement above and draw its equivalent loop schematic, shown below.



Remember that a major loop goes to Manual Mode automatically if its minor loop is taken out of Cascade Mode.

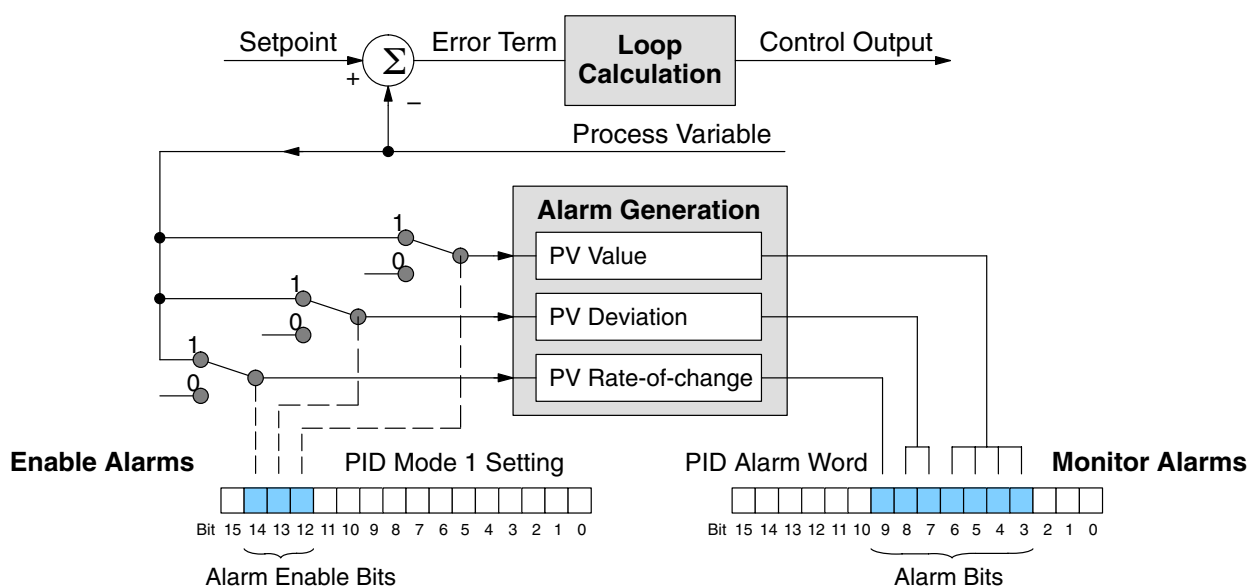
## Process Alarms

The performance of a process control loop may be generally measured by how closely the process variable matches the setpoint. Most process control loops in industry operate continuously, and will eventually lose control of the PV due to an error condition. Process alarms are vital in early discovery of a loop error condition, and can alert plant personnel to manually control a loop or take other measures until the error condition has been repaired.

The DL05 CPU has a sophisticated set of alarm features for each loop:

- **PV Absolute Value Alarms** – monitors the PV with respect to two lower limit values and two upper limit values. It generates alarms whenever the PV goes outside these programmed limits.
- **PV Deviation Alarm** – monitors the PV value as compared to the SP. It alarms when the difference between the PV and SP exceed the programmed alarm value.
- **PV Rate-of-change Alarm** – computes the rate-of-change of the PV, and alarms if it exceeds the programmed alarm amount
- **Alarm Hysteresis** – works in conjunction with the absolute value and deviation alarms to eliminate alarm “chatter” near alarm thresholds.

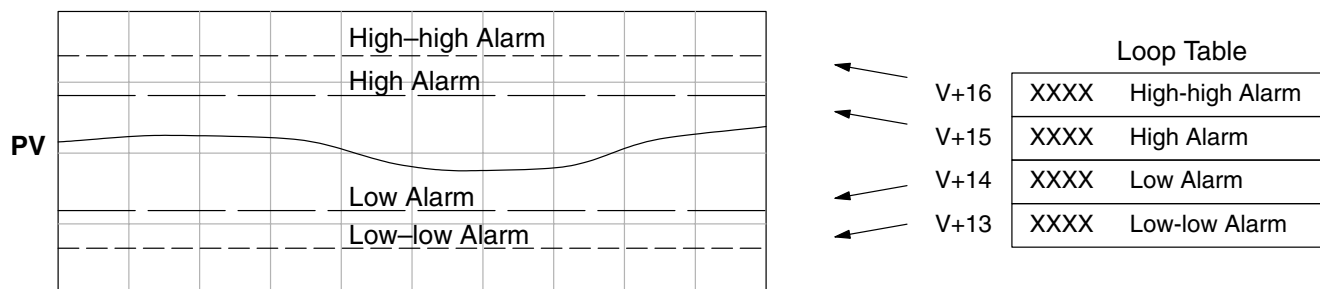
The alarm thresholds are fully programmable, and each type of alarm may be independently enabled and monitored. The following diagram shows the PV monitoring function. Bits 12, 13, and 14 of PID Mode 1 Setting V+00 word in the loop parameter table to enable/disable the alarms. *DirectSOFT32's* PID View setup dialog screens allow easy programming, enabling, and monitoring of the alarms. Ladder logic may monitor the alarm status by examining bits 3 through 9 of PID Mode and alarm Status word V+06 in the loop table.



Unlike the PID calculations, the alarms are always functioning any time the CPU is in Run Mode. The loop may be in Manual, Auto, or Cascade, and the alarms will be functioning if the enable bit(s) as listed above are set =1.

### PV Absolute Value Alarms

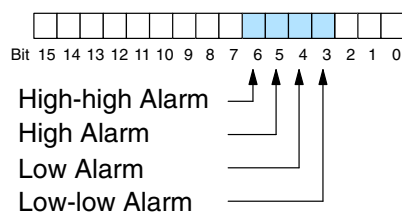
The PV absolute value alarms are organized as two upper and two lower alarms. The alarm status is false as long as the PV value remains in the region between the upper and lower alarms, as shown below. The alarms nearest the safe zone are named *High Alarm* and *Low Alarm*. If the loop loses control, the PV will cross one of these thresholds first. Therefore, you can program the appropriate alarm threshold values in the loop table locations shown below to the right. The data format is the same as the PV and SP (12-bit or 15-bit). The threshold values for these alarms should be set to give an operator an early warning if the process loses control.



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.

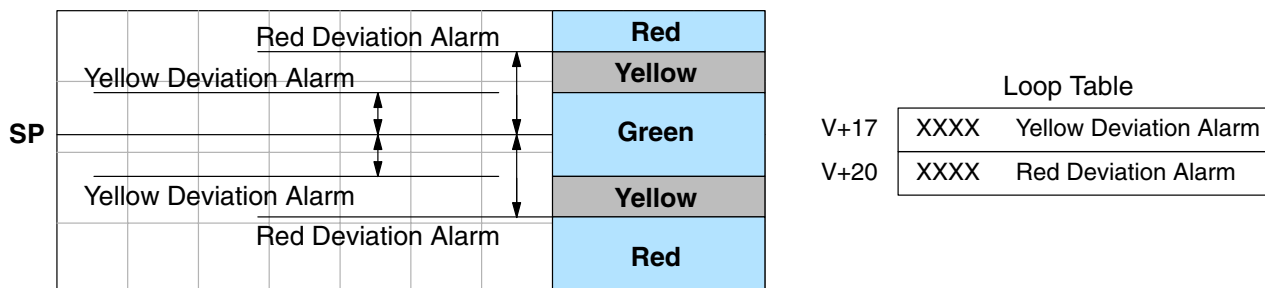
The PV Absolute Value Alarms are reported in the four bits in the PID Mode and Alarm Status word in the loop table, as shown to the right. We highly recommend using ladder logic to monitor these bits. The bit-of-word instructions make this easy to do. Additionally, you can monitor PID alarms using **DirectSOFT**.

PID Mode and Alarm Status V+06



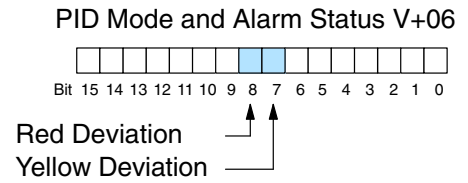
### PV Deviation Alarms

The PV Deviation Alarms monitor the PV deviation with respect to the SP value. The deviation alarm has two programmable thresholds, and each threshold is applied equally above and below the current SP value. In the figure below, the smaller deviation alarm is called the “Yellow Deviation”, indicating a cautionary condition for the loop. The larger deviation alarm is called the “Red Deviation”, indicating a strong error condition for the loop. The threshold values use the loop parameter table locations V+17 and V+20 as shown.



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 **DirectSOFT**.



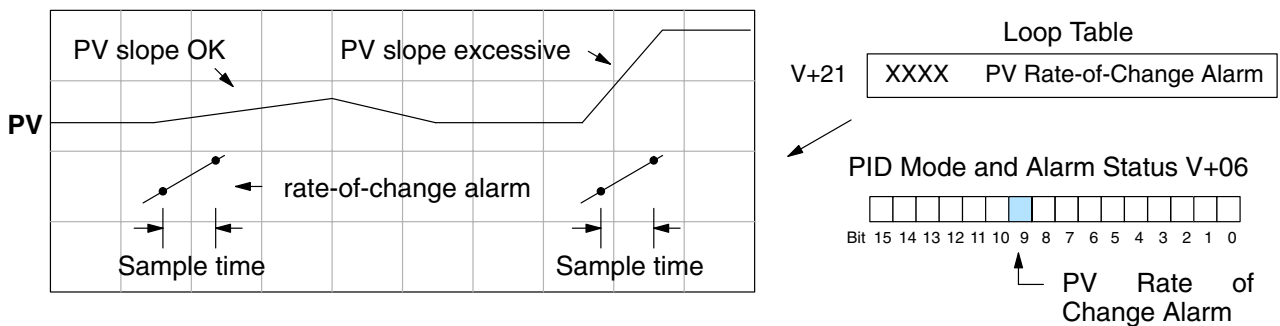
The PV Deviation Alarm can be independently enabled and disabled from the other PV alarms, using bit 13 of the PID Mode 1 Setting V+00 word.

Remember the alarm hysteresis feature works in conjunction with both the deviation and absolute value alarms, and is discussed at the end of this section.

### PV Rate-of-Change Alarm

One powerful way to get an early warning of a process fault is to monitor the *rate-of-change* of the PV. Most batch processes have large masses and slowly-changing PV values. A relatively fast-changing PV will result from a broken signal wire for either the PV or control output, a SP value error, or other causes. If the operator responds to a PV Rate-of-Change Alarm quickly and effectively, the PV absolute value will not reach the point where the material in process would be ruined.

The DL05 loop controller provides a programmable PV Rate-of-Change Alarm, as shown below. The rate-of-change is specified in PV units change per loop sample time. This value is programmed into the loop table location V+21.



As an example, suppose the PV is temperature for our process, and we want an alarm when the temperature changes faster than 15 degrees / minute. We must know PV counts per degree and the loop sample rate. Then, suppose the PV value (in V+03 location) represents 10 counts per degree, and the loop sample rate is 2 seconds. We will use the formula below to convert our engineering units to counts / sample period:

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

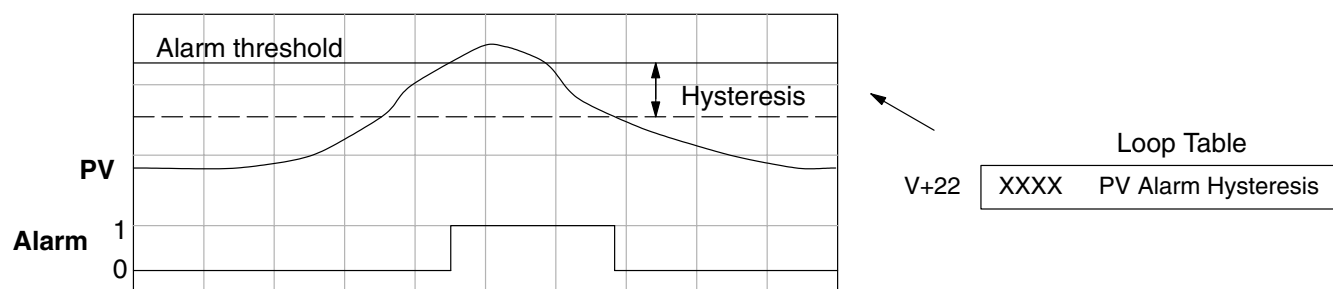
From the calculation result, we would program the value 5 in the loop table for the rate-of-change. The PV Rate-of-Change Alarm can be independently enabled and disabled from the other PV alarms, using bit 14 of the PID Mode 1 Setting V+00 word.

The alarm hysteresis feature (discussed next) does not affect the Rate-of-Change Alarm.

### PV Alarm Hysteresis

The PV Absolute Value Alarm and PV Deviation Alarm are programmed using threshold values. When the absolute value or deviation exceeds the threshold, the alarm status becomes true. Real-world PV signals have some noise on them, which can cause some fluctuation in the PV value in the CPU. As the PV value crosses an alarm threshold, its fluctuations cause the alarm to be intermittent and annoy process operators. The solution is to use the PV Alarm Hysteresis feature.

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



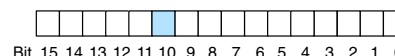
The hysteresis amount is applied *after* the threshold is crossed, and toward the safe zone. In this way, the alarm activates immediately above the programmed threshold value. It delays turning off until the PV value has returned through the threshold by the hysteresis amount.

### Alarm Programing Error

The PV Alarm threshold values must have certain mathematical relationships to be valid. The requirements are listed below. If not met, the Alarm Programming Error bit will be set, as indicated to the right.

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

PID Mode and Alarm Status V+06



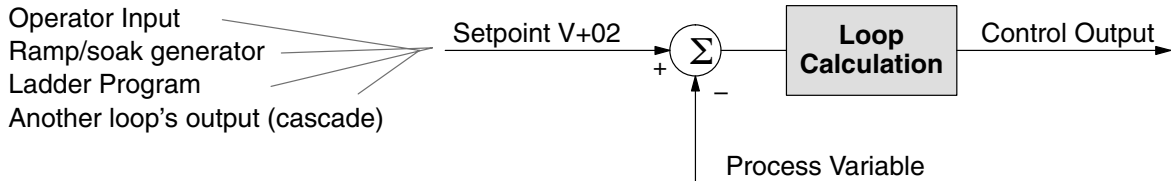
Alarm Programming Error

## Ramp/Soak Generator

### Introduction

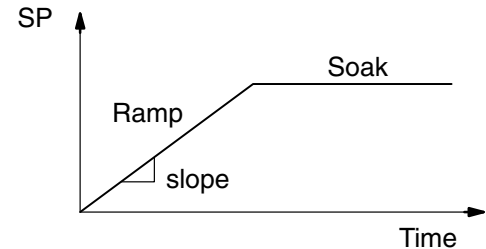
Our discussion of basic loop operation noted the setpoint for a loop will be generated in various ways, depending on the loop operating mode and programming preferences. In the figure below, the ramp / soak generator is one of the ways the SP may be generated. *It is the responsibility of your ladder program to ensure only one source attempts to write the SP value at V+02 at any particular time.*

#### Setpoint Sources:



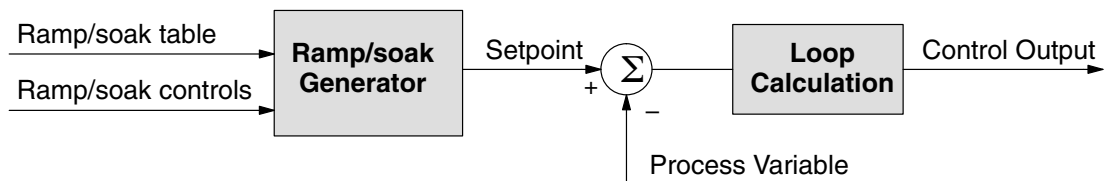
If the SP for your process rarely changes or can tolerate step changes, you probably will not need to use the ramp/soak generator. However, some processes require precisely-controlled SP value changes. The ramp / soak generator can greatly reduce the amount of programming required for these applications.

The terms “ramp” and “soak” have special meanings in the process control industry, and refer to desired setpoint (SP) values in temperature control applications. In the figure to the right, the setpoint increases during the ramp segment. It remains steady at one value during the soak segment.



Complex SP profiles can be generated by specifying a series of ramp/soak segments. The ramp segments are specified in SP units per second time. The soak time is also programmable in minutes.

It is instructive to view the ramp/soak generator as a dedicated function to generate SP values, as shown below. It has two categories of inputs which determine the SP values generated. The *ramp/soak table* must be programmed in advance, containing the values that will define the ramp/soak profile. The loop reads from the table during each PID calculation as necessary. The ramp/soak controls are bits in a special loop table word that control the real-time start/stop functionality of the ramp/soak generator. The ladder program can monitor the status of the ramp soak profile (current ramp/segment number).

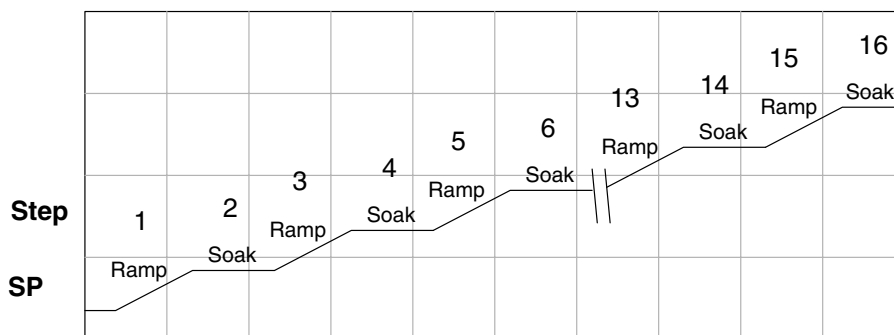




Now that we have described the general ramp/soak generator operation, we list its specific features:

- Each loop has its own ramp/soak generator (use is optional).
- You may specify up to eight ramp/soak steps (16 segments).
- The ramp/soak generator can run anytime the PLC is in Run mode. Its operation is independent of the loop mode (Manual or Auto).
- Ramp/soak real-time controls include Start, Hold, Resume, and Jog.
- Ramp/soak monitoring includes Profile Complete, Soak Deviation (SP minus PV), and current ramp/soak step number.

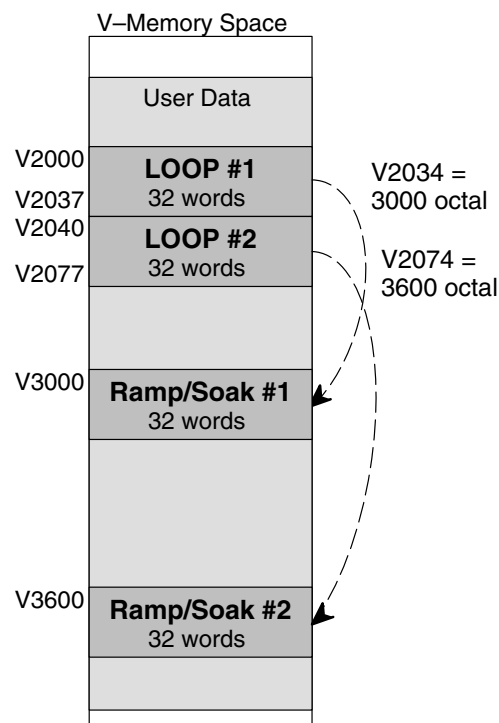
The following figure shows a SP profile consisting of ramp/soak segment pairs. The segments are individually numbered as steps from 1 to 16. The slope of each of the ramp may be either increasing or decreasing. The ramp/soak generator automatically knows whether to increase or decrease the SP based on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



**Ramp/Soak Table**

The parameters which define the ramp/soak profile for a loop are in a ramp/soak table. Each loop may have its own ramp/soak table, but it is optional. Recall the Loop Parameter table consists a 32-word block of memory for each loop, and together they occupy one contiguous memory area. However, the ramp/soak table for a loop is individually located, because it is optional for each loop. An address pointer in location V+34 in the loop table specifies the starting location of the ramp/soak table.

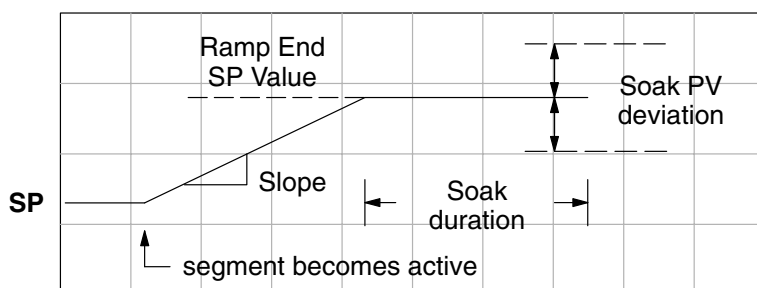
In the example to the right, the loop parameter tables for Loop #1 and #2 occupy contiguous 32-word blocks as shown. Each has a pointer to its ramp/soak table, independently located elsewhere in user V-memory. Of course, you may locate all the tables in one group, as long as they do not overlap.





The parameters in the ramp/soak table must be user-defined. the most convenient way is to use **DirectSOFT**, 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

## Ramp/Soak Table Flags

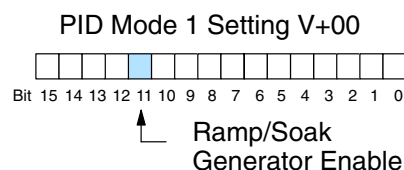
Many applications do not require all 16 R/S steps. Use all zeros in the table for unused steps. The R/S generator ends the profile when it finds ramp slope=0.

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)	

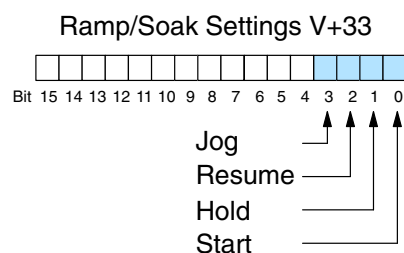
## Ramp/Soak Generator Enable

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



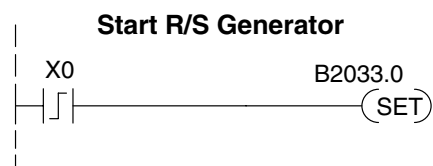
## Ramp/Soak Controls

The four main controls for the ramp/soak generator are in bits 0 to 3 of the ramp/soak settings word in the loop parameter table. **DirectSOFT32** controls these bits directly from the ramp/soak settings dialog. However, you must use ladder logic to control these bits during program execution. We recommend using the bit-of-word instructions.



Ladder logic must set a control bit to a “1” to command the corresponding function. When the loop controller reads the ramp/soak value, it automatically turns off the bit for you. Therefore, a reset of the bit is not required, when the CPU is in Run Mode.

The example program rung to the right shows how an external switch X0 can turn on, and the PD contact uses the leading edge to set the proper control bit to start the ramp soak profile. This uses the Set Bit-of-word instruction.



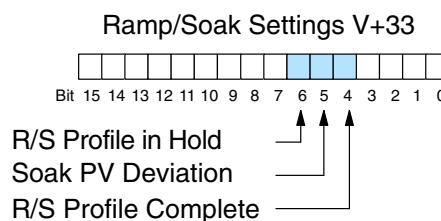
The normal state for the ramp/soak control bits is all zeros. Ladder logic must set only one control bit at a time.

- **Start** – a 0-to-1 transition will start the ramp soak profile. The CPU must be in Run Mode, and the loop can be in Manual or Auto Mode. If the profile is not interrupted by a Hold or Jog command, it finishes normally.
- **Hold** – a 0-to-1 transition will stop the ramp/soak profile in its current state, and the SP value will be frozen.
- **Resume** – a 0-to-1 transition cause the ramp/soak generator to resume operation if it is in the hold state. The SP values will resume from their previous value.
- **Jog** – a 0-to-1 transition will cause the ramp/soak generator to truncate the current segment (step), and go to the next segment.

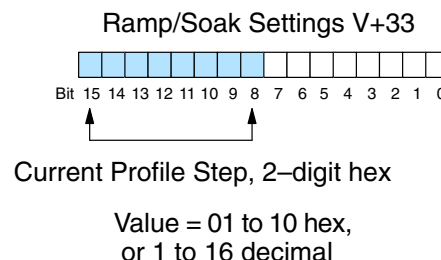
### Ramp/Soak Profile Monitoring

You can monitor the Ramp/Soak profile status using other bits in the Ramp/Soak Settings V+33 word, shown to the right.

- R/S Profile Complete – =1 when the last programmed step is done.
- Soak PV Deviation – =1 when the error (SP–PV) exceeds the specified deviation in the R/S table.
- R/S Profile in Hold – =1 when the profile was active but is now in hold.

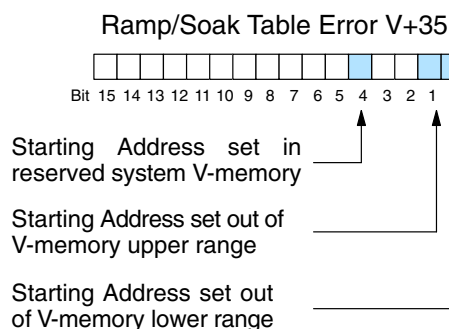


The number of the current step is available in the upper 8 bits of the Ramp/Soak Settings V+33 word. The bits represent a 2-digit hex number, ranging from 1 to 10. Ladder logic can monitor these to synchronize other parts of the program with the ramp/soak profile. Load this word to the accumulator and shift right 8 bits, and you have the step number.



### Ramp/Soak Programming Errors

The starting address for the ramp/soak table must be a valid location. If the address points outside the range of user V-memory, one of the bits to the right will turn on when the ramp/soak generator is started. We recommend using **DirectSOFT32** 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 **DirectSOFT32**'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 V+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 **DirectSOFT**, 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 V+00. It must be set =1.
- The hold bit or other bits in the Ramp/Soak control are on.
- The beginning SP value and the first ramp ending SP value are the same, so first ramp segment has no slope and consequently has no duration. The ramp/soak generator moves quickly to the soak segment, giving the illusion the first ramp is not working.
- The loop is in Cascade Mode, and is trying to get the SP remotely.
- The SP upper limit value in the loop table location V+27 is too low.
- Check your ladder program to verify it is not writing to the SP location (V+02 in the loop table). A quick way to do this is to temporarily place an end coil at the beginning of your program, then go to PLC Run Mode, and manually start the ramp/soak generator.

### Q. The PV value in the table is constant, even though the analog module receives the PV signal.

#### A. Your ladder program must read the analog value from the module successfully and write it into the loop table V+03 location. Verify the analog module is generating the value, and the ladder is working.

### Q. The Derivative gain doesn't seem to have any affect on the output.

#### A. The derivative limit is probably enabled (see section on derivative gain limiting).

**Q. The loop Setpoint appears to be changing by itself.****A.** Check the following for possible causes:

- The Ramp/Soak generator is enabled, and is generating setpoints.
- If this symptom occurs on loop Manual-to-Auto Mode changes, the loop automatically sets the SP=PV (bumpless transfer feature).
- Check your ladder program to verify it is not writing to the SP location (V+02 in the loop table). A quick way to do this is to temporarily place an end coil at the beginning of your program, then go to PLC Run Mode.

**Q. The SP and PV values I enter with *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 I gains I use.****A.** Check the following for possible causes:

- The loop sample time is set too long. Refer to the section near the front of this chapter on selecting the loop update time.
- The gains are too high. Start out by reducing the derivative gain to zero. Then reduce the integral gain, and the proportional gain if necessary.
- There is too much transfer lag in your process. This means the PV reacts sluggishly to control output changes. There may be too much “distance” between actuator and PV sensor, or the actuator may be weak in its ability to transfer energy into the process.
- There may be a process disturbance that is over-powering the loop. Make sure the PV is relatively steady when the SP is not changing.

## Bibliography

Fundamentals of Process Control Theory, Second Edition Author: Paul W. Murrill Publisher: Instrument Society of America ISBN 1-55617-297-4	Application Concepts of Process Control Author: Paul W. Murrill Publisher: Instrument Society of America ISBN 1-55617-080-7
PID Controllers: Theory, Design, and Tuning, 2nd Edition Author: K. Astrom and T Hagglund Publisher: Instrument Society of America ISBN 1-55617-516-7	Fundamentals of Temperature, Pressure, and Flow Measurements, Third edition Author: Robert P. Benedict Publisher: John Wiley and Sons ISBN 0-471-89383-8
Process / Industrial Instruments & Controls Handbook, Fourth Edition Author (Editor-in-Chief): Douglas M. Considine Publisher: McGraw-Hill, Inc. ISBN 0-07-012445-0	pH Measurement and Control, Second Edition Author: Gregory K. McMillan Publisher: Instrument Society of America ISBN 1-55617-483-7
Process Control, Third Edition Instrument Engineer's Handbook Author (Editor-in-Chief): Bela G. Liptak Publisher: Chilton ISBN 0-8019-8242-1	Process Measurement and Analysis, Third Edition Instrument Engineer's Handbook Author (Editor-in-Chief): Bela G. Liptak Publisher: Chilton ISBN 0-8019-8197-2

## Glossary of PID Loop Terminology

<b>Automatic Mode</b>	An operational mode of a loop, in which it makes PID calculations and updates the loop's control output.
<b>Bias Freeze</b>	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.
<b>Bias Term</b>	In the position form of the PID equation, it is the sum of the integrator and the initial control output value.
<b>Bumpless Transfer</b>	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.
<b>Cascaded Loops</b>	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.
<b>Cascade Mode</b>	An operational mode of a loop, in which it receives its SP from another loop's output.
<b>Continuous Control</b>	Control of a process done by delivering a smooth (analog) signal as the control output.
<b>Direct-Acting Loop</b>	A loop in which the PV increases in response to a control output increase. In other words, the process has a positive gain.
<b>Error</b>	The difference in value between the SP and PV, $\text{Error} = \text{SP} - \text{PV}$
<b>Error Deadband</b>	An optional feature which makes the loop insensitive to errors when they are small. You can specify the size of the deadband.
<b>Error Squared</b>	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.
<b>Feedforward</b>	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.
<b>Control Output</b>	The numerical result of a PID equation which is sent by the loop with the intention of nulling out the current error.
<b>Derivative Gain</b>	A constant that determines the magnitude of the PID derivative term in response to the current error.
<b>Integral Gain</b>	A constant that determines the magnitude of the PID integral term in response to the current error.
<b>Major Loop</b>	In cascade control, it is the loop that generates a setpoint for the cascaded loop.
<b>Manual Mode</b>	An operational mode of a loop, in which the PID calculations are stopped. The operator must manually control the loop by writing to the control output value directly.
<b>Minor Loop</b>	In cascade control, the minor loop is the subordinate loop that receives its SP from the major loop.
<b>On / Off Control</b>	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 DL05's continuous loop output to on/off control.
<b>PID Loop</b>	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.
<b>Position Algorithm</b>	The control output is calculated so it responds to the displacement (position) of the PV from the SP (error term)
<b>Process</b>	A manufacturing procedure which adds value to raw materials. Process control particularly refers to inducing <i>chemical</i> changes to the material in process.
<b>Process Variable (PV)</b>	A quantitative measurement of a physical property of the material in process, which affects final product quality and is important to monitor and control.

<b>Proportional Gain</b>	A constant that determines the magnitude of the PID proportional term in response to the current error.
<b>PV Absolute Alarm</b>	A programmable alarm that compares the PV value to alarm threshold values.
<b>PV Deviation Alarm</b>	A programmable alarm that compares the difference between the SP and PV values to a deviation threshold value.
<b>Ramp / Soak Profile</b>	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.
<b>Rate</b>	Also called differentiator, the rate term responds to the <i>changes</i> in the error term.
<b>Remote Setpoint</b>	The location where a loop reads its setpoint when it is configured as the minor loop in a cascaded loop topology.
<b>Reset</b>	Also called integrator, the reset term adds each sampled error to the previous, maintaining a running total called the bias.
<b>Reset Windup</b>	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.
<b>Reverse-Acting Loop</b>	A loop in which the PV increases in response to a control output decrease. In other words, the process has a negative gain.
<b>Sampling time</b>	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.
<b>Setpoint (SP)</b>	The desired value for the process variable. The setpoint (SP) is the input command to the loop controller during closed loop operation.
<b>Soak Deviation</b>	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.
<b>Step Response</b>	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)
<b>Transfer</b>	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.
<b>Velocity Algorithm</b>	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
  - Diagnostics
  - CPU Indicators
  - Communications Problems
  - I/O Point Troubleshooting
  - Noise Troubleshooting
  - Machine Startup and Program Troubleshooting
-



## 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** – Check 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 DL05 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

### Finding Diagnostic Information

The programming devices will notify you of an error if one occurs while online.

- **DirectSOFT** 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.

**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)** 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

**DL05 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	HP Peripheral port time-out
E404	HP Peripheral port time-out
E405	HP Peripheral port time-out
E406	Missing IRT
E412	SBR / LBL >64
E421	Duplicate stage reference
E422	Duplicate SBR/LBL reference
E423	HP Peripheral port time-out
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 DL05 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 (LED off)	1. System voltage incorrect 2. PLC power supply faulty
RUN (LED off)	1. CPU programming error 2. (CPU in program mode)
CPU (LED on)	1. Electrical noise interference 2. Internal CPU defective

### PWR Indicator

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

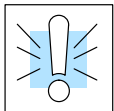
1. Power to the unit is incorrect or is not applied.
2. PLC power supply is faulty.
3. Other component(s) have the power supply shut down.

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 DL05 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 DL05 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 **DirectSOFT™**, 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.

## 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.
- 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 **DirectSOFT** on a personal computer, refer to the **DirectSOFT** 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 DL05 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 DL05 Micro PLCs there are a few facts you should be aware of. These facts 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 $\Omega$  resistor will work. Verify the wattage rating of the resistor is correct for your application.
- Because of the removable terminal blocks on the DL05, 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.

### Testing Output Points

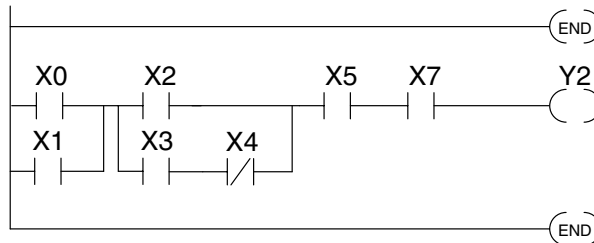
Output points can be set on or off in the DL05 series CPUs. If you want to do an I/O check out independent of the application program, follow the procedure below:

Step	Action
1	Use a handheld programmer or <b>DirectSOFT™</b> 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



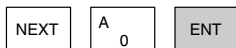
Insert an END statement at the beginning of the program. This disables the remainder of the program.

From a clear display, use the following keystrokes



```
16P STATUS
BIT REF  X
```

Use the PREV or NEXT keys to select the Y data type



```
Y 10 Y 0
□□□□□□□□□□□□□□□□
```

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



Y2 is now on

```
Y 10 Y 0
□□□□□□□□□□□□■□□□
```



## Noise Troubleshooting

### Electrical Noise Problems

Noise is one of the most difficult problems to diagnose. Electrical noise can enter a system in many different ways and 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 a 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

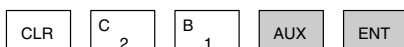
The DL05 Micro PLCs provide several features that can help you debug your program before and during machine startup. This section discusses the following topics which can be very helpful.

- Program Syntax Check
- Duplicate Reference Check
- Special Instructions
- Run Time Edits
- Forcing I/O Points

### Syntax Check

Even though the Handheld Programmer and **DirectSOFT** 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 **DirectSOFT**. 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



```
AUX 21 CHECK PRO
1:SYN 2:DUP REF
```

#### Select syntax check (default selection)



(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
?
```

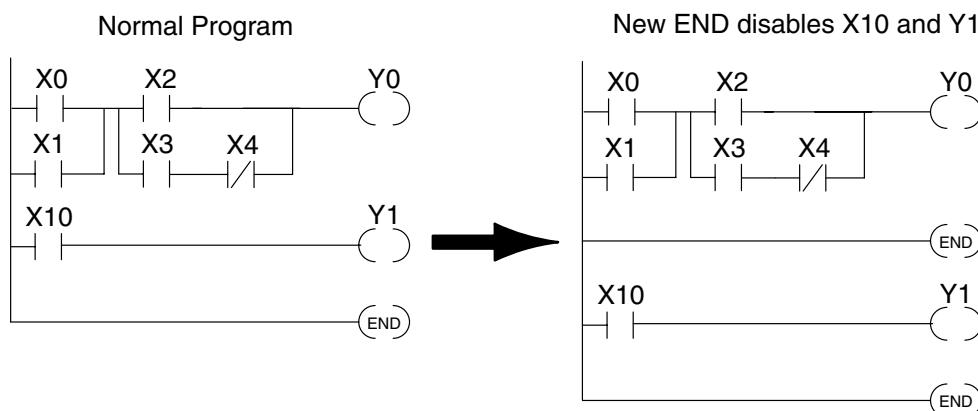
See the Error Codes Section for a complete listing of programming error codes. If you get an error, just press CLR and the Handheld will display the instruction where the error occurred. Correct the problem and continue running the Syntax check until the NO SYNTAX ERROR message appears.

### Special Instructions

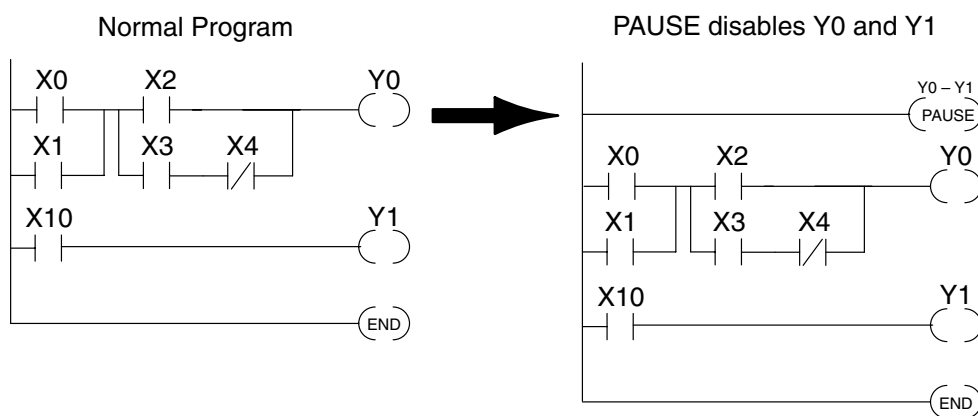
There are several instructions that can be used to help you debug your program during machine startup operations.

- END
- PAUSE
- STOP

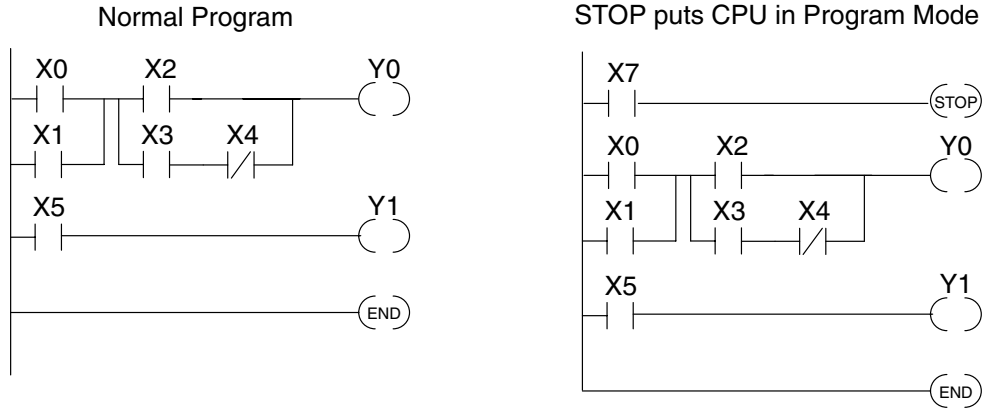
**END Instruction:** If you need a way to quickly disable part of the program, just insert an END statement prior to the portion that should be disabled. When the CPU encounters the END statement, it assumes that is the end of the program. The following diagram shows an example.



**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 X10 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 **DirectSOFT**. 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  
?

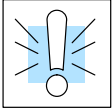
If you get an error, just press CLR and the Handheld will display the instruction where the error occurred. Correct the problem and continue running the Duplicate Reference check until no duplicate references are found.



**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 DL05 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.

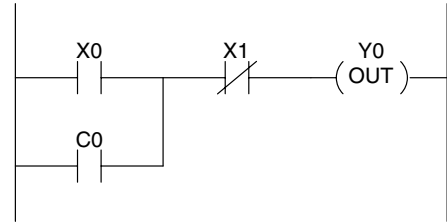
1. If there is a syntax error in the new instruction, the CPU *will not* enter the Run Mode.
2. If you delete an output coil reference and the output was on at the time, the output will remain on until it is forced off with a programming device.
3. Input point changes are not acknowledged during Run 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
AND, ANDN	And, And not
OR, ORN	Or, Or not
STRE, STRNE	Store equal, Store not equal
ANDE, ANDNE	And equal, And not equal
ORE, ORNE	Or equal, Or not equal
STR, STRN	Store greater than or equal Store less than
AND, ANDN	And greater than or equal And less than

Mnemonic	Description
OR, ORN	Or greater than or equal Or less than
LD	Load data (constant)
LDD	Load data double (constant)
ADDD	Add data double (constant)
SUBD	Subtract data double (constant)
MUL	Multiply (constant)
DIV	Divide (constant)
CMPD	Compare accumulator (constant)
ANDD	And accumulator (constant)
ORD	Or accumulator (constant)
XORD	Exclusive or accumulator (constant)
LDF	Load discrete points to accumulator
OUTF	Output accumulator to discrete points
SHFR	Shift accumulator right
SHFL	Shift accumulator left
NCON	Numeric constant

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



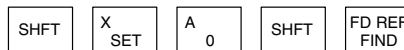
\*MODE CHANGE\*  
RUN TIME EDIT?

### 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.)

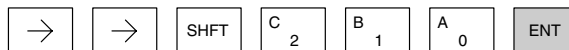
\*MODE CHANGE\*  
RUNTIME EDITS

### Find the instruction you want to change (X0)



\$00000 STR X0

### Press the arrow key to move to the X. Then enter the new contact (C10).



RUNTIME EDIT?  
STR C10

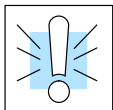
### Press ENT to confirm the change

**ENT** (Note, once you press ENT, the next address is displayed.)

OR C0

**Forcing I/O Points**

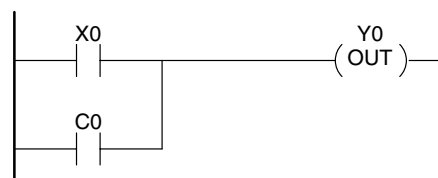
There are many times, especially during machine startup and troubleshooting, that you need the capability to force an I/O point to be either on or off. Before you use a programming device to force any data type it is important you understand how the DL05 CPUs process the forcing requests.



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

**Bit Forcing** — Bit forcing temporarily changes the status of a discrete bit. For example, you may want to force an input on even though the program has turned it off. This allows you to change the point status stored in the image register. The forced value will be valid until the CPU writes to the image register location during the next scan. This is useful you just need to force a bit on to trigger another event.

The following diagrams show a brief example of how you could use the D2-HPP Handheld Programmer to force an I/O point. The example assumes you have already placed the CPU into Run Mode.

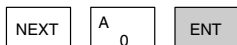


From a clear display, use the following keystrokes



```
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

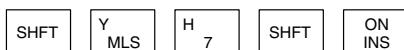


Y2 is now on

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

**Bit Forcing with Direct Access**

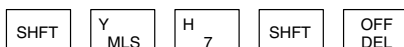
From a blank display, use the following keystrokes to force Y7 ON



Solid fill indicates point is on.

```
BIT FORCE
Y7
```

From a blank display, use the following keystrokes to force Y7 OFF



No fill indicates point is off.

```
BIT FORCE
Y7
```

# Auxiliary Functions

---

In This Appendix. . . .

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



## Introduction

### 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 **DirectSOFT™** 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 **DirectSOFT** 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 DL05 PLCs.

AUX Function and Description		DL05
<b>AUX 2* — RLL Operations</b>		
21	Check Program	○
22	Change Reference	○
23	Clear Ladder Range	○
24	Clear All Ladders	○
<b>AUX 3* — V-Memory Operations</b>		
31	Clear V Memory	○
<b>AUX 4* — I/O Configuration</b>		
41	Show I/O Configuration	○
<b>AUX 5* — CPU Configuration</b>		
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	○

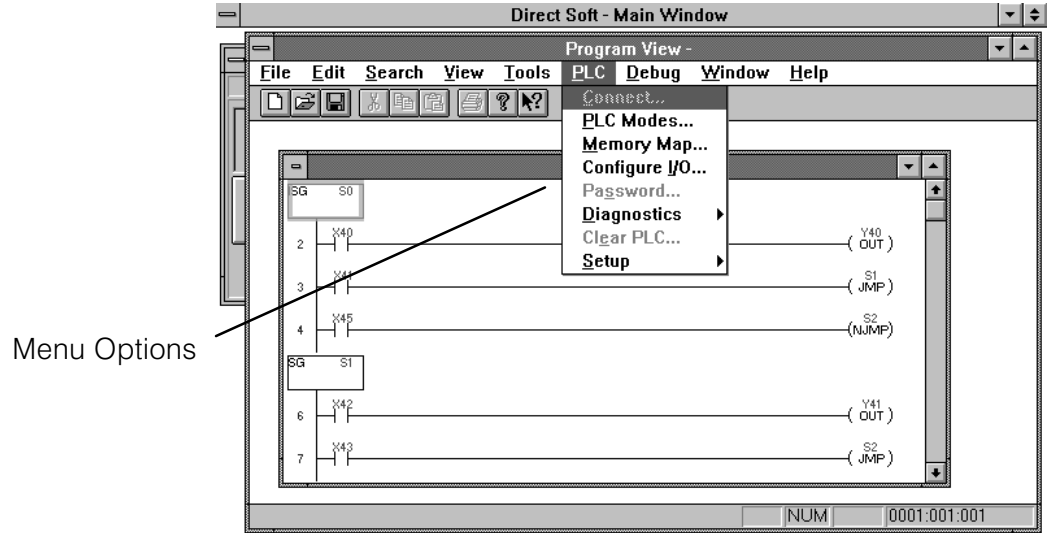
○ — supported

HP — Handheld Programmer function

AUX Function and Description		DL05
<b>AUX 6* — Handheld Programmer Configuration</b>		
61	Show Revision Numbers	○
62	Beeper On / Off	HP
65	Run Self Diagnostics	HP
<b>AUX 7* — EEPROM Operations</b>		
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
<b>AUX 8* — Password Operations</b>		
81	Modify Password	○
82	Unlock CPU	○
83	Lock CPU	○

## Accessing AUX Functions via DirectSOFT

**DirectSOFT™** provides various menu options during both online and offline programming. Some of the AUX functions are only available during online programming, some only during offline programming, and some during both online and offline programming. The following diagram shows an example of the PLC operations menu available within **DirectSOFT**.



## 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.



AUX FUNCTION SELECTION  
AUX 2\* RLL OPERATIONS

### Use NXT or PREV to cycle through the menus



AUX FUNCTION SELECTION  
AUX 3\* V OPERATIONS

### Press ENT to select sub-menus



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



AUX 3\* V OPERATIONS  
AUX 31 CLR V MEMORY

## AUX 2\* — RLL Operations

### AUX 21 Check Program

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

Both the Handheld and **DirectSOFT™** 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 **DirectSOFT**.

### 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. **DirectSOFT** 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 **DirectSOFT**.

## 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 **DirectSOFT**.

## 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 DL05. Both the Handheld Programmer and **DirectSOFT™** 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

DL05 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 **DirectSOFT™** 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 **DirectSOFT** 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 **DirectSOFT™** by using the PLC/Setup sub-menu.

### AUX 55 Set Watchdog Timer

DL05 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 **DirectSOFT** by using the PLC/Setup sub-menu.

### AUX 56 CPU Network Address

Since the DL05 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, **DirectSOFT**, or, as a communication port for **DirectNET** and MODBUS. Refer to **DirectNET** and MODBUS manuals for additional information about communication settings required for network operation.



### AUX 57 Set Retentive Ranges

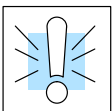
**NOTE:** You will only need to use this procedure if you have port 2 connected to a network. Otherwise, the default settings will work fine.

Use AUX 56 to set the network address and communication parameters. You can also perform this operation from within **DirectSOFT** by using the PLC/Setup sub-menu.

DL05 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	DL05	
	Default Range	Available Range
Control Relays	C400 – C777	C0 – C777
V Memory	V1400 – V7777	V0 – V7777
Timers	None by default	T0 – T177
Counters	CT0 – CT177	CT0 – CT177
Stages	None by default	S0 – S377

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



**WARNING:** The DL05 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

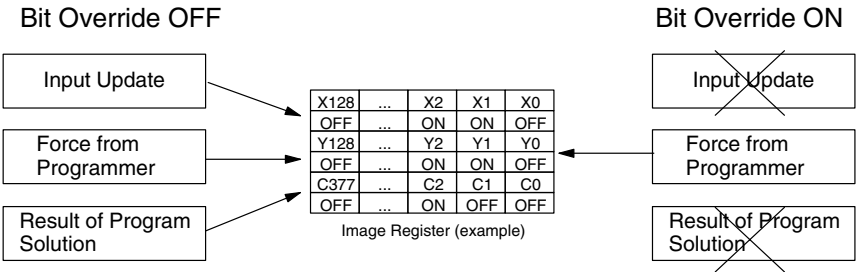
AUX 58 is used to override the output disable function of the Pause instruction. Use AUX 58 to program a single output or a range of outputs which will operate normally even when those points are within the scope of the pause instruction.

### 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 **DirectSOFT™**. 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 DL05 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.

## AUX 6\* — Handheld Programmer Configuration

### AUX 61 Show Revision Numbers

The following auxiliary functions allow you to setup, view, or change the Handheld Programmer configuration.

As with most industrial control products, there are cases when additional features and enhancements are made. Sometimes these new features only work with certain releases of firmware. By using AUX 61 you can quickly view the CPU and Handheld Programmer firmware revision numbers. This information (for the CPU) is also available from within **Direct**SOFT from the PLC/Diagnostics sub-menu.

### AUX 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

### Transferrable Memory Areas

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

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	DL05 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 <i>non-volatile</i> V-memory 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 DL05. 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

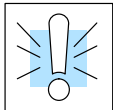
**AUX 81  
Modify Password**

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.



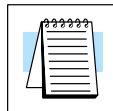


You can also enter or modify a password from within **DirectSOFT™** by using the PLC/Password sub-menu. This feature works slightly differently in **DirectSOFT**. Once you've entered a password, the CPU is automatically locked when you exit the software package. It will also be locked if the CPU is power 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 DL05 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. **DirectSOFT** 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 **DirectSOFT** since the CPU is automatically locked whenever you exit the software package.

# DL05 Error Codes

---

In This Appendix. . . .  
— Error Code Table

---

DL05 Error Code	Description
<b>E003</b> 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.
<b>E004</b> INVALID INSTRUCTION	The CPU attempted to execute an instruction code, but the RAM contents had a parity error. Performing a program download to the CPU in an electrically noisy environment can corrupt a program's contents. Clear the CPU program memory, and download the program again.
<b>E043</b> MC BATTERY LOW	The battery in the CMOS RAM cartridge is low and should be replaced.
<b>E104</b> WRITE FAILED	A write to the CPU was not successful. Disconnect the power, remove the CPU, and make sure the EEPROM is not write protected. If the EEPROM is not write protected, make sure the EEPROM is installed correctly. If both conditions are OK, replace the CPU.
<b>E151</b> 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 PLC.
<b>E311</b> HP COMM ERROR 1	A request from the handheld programmer could not be processed by the CPU. Clear the error and retry the request. If the error continues replace the CPU. SP46 will be on and the error code will be stored in V7756.
<b>E312</b> HP COMM ERROR 2	A data error was encountered during communications with the CPU. Clear the error and retry the request. If the error continues check the cabling between the two devices, replace the handheld programmer, then if necessary replace the CPU. The error code will be stored in V7756.
<b>E313</b> HP COMM ERROR 3	An address error was encountered during communications with the CPU. Clear the error and retry the request. If the error continues check the cabling between the two devices, replace the handheld programmer, then if necessary replace the CPU. The error code will be stored in V7756.
<b>E316</b> HP COMM ERROR 6	A mode error was encountered during communications with the CPU. Clear the error and retry the request. If the error continues replace the handheld programmer, then if necessary replace the CPU. The error code will be stored in V7756.
<b>E320</b> HP COMM TIME-OUT	The CPU did not respond to the handheld programmer communication request. Check to insure cabling is correct and not defective. Power cycle the system if the error continues replace the CPU first and then the handheld programmer if necessary.
<b>E321</b> COMM ERROR	A data error was encountered during communication with the CPU. Check to insure cabling is correct and not defective. Power cycle the system and if the error continues replace the CPU first and then the handheld programmer if necessary.

DL05 Error Code	Description
<b>E360</b> HP PERIPHERAL PORT TIME-OUT	The device connected to the peripheral port did not respond to the handheld programmer communication request. Check to insure cabling is correct and not defective. The peripheral device or handheld programmer could be defective.
<b>E4**</b> 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.
<b>E401</b> 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.
<b>E402</b> MISSING LBL	A MOVMC or LDLBL instruction was used without the appropriate label. Refer to the Chapter 5 for details on these instructions. SP52 will be on and the error code will be stored in V7755.
<b>E403</b> 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.
<b>E404</b> 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.
<b>E405</b> 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.
<b>E406</b> 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.
<b>E412</b> SBR/LBL>64	There is greater than 64 SBR or DLBL instructions in the program. This error is also returned if there is greater than 2 INT instructions used in the program. SP52 will be on and the error code will be stored in V7755.
<b>E421</b> 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.
<b>E422</b> 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.
<b>E423</b> 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.
<b>E431</b> 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.

DL05 Error Code	Description
<b>E433</b> 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.
<b>E434</b> 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.
<b>E435</b> 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.
<b>E436</b> 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.
<b>E437</b> 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.
<b>E438</b> 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.
<b>E440</b> 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).
<b>E441</b> 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.
<b>E451</b> BAD MLS/MLR	MLS instructions must be numbered in ascending order from top to bottom.
<b>E453</b> MISSING T/C	A timer or counter contact is being used where the associated timer or counter does not exist.
<b>E454</b> BAD TMRA	One of the contacts is missing from a TMRA instruction.
<b>E455</b> BAD CNT	One of the contacts is missing from a CNT or UDC instruction.
<b>E456</b> BAD SR	One of the contacts is missing from the SR instruction.

DL05 Error Code	Description
<b>E461</b> STACK OVERFLOW	More than nine levels of logic have been stored on the stack. Check the use of OR STR and AND STR instructions.
<b>E462</b> 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.
<b>E463</b> LOGIC ERROR	A STR instruction was not used to begin a rung of ladder logic.
<b>E464</b> MISSING CKT	A rung of ladder logic is not terminated properly.
<b>E471</b> DUPLICATE COIL REFERENCE	Two or more OUT instructions reference the same I/O point.
<b>E472</b> DUPLICATE TMR REFERENCE	Two or more TMR instructions reference the same number.
<b>E473</b> DUPLICATE CNT REFERENCE	Two or more CNT instructions reference the same number.
<b>E499</b> PRINT INSTRUCTION	Invalid PRINT instruct usage. Quotations and/or spaces were not entered or entered incorrectly.

DL05 Error Code	Description
<b>E501</b> BAD ENTRY	An invalid keystroke or series of keystrokes was entered into the handheld programmer.
<b>E502</b> BAD ADDRESS	An invalid or out of range address was entered into the handheld programmer.
<b>E503</b> BAD COMMAND	An invalid command was entered in the handheld programmer.
<b>E504</b> BAD REF/VAL	An invalid value or reference number was entered with an instruction.
<b>E505</b> INVALID INSTRUCTION	An invalid instruction was entered into the handheld programmer.
<b>E506</b> INVALID OPERATION	An invalid operation was attempted by the handheld programmer.
<b>E520</b> BAD OP-RUN	An operation which is invalid in the RUN mode was attempted by the handheld programmer.
<b>E521</b> BAD OP-TRUN	An operation which is invalid in the TEST RUN mode was attempted by the handheld programmer.
<b>E523</b> BAD OP-TPGM	An operation which is invalid in the TEST PROGRAM mode was attempted by the handheld programmer.
<b>E524</b> BAD OP-PGM	An operation which is invalid in the PROGRAM mode was attempted by the handheld programmer.
<b>E525</b> MODE SWITCH	An operation was attempted by the handheld programmer while the CPU mode switch was in a position other than the TERM position.
<b>E526</b> OFF LINE	The handheld programmer is in the OFFLINE mode. To change to the ONLINE mode use the MODE the key.
<b>E527</b> ON LINE	The handheld programmer is in the ON LINE mode. To change to the OFF LINE mode use the MODE the key.
<b>E528</b> CPU MODE	The operation attempted is not allowed during a Run Time Edit.
<b>E540</b> CPU LOCKED	The CPU has been password locked. To unlock the CPU use AUX82 with the password.
<b>E541</b> WRONG PASSWORD	The password used to unlock the CPU with AUX82 was incorrect.
<b>E542</b> PASSWORD RESET	The CPU powered up with an invalid password and reset the password to 00000000. A password may be re-entered using AUX81.
<b>E601</b> MEMORY FULL	Attempted to enter an instruction which required more memory than is available in the CPU.
<b>E602</b> INSTRUCTION MISSING	A search function was performed and the instruction was not found.

DL05 Error Code	Description
<b>E603</b> DATA MISSING	A search function was performed and the data was not found.
<b>E604</b> REFERENCE MISSING	A search function was performed and the reference was not found.
<b>E620</b> OUT OF MEMORY	An attempt to transfer more data between the CPU and handheld programmer than the receiving device can hold.
<b>E621</b> 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.
<b>E622</b> NO HPP EEPROM	A data transfer was attempted with no EEPROM (or possibly a faulty EEPROM) installed in the handheld programmer.
<b>E623</b> SYSTEM EEPROM	A function was requested with an EEPROM in the handheld programmer which contains system information only.
<b>E624</b> V-MEMORY ONLY	A function was requested with an EEPROM in the handheld programmer which contains V-memory data only.
<b>E625</b> PROGRAM ONLY	A function was requested with an EEPROM in the handheld programmer which contains program data only.
<b>E626</b> PROM MC	An attempt to transfer data from a tape to a UVPROM Memory Cartridge. This transfer must be made using a CMOS RAM Cartridge.
<b>E627</b> BAD WRITE	An attempt to write to a write-protected or faulty EEPROM in the handheld programmer was made. Check the write protect jumper and replace the EEPROM if necessary.
<b>E628</b> EEPROM TYPE ERROR	The wrong size EEPROM is being used in the handheld programmer. This error occurs when the program size is larger than what the HPP can hold.
<b>E640</b> COMPARE ERROR	A compare between the EEPROM handheld programmer and the CPU was found to be in error.
<b>E641</b> VOLUME LEVEL	The volume level of the cassette player is not set properly. Adjust the volume and retry the operation.
<b>E642</b> CHECKSUM ERROR	An error was detected while data was being transferred to the handheld programmer's Memory Cartridge. Check cabling and retry the operation.
<b>E650</b> 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.
<b>E651</b> 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.
<b>E652</b> 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
- Instruction Execution Times

## Introduction

This appendix contains several tables that provide the instruction execution times for DL05 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	DL05
Timer Current Values	V0 - V177
Counter Current Values	V1000 - V1177
User Data Words	V1200 - V7377 V7400 - V7577

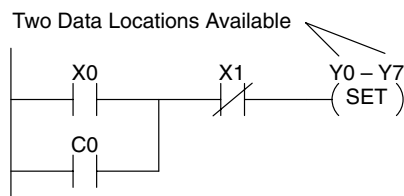
### V-Memory Bit Registers

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	DL05
Input Points (X)	V40400 - V40417
Output Points (Y)	V40500 - V40517
Control Relays (C)	V40600 - V40637
Stages (S)	V41000 - V41017
Timer Status Bits	V41100 - V41107
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 that 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 2nd #: X, Y, C, S (N pt)	32.2 $\mu$ s $14\mu\text{s} + 3.1\mu\text{s} \times \text{N}$
RST	1st #: X, Y, C, S 2nd #: X, Y, C, S (N pt)	34.4 $\mu$ s $16 + 3.2 \times \text{N}$

Execution depends on numbers of locations and types of data used

# Instruction Execution Times

## Boolean Instructions

Boolean Instructions		DL05	
Instruction	Legal Data Types	Execute	Not Execute
STR	X, Y, C, T, CT, S, SP	2.0 $\mu$ s	2.0 $\mu$ s
STRN	X, Y, C, T, CT, S, SP	2.3 $\mu$ s	2.3 $\mu$ s
OR	X, Y, C, T, CT, S, SP	1.6 $\mu$ s	1.6 $\mu$ s
ORN	X, Y, C, T, CT, S, SP	1.9 $\mu$ s	1.9 $\mu$ s
AND	X, Y, C, T, CT, S, SP	1.4 $\mu$ s	1.4 $\mu$ s
ANDN	X, Y, C, T, CT, S, SP	1.6 $\mu$ s	1.6 $\mu$ s
ANDSTR	None	1.3 $\mu$ s	1.3 $\mu$ s
ORSTR	None	1.3 $\mu$ s	1.3 $\mu$ s
OUT	X, Y, C	6.8 $\mu$ s	6.8 $\mu$ s
OROUT	X, Y, C	6.7 $\mu$ s	6.7 $\mu$ s
NOT	None	1.6 $\mu$ s	1.6 $\mu$ s
PD	X, Y, C	52.3 $\mu$ s	53.0 $\mu$ s
STRPD	X, Y, C, T, CT, S, SP	20.2 $\mu$ s	12.9 $\mu$ s
STRND	X, Y, C, T, CT, S, SP	20.1 $\mu$ s	13.0 $\mu$ s
ORPD	X, Y, C, T, CT, S, SP	20.0 $\mu$ s	12.6 $\mu$ s
ORND	X, Y, C, T, CT, S, SP	19.8 $\mu$ s	12.7 $\mu$ s
ANDPD	X, Y, C, T, CT, S, SP	20.0 $\mu$ s	12.6 $\mu$ s
ANDND	X, Y, C, T, CT, S, SP	19.9 $\mu$ s	12.8 $\mu$ s
SET	1st #: X, Y, C, S	32.2 $\mu$ s	3.7 $\mu$ s
	2nd #: X, Y, C, S (N pt)	14 $\mu$ s+3.1 $\mu$ sxN	4.7 $\mu$ s
RST	1st #: X, Y, C, S	34.4 $\mu$ s	3.7 $\mu$ s
	2nd #: X, Y, C, S (N pt)	16+3.2xN	4.7 $\mu$ s
	1st #: T, CT	63.6 $\mu$ s	3.7 $\mu$ s
	2nd #: T, CT (N pt)	39+6.7xN	4.9 $\mu$ s
PAUSE	1wd: Y	23.4 $\mu$ s	23.0 $\mu$ s
	2wd: Y (N points)	19.7+1.5xN	19.5+1.4xN

Comparative  
Boolean  
Instructions

Comparative Boolean Instructions			DL05	
Instruction	Legal Data Types		Execute	Not Execute
STRE	1st	2nd		
		V: Data Reg.		
	V: Data Reg.	V:Data Reg.	17.0 $\mu$ s	16.8 $\mu$ s
		V:Bit Reg.	17.0 $\mu$ s	16.8 $\mu$ s
		K:Constant	11.7 $\mu$ s	11.6 $\mu$ s
		P:Indir. (Data)	42.8 $\mu$ s	42.7 $\mu$ s
		P:Indir. (Bit)	42.8 $\mu$ s	42.7 $\mu$ s
	V: Bit Reg.	V:Data Reg.	17.0 $\mu$ s	16.8 $\mu$ s
		V:Bit Reg.	17.0 $\mu$ s	16.8 $\mu$ s
		K:Constant	11.7 $\mu$ s	11.6 $\mu$ s
		P:Indir. (Data)	42.8 $\mu$ s	42.7 $\mu$ s
		P:Indir. (Bit)	42.8 $\mu$ s	42.7 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	42.8 $\mu$ s	42.7 $\mu$ s
		V:Bit Reg.	42.8 $\mu$ s	42.7 $\mu$ s
		K:Constant	38.1 $\mu$ s	38.0 $\mu$ s
		P:Indir. (Data)	66.8 $\mu$ s	66.7 $\mu$ s
		P:Indir. (Bit)	66.8 $\mu$ s	66.7 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	42.8 $\mu$ s	42.7 $\mu$ s
		V:Bit Reg.	42.8 $\mu$ s	42.7 $\mu$ s
		K:Constant	38.1 $\mu$ s	38.0 $\mu$ s
		P:Indir. (Data)	66.8 $\mu$ s	66.7 $\mu$ s
		P:Indir. (Bit)	66.8 $\mu$ s	66.7 $\mu$ s
STRNE	1st	2nd		
		V: Data Reg.		
	V: Data Reg.	V:Data Reg.	17.1 $\mu$ s	17.3 $\mu$ s
		V:Bit Reg.	17.1 $\mu$ s	17.3 $\mu$ s
		K:Constant	11.8 $\mu$ s	12.0 $\mu$ s
		P:Indir. (Data)	43.0 $\mu$ s	43.1 $\mu$ s
		P:Indir. (Bit)	43.0 $\mu$ s	43.1 $\mu$ s
	V: Bit Reg.	V:Data Reg.	17.1 $\mu$ s	17.3 $\mu$ s
		V:Bit Reg.	17.1 $\mu$ s	17.3 $\mu$ s
		K:Constant	11.8 $\mu$ s	12.0 $\mu$ s
		P:Indir. (Data)	43.0 $\mu$ s	43.1 $\mu$ s
		P:Indir. (Bit)	43.0 $\mu$ s	43.1 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	43.0 $\mu$ s	43.1 $\mu$ s
		V:Bit Reg.	43.0 $\mu$ s	43.1 $\mu$ s
		K:Constant	38.2 $\mu$ s	38.4 $\mu$ s
		P:Indir. (Data)	67.0 $\mu$ s	67.1 $\mu$ s
		P:Indir. (Bit)	67.0 $\mu$ s	67.1 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	43.0 $\mu$ s	43.1 $\mu$ s
		V:Bit Reg.	43.0 $\mu$ s	43.1 $\mu$ s
		K:Constant	38.2 $\mu$ s	38.4 $\mu$ s
		P:Indir. (Data)	67.0 $\mu$ s	67.1 $\mu$ s
		P:Indir. (Bit)	67.0 $\mu$ s	67.1 $\mu$ s

Comparative Boolean (cont.)			DL05	
Instruct	Legal Data Types		Execute	Not Execute
ORE	1st	2nd		
	V: Data Reg.	V:Data Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		V:Bit Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		K:Constant	11.5 $\mu$ s	11.4 $\mu$ s
		P:Indir. (Data)	42.6 $\mu$ s	42.5 $\mu$ s
		P:Indir. (Bit)	42.6 $\mu$ s	42.5 $\mu$ s
	V: Bit Reg.	V:Data Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		V:Bit Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		K:Constant	11.5 $\mu$ s	11.4 $\mu$ s
		P:Indir. (Data)	42.6 $\mu$ s	42.5 $\mu$ s
		P:Indir. (Bit)	42.6 $\mu$ s	42.5 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	37.7 $\mu$ s	37.6 $\mu$ s
		V:Bit Reg.	37.7 $\mu$ s	37.6 $\mu$ s
		K:Constant	42.6 $\mu$ s	42.5 $\mu$ s
		P:Indir. (Data)	66.5 $\mu$ s	66.4 $\mu$ s
		P:Indir. (Bit)	66.5 $\mu$ s	66.4 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	37.7 $\mu$ s	37.6 $\mu$ s
		V:Bit Reg.	37.7 $\mu$ s	37.6 $\mu$ s
		K:Constant	42.6 $\mu$ s	42.5 $\mu$ s
		P:Indir. (Data)	66.5 $\mu$ s	66.4 $\mu$ s
		P:Indir. (Bit)	66.5 $\mu$ s	66.4 $\mu$ s
ORNE	1st	2nd		
	V: Data Reg.	V:Data Reg.	16.7 $\mu$ s	16.8 $\mu$ s
		V:Bit Reg.	16.7 $\mu$ s	16.8 $\mu$ s
		K:Constant	11.6 $\mu$ s	11.7 $\mu$ s
		P:Indir. (Data)	42.7 $\mu$ s	42.9 $\mu$ s
		P:Indir. (Bit)	42.7 $\mu$ s	42.9 $\mu$ s
	V: Bit Reg.	V:Data Reg.	16.7 $\mu$ s	16.8 $\mu$ s
		V:Bit Reg.	16.7 $\mu$ s	16.8 $\mu$ s
		K:Constant	11.6 $\mu$ s	11.7 $\mu$ s
		P:Indir. (Data)	42.7 $\mu$ s	42.9 $\mu$ s
		P:Indir. (Bit)	42.7 $\mu$ s	42.9 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	42.7 $\mu$ s	42.8 $\mu$ s
		V:Bit Reg.	42.7 $\mu$ s	42.8 $\mu$ s
		K:Constant	37.8 $\mu$ s	38.0 $\mu$ s
		P:Indir. (Data)	66.6 $\mu$ s	66.7 $\mu$ s
		P:Indir. (Bit)	66.6 $\mu$ s	66.7 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	42.7 $\mu$ s	42.8 $\mu$ s
		V:Bit Reg.	42.7 $\mu$ s	42.8 $\mu$ s
		K:Constant	37.8 $\mu$ s	38.0 $\mu$ s
		P:Indir. (Data)	66.6 $\mu$ s	66.7 $\mu$ s
		P:Indir. (Bit)	66.6 $\mu$ s	66.7 $\mu$ s

Comparative Boolean (cont.)			DL05	
Instruct	Legal Data Types		Execute	Not Execute
ANDE	1st	2nd		
	V: Data Reg.	V:Data Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		V:Bit Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		K:Constant	11.5 $\mu$ s	11.4 $\mu$ s
		P:Indir. (Data)	42.6 $\mu$ s	42.5 $\mu$ s
		P:Indir. (Bit)	42.6 $\mu$ s	42.5 $\mu$ s
	V: Bit Reg.	V:Data Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		V:Bit Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		K:Constant	11.5 $\mu$ s	11.4 $\mu$ s
		P:Indir. (Data)	42.6 $\mu$ s	42.5 $\mu$ s
		P:Indir. (Bit)	42.6 $\mu$ s	42.5 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	42.6 $\mu$ s	42.5 $\mu$ s
		V:Bit Reg.	42.6 $\mu$ s	42.5 $\mu$ s
		K:Constant	37.7 $\mu$ s	37.6 $\mu$ s
		P:Indir. (Data)	66.5 $\mu$ s	66.3 $\mu$ s
		P:Indir. (Bit)	66.5 $\mu$ s	66.3 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	42.6 $\mu$ s	42.5 $\mu$ s
		V:Bit Reg.	42.6 $\mu$ s	42.5 $\mu$ s
		K:Constant	37.7 $\mu$ s	37.6 $\mu$ s
		P:Indir. (Data)	66.5 $\mu$ s	66.3 $\mu$ s
		P:Indir. (Bit)	66.5 $\mu$ s	66.3 $\mu$ s
ANDNE	1st	2nd		
	V: Data Reg.	V:Data Reg.	16.7 $\mu$ s	16.8 $\mu$ s
		V:Bit Reg.	16.7 $\mu$ s	16.8 $\mu$ s
		K:Constant	11.6 $\mu$ s	11.7 $\mu$ s
		P:Indir. (Data)	42.7 $\mu$ s	42.9 $\mu$ s
		P:Indir. (Bit)	42.7 $\mu$ s	42.9 $\mu$ s
	V: Bit Reg.	V:Data Reg.	16.7 $\mu$ s	16.8 $\mu$ s
		V:Bit Reg.	16.7 $\mu$ s	16.8 $\mu$ s
		K:Constant	11.6 $\mu$ s	11.7 $\mu$ s
		P:Indir. (Data)	42.7 $\mu$ s	42.9 $\mu$ s
		P:Indir. (Bit)	42.7 $\mu$ s	42.9 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	42.7 $\mu$ s	42.9 $\mu$ s
		V:Bit Reg.	42.7 $\mu$ s	42.9 $\mu$ s
		K:Constant	37.9 $\mu$ s	38.1 $\mu$ s
		P:Indir. (Data)	66.6 $\mu$ s	66.8 $\mu$ s
		P:Indir. (Bit)	66.6 $\mu$ s	66.8 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	42.7 $\mu$ s	42.9 $\mu$ s
		V:Bit Reg.	42.7 $\mu$ s	42.9 $\mu$ s
		K:Constant	37.9 $\mu$ s	38.1 $\mu$ s
		P:Indir. (Data)	66.6 $\mu$ s	66.8 $\mu$ s
		P:Indir. (Bit)	66.6 $\mu$ s	66.8 $\mu$ s

Comparative Boolean (cont.)			DL05	
Instruc	Legal Data Types		Execute	Not Execute
STR	1st T, CT	2nd V:Data Reg.	17.0 $\mu$ s	16.9 $\mu$ s
		V:Bit Reg.	17.0 $\mu$ s	16.9 $\mu$ s
		K:Constant	11.7 $\mu$ s	11.6 $\mu$ s
		P:Indir. (Data)	42.8 $\mu$ s	42.7 $\mu$ s
		P:Indir. (Bit)	42.8 $\mu$ s	42.7 $\mu$ s
	1st V: Data Reg.	2nd V:Data Reg.	17.0 $\mu$ s	16.9 $\mu$ s
		V:Bit Reg.	17.0 $\mu$ s	16.9 $\mu$ s
		K:Constant	11.7 $\mu$ s	11.6 $\mu$ s
		P:Indir. (Data)	42.8 $\mu$ s	42.7 $\mu$ s
		P:Indir. (Bit)	42.8 $\mu$ s	42.7 $\mu$ s
	V: Bit Reg.	V:Data Reg.	17.0 $\mu$ s	16.9 $\mu$ s
		V:Bit Reg.	17.0 $\mu$ s	16.9 $\mu$ s
		K:Constant	11.7 $\mu$ s	11.6 $\mu$ s
		P:Indir. (Data)	42.8 $\mu$ s	42.7 $\mu$ s
		P:Indir. (Bit)	42.8 $\mu$ s	42.7 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	42.9 $\mu$ s	42.8 $\mu$ s
		V:Bit Reg.	42.9 $\mu$ s	42.8 $\mu$ s
		K:Constant	38.1 $\mu$ s	38.0 $\mu$ s
		P:Indir. (Data)	66.8 $\mu$ s	66.7 $\mu$ s
		P:Indir. (Bit)	66.8 $\mu$ s	66.7 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	42.9 $\mu$ s	42.8 $\mu$ s
		V:Bit Reg.	42.9 $\mu$ s	42.8 $\mu$ s
		K:Constant	38.1 $\mu$ s	38.0 $\mu$ s
		P:Indir. (Data)	66.8 $\mu$ s	66.7 $\mu$ s
		P:Indir. (Bit)	66.8 $\mu$ s	66.7 $\mu$ s
STRN	1st T, CT	2nd V:Data Reg.	17.1 $\mu$ s	17.2 $\mu$ s
		V:Bit Reg.	17.1 $\mu$ s	17.2 $\mu$ s
		K:Constant	11.9 $\mu$ s	12.0 $\mu$ s
		P:Indir. (Data)	43.0 $\mu$ s	43.1 $\mu$ s
		P:Indir. (Bit)	43.0 $\mu$ s	43.1 $\mu$ s
	1st V: Data Reg.	2nd V:Data Reg.	17.1 $\mu$ s	17.2 $\mu$ s
		V:Bit Reg.	17.1 $\mu$ s	17.2 $\mu$ s
		K:Constant	11.9 $\mu$ s	12.0 $\mu$ s
		P:Indir. (Data)	43.0 $\mu$ s	43.1 $\mu$ s
		P:Indir. (Bit)	43.0 $\mu$ s	43.1 $\mu$ s
	V: Bit Reg.	V:Data Reg.	17.1 $\mu$ s	17.2 $\mu$ s
		V:Bit Reg.	17.1 $\mu$ s	17.2 $\mu$ s
		K:Constant	11.9 $\mu$ s	12.0 $\mu$ s
		P:Indir. (Data)	43.0 $\mu$ s	43.1 $\mu$ s
		P:Indir. (Bit)	43.0 $\mu$ s	43.1 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	43.0 $\mu$ s	43.1 $\mu$ s
		V:Bit Reg.	43.0 $\mu$ s	43.1 $\mu$ s
		K:Constant	38.3 $\mu$ s	38.4 $\mu$ s
		P:Indir. (Data)	67.0 $\mu$ s	67.1 $\mu$ s
		P:Indir. (Bit)	67.0 $\mu$ s	67.1 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	43.0 $\mu$ s	43.1 $\mu$ s
		V:Bit Reg.	43.0 $\mu$ s	43.1 $\mu$ s
		K:Constant	38.3 $\mu$ s	38.4 $\mu$ s
		P:Indir. (Data)	67.0 $\mu$ s	67.1 $\mu$ s
		P:Indir. (Bit)	67.0 $\mu$ s	67.1 $\mu$ s

Comparative Boolean (cont.)			DL05	
Instruc	Legal Data Types		Execute	Not Execute
OR	1st T, CT	2nd V:Data Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		V:Bit Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		K:Constant	11.5 $\mu$ s	11.4 $\mu$ s
		P:Indir. (Data)	42.6 $\mu$ s	42.5 $\mu$ s
		P:Indir. (Bit)	42.6 $\mu$ s	42.5 $\mu$ s
	1st V: Data Reg.	2nd V:Data Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		V:Bit Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		K:Constant	11.5 $\mu$ s	11.4 $\mu$ s
		P:Indir. (Data)	42.6 $\mu$ s	42.5 $\mu$ s
		P:Indir. (Bit)	42.6 $\mu$ s	42.5 $\mu$ s
	V: Bit Reg.	V:Data Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		V:Bit Reg.	16.6 $\mu$ s	16.5 $\mu$ s
		K:Constant	11.5 $\mu$ s	11.4 $\mu$ s
		P:Indir. (Data)	42.6 $\mu$ s	42.5 $\mu$ s
		P:Indir. (Bit)	42.6 $\mu$ s	42.5 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	42.6 $\mu$ s	42.5 $\mu$ s
		V:Bit Reg.	42.6 $\mu$ s	42.5 $\mu$ s
		K:Constant	37.7 $\mu$ s	37.6 $\mu$ s
		P:Indir. (Data)	66.5 $\mu$ s	66.4 $\mu$ s
		P:Indir. (Bit)	66.5 $\mu$ s	66.4 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	42.6 $\mu$ s	42.5 $\mu$ s
		V:Bit Reg.	42.6 $\mu$ s	42.5 $\mu$ s
		K:Constant	37.7 $\mu$ s	37.6 $\mu$ s
		P:Indir. (Data)	66.5 $\mu$ s	66.4 $\mu$ s
		P:Indir. (Bit)	66.5 $\mu$ s	66.4 $\mu$ s
ORN	1st T, CT	2nd V:Data Reg.	15.8 $\mu$ s	15.8 $\mu$ s
		V:Bit Reg.	15.8 $\mu$ s	15.8 $\mu$ s
		K:Constant	10.8 $\mu$ s	10.8 $\mu$ s
		P:Indir. (Data)	41.9 $\mu$ s	41.9 $\mu$ s
		P:Indir. (Bit)	41.9 $\mu$ s	41.9 $\mu$ s
	1st V: Data Reg.	2nd V:Data Reg.	15.8 $\mu$ s	15.8 $\mu$ s
		V:Bit Reg.	15.8 $\mu$ s	15.8 $\mu$ s
		K:Constant	10.8 $\mu$ s	10.8 $\mu$ s
		P:Indir. (Data)	41.9 $\mu$ s	41.9 $\mu$ s
		P:Indir. (Bit)	41.9 $\mu$ s	41.9 $\mu$ s
	V: Bit Reg.	V:Data Reg.	15.8 $\mu$ s	15.8 $\mu$ s
		V:Bit Reg.	15.8 $\mu$ s	15.8 $\mu$ s
		K:Constant	10.8 $\mu$ s	10.8 $\mu$ s
		P:Indir. (Data)	41.9 $\mu$ s	41.9 $\mu$ s
		P:Indir. (Bit)	41.9 $\mu$ s	41.9 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	41.8 $\mu$ s	41.8 $\mu$ s
		V:Bit Reg.	41.8 $\mu$ s	41.8 $\mu$ s
		K:Constant	37.1 $\mu$ s	37.1 $\mu$ s
		P:Indir. (Data)	65.9 $\mu$ s	65.9 $\mu$ s
		P:Indir. (Bit)	65.9 $\mu$ s	65.9 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	41.8 $\mu$ s	41.8 $\mu$ s
		V:Bit Reg.	41.8 $\mu$ s	41.8 $\mu$ s
		K:Constant	37.1 $\mu$ s	37.1 $\mu$ s
		P:Indir. (Data)	65.9 $\mu$ s	65.9 $\mu$ s
		P:Indir. (Bit)	65.9 $\mu$ s	65.9 $\mu$ s



Comparative Boolean (cont.)			DL05	
Instruc	Legal Data Types		Execute	Not Execute
AND	1st T, CT	2nd V:Data Reg.	15.6 $\mu$ s	15.6 $\mu$ s
		V:Bit Reg.	15.6 $\mu$ s	15.6 $\mu$ s
		K:Constant	10.9 $\mu$ s	10.9 $\mu$ s
		P:Indir. (Data)	41.6 $\mu$ s	41.6 $\mu$ s
		P:Indir. (Bit)	41.6 $\mu$ s	41.6 $\mu$ s
	1st V: Data Reg.	2nd V:Data Reg.	15.6 $\mu$ s	15.6 $\mu$ s
		V:Bit Reg.	15.6 $\mu$ s	15.6 $\mu$ s
		K:Constant	10.9 $\mu$ s	10.9 $\mu$ s
		P:Indir. (Data)	41.6 $\mu$ s	41.6 $\mu$ s
		P:Indir. (Bit)	41.6 $\mu$ s	41.6 $\mu$ s
	V: Bit Reg.	V:Data Reg.	15.6 $\mu$ s	15.6 $\mu$ s
		V:Bit Reg.	15.6 $\mu$ s	15.6 $\mu$ s
		K:Constant	10.9 $\mu$ s	10.9 $\mu$ s
		P:Indir. (Data)	41.6 $\mu$ s	41.6 $\mu$ s
		P:Indir. (Bit)	41.6 $\mu$ s	41.6 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	41.6 $\mu$ s	41.6 $\mu$ s
		V:Bit Reg.	41.6 $\mu$ s	41.6 $\mu$ s
		K:Constant	36.9 $\mu$ s	36.9 $\mu$ s
		P:Indir. (Data)	65.6 $\mu$ s	65.6 $\mu$ s
		P:Indir. (Bit)	65.6 $\mu$ s	65.6 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	41.6 $\mu$ s	41.6 $\mu$ s
		V:Bit Reg.	41.6 $\mu$ s	41.6 $\mu$ s
		K:Constant	36.9 $\mu$ s	36.9 $\mu$ s
		P:Indir. (Data)	65.6 $\mu$ s	65.6 $\mu$ s
		P:Indir. (Bit)	65.6 $\mu$ s	65.6 $\mu$ s
ANDN	1st T, CT	2nd V:Data Reg.	15.8 $\mu$ s	15.7
		V:Bit Reg.	15.8 $\mu$ s	15.7
		K:Constant	10.8 $\mu$ s	10.8
		P:Indir. (Data)	41.9 $\mu$ s	41.9
		P:Indir. (Bit)	41.9 $\mu$ s	41.9
	1st V: Data Reg.	2nd V:Data Reg.	15.8 $\mu$ s	15.7 $\mu$ s
		V:Bit Reg.	15.8 $\mu$ s	15.7 $\mu$ s
		K:Constant	10.8 $\mu$ s	10.8 $\mu$ s
		P:Indir. (Data)	41.9 $\mu$ s	41.9 $\mu$ s
		P:Indir. (Bit)	41.9 $\mu$ s	41.9 $\mu$ s
	V: Bit Reg.	V:Data Reg.	15.8 $\mu$ s	15.7 $\mu$ s
		V:Bit Reg.	15.8 $\mu$ s	15.7 $\mu$ s
		K:Constant	10.8 $\mu$ s	10.8 $\mu$ s
		P:Indir. (Data)	41.9 $\mu$ s	41.9 $\mu$ s
		P:Indir. (Bit)	41.9 $\mu$ s	41.9 $\mu$ s
	P:Indir. (Data)	V:Data Reg.	41.8 $\mu$ s	41.8 $\mu$ s
		V:Bit Reg.	41.8 $\mu$ s	41.8 $\mu$ s
		K:Constant	37.8 $\mu$ s	37.8 $\mu$ s
		P:Indir. (Data)	65.9 $\mu$ s	65.9 $\mu$ s
		P:Indir. (Bit)	65.9 $\mu$ s	65.9 $\mu$ s
	P:Indir. (Bit)	V:Data Reg.	41.8 $\mu$ s	41.8 $\mu$ s
		V:Bit Reg.	41.8 $\mu$ s	41.8 $\mu$ s
		K:Constant	37.8 $\mu$ s	37.8 $\mu$ s
		P:Indir. (Data)	65.9 $\mu$ s	65.9 $\mu$ s
		P:Indir. (Bit)	65.9 $\mu$ s	65.9 $\mu$ s

## Immediate Instructions

Immediate Instructions		DL05	
Instruction	Legal Data Types	Execute	Not Execute
STRI	X	38.2 $\mu$ s	38.2 $\mu$ s
STRNI	X	38.5 $\mu$ s	38.5 $\mu$ s
ORI	X	37.7 $\mu$ s	37.7 $\mu$ s
ORNI	X	38.2 $\mu$ s	38.2 $\mu$ s
ANDI	X	37.7 $\mu$ s	37.7 $\mu$ s
ANDNI	X	38.1 $\mu$ s	38.1 $\mu$ s
OUTI	Y	77.5 $\mu$ s	77.5 $\mu$ s
OROUTI	Y	85.7 $\mu$ s	85.7 $\mu$ s
SETI	1st #: Y 2nd #: Y (N pt)	68.0 $\mu$ s 108.5 $\mu$ s+1.6 $\mu$ sxN	3.2 $\mu$ s 4.3 $\mu$ s
RSTI	1st #: Y 2nd #: Y (N pt)	66.9 10.8 $\mu$ s+1.7 $\mu$ sxN	3.3 $\mu$ s 4.4 $\mu$ s

## Timer, Counter, and Shift Register

Timer, Counter, and Shift Register			DL05	
Instruction	Legal Data Types		Execute	Not Execute
TMR	1st	2nd		
	T	V:Data Reg.	64.0 $\mu$ s	59.0 $\mu$ s
		V:Bit Reg.	64.0 $\mu$ s	59.0 $\mu$ s
		K:Constant	58.1 $\mu$ s	53.1 $\mu$ s
		P:Indir. (Data)	91.0 $\mu$ s	86.0 $\mu$ s
		P:Indir. (Bit)	91.0 $\mu$ s	86.0 $\mu$ s
TMRF	1st	2nd		
	T	V:Data Reg.	64.8 $\mu$ s	57.3 $\mu$ s
		V:Bit Reg.	64.8 $\mu$ s	57.3 $\mu$ s
		K:Constant	59.5 $\mu$ s	52.0 $\mu$ s
		P:Indir. (Data)	92.5 $\mu$ s	85.0 $\mu$ s
		P:Indir. (Bit)	92.5 $\mu$ s	85.0 $\mu$ s
TMRA	1st	2nd		
	T	V:Data Reg.	72.2 $\mu$ s	66.5 $\mu$ s
		V:Bit Reg.	72.2 $\mu$ s	66.5 $\mu$ s
		K:Constant	65.7 $\mu$ s	60.0 $\mu$ s
		P:Indir. (Data)	99.2 $\mu$ s	93.5 $\mu$ s
		P:Indir. (Bit)	99.2 $\mu$ s	93.5 $\mu$ s
TMRAF	1st	2nd		
	T	V:Data Reg.	71.4 $\mu$ s	65.6 $\mu$ s
		V:Bit Reg.	71.4 $\mu$ s	65.6 $\mu$ s
		K:Constant	65.2 $\mu$ s	59.5 $\mu$ s
		P:Indir. (Data)	98.9 $\mu$ s	93.1 $\mu$ s
		P:Indir. (Bit)	98.9 $\mu$ s	93.1 $\mu$ s
CNT	1st	2nd		
	CT	V:Data Reg.	79.5 $\mu$ s	66.9 $\mu$ s
		V:Bit Reg.	79.5 $\mu$ s	66.9 $\mu$ s
		K:Constant	73.7 $\mu$ s	61.1 $\mu$ s
		P:Indir. (Data)	107.0 $\mu$ s	94.4 $\mu$ s
		P:Indir. (Bit)	107.0 $\mu$ s	94.4 $\mu$ s

Timer, Counter, and Shift Register			DL05	
Instruction	Legal Data Types		Execute	Not Execute
SGCNT	1st CT	2nd		
		V:Data Reg.	76.5 $\mu$ s	75.3 $\mu$ s
		V:Bit Reg.	76.5 $\mu$ s	75.3 $\mu$ s
		K:Constant	70.8 $\mu$ s	69.6 $\mu$ s
		P:Indir. (Data)	106.8 $\mu$ s	105.4 $\mu$ s
		P:Indir. (Bit)	106.8 $\mu$ s	105.4 $\mu$ s
UDC	1st CT	2nd		
		V:Data Reg.	118.3 $\mu$ s	101.1 $\mu$ s
		V:Bit Reg.	118.3 $\mu$ s	101.1 $\mu$ s
		K:Constant	111.8 $\mu$ s	94.6 $\mu$ s
		P:Indir. (Data)	145.0 $\mu$ s	128.0 $\mu$ s
		P:Indir. (Bit)	145.0 $\mu$ s	128.0 $\mu$ s
SR	C	(N points to shift)	43.4 $\mu$ s+25.6 $\mu$ sxN	31.4 $\mu$ s

### Accumulator Data Instructions

Accumulator / Stack Load and Output Data Instructions			DL05	
Instruction	Legal Data Types		Execute	Not Execute
LD	V:Data Reg.		43.7 $\mu$ s	3.7 $\mu$ s
	V:Bit Reg.		43.7 $\mu$ s	3.7 $\mu$ s
	K:Constant		42.7 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)		68.7 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)		68.7 $\mu$ s	3.7 $\mu$ s
LDD	V:Data Reg.		47.1 $\mu$ s	3.7 $\mu$ s
	V:Bit Reg.		47.1 $\mu$ s	3.7 $\mu$ s
	K:Constant		42.8 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)		72.2 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)		72.2 $\mu$ s	3.7 $\mu$ s
LDF	1st	2nd		
	X, Y, C, S T, CT, SP	K:Constant (N pt)	65.8 $\mu$ s+13.9 $\mu$ sxN	4.9 $\mu$ s
LDA	O: (Octal constant for address)		42.7 $\mu$ s	3.7 $\mu$ s
OUT	V:Data Reg.		16.6 $\mu$ s	3.7 $\mu$ s
	V:Bit Reg.		16.6 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)		41.8 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)		41.8 $\mu$ s	3.7 $\mu$ s
OUTD	V:Data Reg.		18.1 $\mu$ s	3.8 $\mu$ s
	V:Bit Reg.		18.1 $\mu$ s	3.8 $\mu$ s
	P:Indir. (Data)		43.3 $\mu$ s	3.8 $\mu$ s
	P:Indir. (Bit)		43.3 $\mu$ s	3.8 $\mu$ s
OUTF	1st	2nd		
	X, Y, C	K:Constant (N pt)	61.9 $\mu$ s+22 $\mu$ sxN	4.7 $\mu$ s
POP	None		41.1 $\mu$ s	2.7 $\mu$ s

Logical  
Instructions

Logical (Accumulator) Instructions		DL05	
Instruction	Legal Data Types	Execute	Not Execute
AND	V:Data Reg.	23.5 $\mu$ s	3.9 $\mu$ s
	V:Bit Reg.	23.5 $\mu$ s	3.9 $\mu$ s
	P:Indir. (Data)	48.3 $\mu$ s	3.9 $\mu$ s
	P:Indir. (Bit)	48.3 $\mu$ s	3.9 $\mu$ s
ANDD	V:Data Reg.	23.3 $\mu$ s	3.7 $\mu$ s
	V:Bit Reg.	23.3 $\mu$ s	3.7 $\mu$ s
	K:Constant	19.0 $\mu$ s	3.9 $\mu$ s
	P:Indir. (Data)	48.3 $\mu$ s	3.9 $\mu$ s
	P:Indir. (Bit)	48.3 $\mu$ s	3.9 $\mu$ s
OR	V:Data Reg.	23.9 $\mu$ s	3.7 $\mu$ s
	V:Bit Reg.	23.9 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)	48.8 $\mu$ s	3.8 $\mu$ s
	P:Indir. (Bit)	48.8 $\mu$ s	3.8 $\mu$ s
ORD	V:Data Reg.	23.8 $\mu$ s	3.8 $\mu$ s
	V:Bit Reg.	23.8 $\mu$ s	3.8 $\mu$ s
	K:Constant	19.4 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)	48.6 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)	48.6 $\mu$ s	3.7 $\mu$ s
XOR	V:Data Reg.	23.5 $\mu$ s	3.9 $\mu$ s
	V:Bit Reg.	23.5 $\mu$ s	3.9 $\mu$ s
	P:Indir. (Data)	48.3 $\mu$ s	3.9 $\mu$ s
	P:Indir. (Bit)	48.3 $\mu$ s	3.9 $\mu$ s
XORD	V:Data Reg.	23.3 $\mu$ s	3.7 $\mu$ s
	V:Bit Reg.	23.3 $\mu$ s	3.7 $\mu$ s
	K:Constant	19.0 $\mu$ s	3.9 $\mu$ s
	P:Indir. (Data)	48.3 $\mu$ s	3.9 $\mu$ s
	P:Indir. (Bit)	48.3 $\mu$ s	3.9 $\mu$ s
CMP	V:Data Reg.	25.4 $\mu$ s	3.7 $\mu$ s
	V:Bit Reg.	25.4 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)	50.0 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)	50.0 $\mu$ s	3.7 $\mu$ s
CMPD	V:Data Reg.	37.3 $\mu$ s	3.9 $\mu$ s
	V:Bit Reg.	37.3 $\mu$ s	3.9 $\mu$ s
	K:Constant	32.7 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)	62.0 $\mu$ s	3.8 $\mu$ s
	P:Indir. (Bit)	62.0 $\mu$ s	3.8 $\mu$ s

## Math Instructions

Math Instructions (Accumulator)		DL05	
Instruction	Legal Data Types	Execute	Not Execute
ADD	V:Data Reg.	140.7 $\mu$ s	3.7 $\mu$ s
	V:Bit Reg.	140.7 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)	185.1 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)	185.1 $\mu$ s	3.7 $\mu$ s
ADDD	V:Data Reg.	152.4 $\mu$ s	3.9 $\mu$ s
	V:Bit Reg.	152.4 $\mu$ s	3.9 $\mu$ s
	K:Constant	123.2 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)	193.5 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)	193.5 $\mu$ s	3.7 $\mu$ s
SUB	V:Data Reg.	148.1 $\mu$ s	3.7 $\mu$ s
	V:Bit Reg.	148.1 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)	192.7 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)	192.7 $\mu$ s	3.7 $\mu$ s

Math Instructions (Accumulator)		DL05	
Instruction	Legal Data Types	Execute	Not Execute
SUBD	V:Data Reg.	157.0 $\mu$ s	3.9 $\mu$ s
	V:Bit Reg.	157.0 $\mu$ s	3.9 $\mu$ s
	K:Constant	131.4 $\mu$ s	3.9 $\mu$ s
	P:Indir. (Data)	201.9 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)	201.9 $\mu$ s	3.7 $\mu$ s
MUL	V:Data Reg.	305.0 $\mu$ s	3.7 $\mu$ s
	V:Bit Reg.	305.0 $\mu$ s	3.7 $\mu$ s
	K:Constant	289.1 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)	349.6 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)	349.6 $\mu$ s	3.7 $\mu$ s
MULD	V:Data Reg.	1261.3 $\mu$ s	3.8 $\mu$ s
	V:Bit Reg.	1261.3 $\mu$ s	3.8 $\mu$ s
	P:Indir. (Data)	1307.3 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)	1307.3 $\mu$ s	3.7 $\mu$ s
DIV	V:Data Reg.	557.1 $\mu$ s	3.7 $\mu$ s
	V:Bit Reg.	557.1 $\mu$ s	3.7 $\mu$ s
	K:Constant	530.5 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Data)	580.5 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)	580.5 $\mu$ s	3.7 $\mu$ s
DIVD	V:Data Reg.	562.2 $\mu$ s	3.8 $\mu$ s
	V:Bit Reg.	562.2 $\mu$ s	3.8 $\mu$ s
	P:Indir. (Data)	596.4 $\mu$ s	3.7 $\mu$ s
	P:Indir. (Bit)	596.4 $\mu$ s	3.7 $\mu$ s
INC	V:Data Reg.	35.7 $\mu$ s	3.4 $\mu$ s
	V:Bit Reg.	35.7 $\mu$ s	3.4 $\mu$ s
	P:Indir. (Data)	60.2 $\mu$ s	3.4 $\mu$ s
	P:Indir. (Bit)	60.2 $\mu$ s	3.4 $\mu$ s
DEC	V:Data Reg.	41.4 $\mu$ s	3.3 $\mu$ s
	V:Bit Reg.	41.4 $\mu$ s	3.3 $\mu$ s
	P:Indir. (Data)	64.2 $\mu$ s	3.3 $\mu$ s
	P:Indir. (Bit)	64.2 $\mu$ s	3.3 $\mu$ s
ADDB	V:Data Reg.	69.5 $\mu$ s	3.3 $\mu$ s
	V:Bit Reg.	69.5 $\mu$ s	3.3 $\mu$ s
	K:Constant	67.3 $\mu$ s	3.3 $\mu$ s
	P:Indir. (Data)	94.2 $\mu$ s	3.6 $\mu$ s
	P:Indir. (Bit)	94.2 $\mu$ s	3.6 $\mu$ s
SUBB	V:Data Reg.	69.3 $\mu$ s	3.3 $\mu$ s
	V:Bit Reg.	69.3 $\mu$ s	3.3 $\mu$ s
	K:Constant	67.8 $\mu$ s	3.5 $\mu$ s
	P:Indir. (Data)	94.3 $\mu$ s	3.2 $\mu$ s
	P:Indir. (Bit)	94.3 $\mu$ s	3.2 $\mu$ s
MULB	V:Data Reg.	23.4 $\mu$ s	3.2 $\mu$ s
	V:Bit Reg.	23.4 $\mu$ s	3.2 $\mu$ s
	K:Constant	19.8 $\mu$ s	3.3 $\mu$ s
	P:Indir. (Data)	48.5 $\mu$ s	3.2 $\mu$ s
	P:Indir. (Bit)	48.5 $\mu$ s	3.2 $\mu$ s
DIVB	V:Data Reg.	76.1 $\mu$ s	3.3 $\mu$ s
	V:Bit Reg.	76.1 $\mu$ s	3.3 $\mu$ s
	K:Constant	76.9 $\mu$ s	3.3 $\mu$ s
	P:Indir. (Data)	105.5 $\mu$ s	3.2 $\mu$ s
	P:Indir. (Bit)	105.5 $\mu$ s	3.2 $\mu$ s

Math Instructions (Accumulator)		DL05	
Instruction	Legal Data Types	Execute	Not Execute
INCB	V:Data Reg.	23.0 $\mu$ s	3.4 $\mu$ s
	V:Bit Reg.	23.0 $\mu$ s	3.4 $\mu$ s
	P:Indir. (Data)	46.8 $\mu$ s	3.3 $\mu$ s
	P:Indir. (Bit)	46.8 $\mu$ s	3.3 $\mu$ s
DECB	V:Data Reg.	23.2 $\mu$ s	2.5 $\mu$ s
	V:Bit Reg.	23.2 $\mu$ s	2.5 $\mu$ s
	P:Indir. (Data)	47.5 $\mu$ s	3.4 $\mu$ s
	P:Indir. (Bit)	47.5 $\mu$ s	3.4 $\mu$ s

## Bit Instructions

Bit Instructions (Accumulator)		DL05	
Instruction	Legal Data Types	Execute	Not Execute
SUM	None	19.2 $\mu$ s	2.3 $\mu$ s
SHFR	V:Data Reg. (N bits)	23.0 $\mu$ s	3.4 $\mu$ s
	V:Bit Reg. (N bits)	23.0 $\mu$ s	3.4 $\mu$ s
	K:Constant (N bits)	20.3 $\mu$ s	3.3 $\mu$ s
SHFL	V:Data Reg. (N bits)	29.7 $\mu$ s	3.4 $\mu$ s
	V:Bit Reg. (N bits)	29.7 $\mu$ s	3.4 $\mu$ s
	K:Constant (N bits)	20.4 $\mu$ s	3.3 $\mu$ s
ENCO	None	12.6 $\mu$ s	2.3 $\mu$ s
DECO	None	20.3 $\mu$ s	2.3 $\mu$ s

## Number Conversion Instructions

Number Conversion Instructions (Accumulator)		DL05	
Instruction	Legal Data Types	Execute	Not Execute
BIN	None	75.8 $\mu$ s	2.3 $\mu$ s
BCD	None	159.9 $\mu$ s	2.2 $\mu$ s
INV	None	6.2 $\mu$ s	2.3 $\mu$ s
ATH	V:Data Reg. (N bits) V:Bit Reg. (N bits)	97 $\mu$ s+20 $\mu$ s $\times$ N	3.3 $\mu$ s
HTA	V:Data Reg. (N bits) V:Bit Reg. (N bits)	98 $\mu$ s+27 $\mu$ s $\times$ N	2.1 $\mu$ s
GRAY	None	224.7 $\mu$ s	2.3 $\mu$ s
SFLDGT	None	95.3 $\mu$ s	2.2 $\mu$ s

## Table Instructions

Table Instructions		DL05	
Instruction	Legal Data Types	Execute	Not Execute
MOV	V:Data Reg. / V:Bit Reg. P:Indir. (Data) / P:Indir. (Bit)	63 $\mu$ s+16 $\mu$ s $\times$ N	3.3 $\mu$ s
MOVMC	Move from E <sup>2</sup> to V:Data Reg. Move from E <sup>2</sup> to V:Bit Reg. N= #of words	50 $\mu$ s+15 $\mu$ s $\times$ N	3.3 $\mu$ s
LDLBL	K	33.5 $\mu$ s	4.2 $\mu$ s

**CPU Control Instructions**

CPU Control Instructions		DL05	
Instruction	Legal Data Types	Execute	Not Execute
NOP	None	1.1 $\mu$ s	1.1 $\mu$ s
END	None	24.0 $\mu$ s	24.0 $\mu$ s
STOP	None	10.0 $\mu$ s	1.1 $\mu$ s
RSTWDT	None	5.9 $\mu$ s	2.2 $\mu$ s
NOT	None	1.6 $\mu$ s	1.6 $\mu$ s

**Program Control Instructions**

Program Control Instructions		DL05	
Instruction	Legal Data Types	Execute	Not Execute
FOR	V, K	125.9 $\mu$ s	14.5 $\mu$ s
NEXT	None	64.4 $\mu$ s	64.4 $\mu$ s
GTS	K	27.5 $\mu$ s	14.8 $\mu$ s
SBR	K	1.5 $\mu$ s	1.5 $\mu$ s
RTC	None	25.7 $\mu$ s	12.1 $\mu$ s
RT	None	21.2 $\mu$ s	21.2 $\mu$ s
MLS	K (1–7)	35.2 $\mu$ s	35.2 $\mu$ s
MLR	K (0–7)	30.9 $\mu$ s	30.9 $\mu$ s

**Interrupt Instructions**

Interrupt Instructions		DL05	
Instruction	Legal Data Types	Execute	Not Execute
ENI	None	24.2 $\mu$ s	2.7 $\mu$ s
DISI	None	9.4 $\mu$ s	2.3 $\mu$ s
INT	O(0,1)	7.5 $\mu$ s	–
IRTC	None	0.9 $\mu$ s	1.3 $\mu$ s
IRT	None	6.6 $\mu$ s	–

**Network Instructions**

Network Instructions		DL05	
Instruction	Legal Data Types	Execute	Not Execute
RX	X, Y, C, T, CT, SP, S, \$	852.0 $\mu$ s	4.4 $\mu$ s
	V:Data Reg.	852.0 $\mu$ s	4.4 $\mu$ s
	V:Bit Reg.	852.0 $\mu$ s	4.4 $\mu$ s
	P:Indir. (Data)	868.2 $\mu$ s	4.2 $\mu$ s
	P:Indir. (Bit)	868.2 $\mu$ s	4.2 $\mu$ s
WX	X, Y, C, T, CT, SP, S, \$	1614.0 $\mu$ s	4.4 $\mu$ s
	V:Data Reg.	1614.0 $\mu$ s	4.4 $\mu$ s
	V:Bit Reg.	1614.0 $\mu$ s	4.4 $\mu$ s
	P:Indir. (Data)	1630.0 $\mu$ s	4.4 $\mu$ s
	P:Indir. (Bit)	1630.0 $\mu$ s	4.4 $\mu$ s

**Message Instructions**

Message Instructions		DL05	
Instruction	Legal Data Types	Execute	Not Execute
FAULT	V:Data Reg. V:Bit Reg. K:Constant	65.0 $\mu$ s 65.0 $\mu$ s 204.7 $\mu$ s	4.4 $\mu$ s 4.4 $\mu$ s 4.4 $\mu$ s
DLBL	K	–	–
NCON	K	–	–
ACON	A	–	–
PRINT	ASCII	631.0 $\mu$ s	3.6 $\mu$ s

**RLL *PLUS* Instructions**

RLL <i>PLUS</i> Instructions		DL05	
Instruction	Legal Data Types	Execute	Not Execute
ISG	S	44.0 $\mu$ s	41.1 $\mu$ s
SG	S	44.0 $\mu$ s	41.1 $\mu$ s
JMP	S	76.0 $\mu$ s	9.3 $\mu$ s
NJMP	S	77.4 $\mu$ s	9.3 $\mu$ s
CV	S	42.1 $\mu$ s	27.5 $\mu$ s
CVJMP	S	89.5 $\mu$ s	17.6 $\mu$ s

**Drum Instructions**

Drum Instructions		DL05	
Instruction	Legal Data Types	Execute	Not Execute
DRUM	CT	840.0 $\mu$ s	339.6 $\mu$ s
EDRUM	CT	753.2 $\mu$ s	357.0 $\mu$ s



# Special Relays

---

In This Appendix. . . .  
— DL05 PLC Special Relays

## DL05 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 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

<b>SP0</b>	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.
<b>SP1</b>	Always ON	provides a contact to insure an instruction is executed every scan.
<b>SP3</b>	1 minute clock	on for 30 seconds and off for 30 seconds.
<b>SP4</b>	1 second clock	on for 0.5 second and off for 0.5 second.
<b>SP5</b>	100 ms clock	on for 50 ms. and off for 50 ms.
<b>SP6</b>	50 ms clock	on for 25 ms. and off for 25 ms.
<b>SP7</b>	Alternate scan	on every other scan.

### CPU Status Relays

<b>SP11</b>	Forced run mode	on when the mode switch is in the run position and the CPU is running.
<b>SP12</b>	Terminal run mode	on when the CPU is in the run mode.
<b>SP13</b>	Test run mode	on when the CPU is in the test run mode.
<b>SP15</b>	Test stop mode	on when the CPU is in the test stop mode.
<b>SP16</b>	Terminal PGM mode	on when the mode switch is the the TERM position and the CPU is in program mode.
<b>SP17</b>	Forced stop	on when the mode switch is in the STOP position.
<b>SP20</b>	Forced stop mode	on when the STOP instruction is executed.
<b>SP22</b>	Interrupt enabled	on when interrupts have been enabled using the ENI instruction.

**System Monitoring**

<b>SP36</b>	Override setup relay	on when the override function is used.
<b>SP37</b>	Scan control error	on when the actual scan time runs over the prescribed scan time.
<b>SP40</b>	Critical error	on when a critical error such as I/O communication loss has occurred.
<b>SP41</b>	Warning	on when a non critical error has occurred.
<b>SP42</b>	Diagnostics error	on when a diagnostics error or a system error occurs.
<b>SP44</b>	Program memory error	on when a memory error such as a memory parity error has occurred.
<b>SP45</b>	I/O error	on when an I/O error such as a blown fuse occurs.
<b>SP46</b>	Communications error	on when a communication error occurs on any of the CPU ports.
<b>SP50</b>	Fault instruction	on when a Fault Instruction is executed.
<b>SP51</b>	Watch Dog timeout	on if the CPU Watch Dog timer times out.
<b>SP52</b>	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.
<b>SP53</b>	Solve logic error	on if CPU cannot solve the logic.
<b>SP54</b>	Communication error	on when RX, WX, RD, WT instructions are executed with the wrong parameters.
<b>SP56</b>	Table instruction overrun	on if a table instruction with a pointer is executed and the pointer value is outside the table boundary.

**Accumulator Status**

<b>SP60</b>	Value less than	on when the accumulator value is less than the instruction value.
<b>SP61</b>	Value equal to	on when the accumulator value is equal to the instruction value.
<b>SP62</b>	Greater than	on when the accumulator value is greater than the instruction value.
<b>SP63</b>	Zero	on when the result of the instruction is zero (in the accumulator.)
<b>SP64</b>	Half borrow	on when the 16 bit subtraction instruction results in a borrow.
<b>SP65</b>	Borrow	on when the 32 bit subtraction instruction results in a borrow.
<b>SP66</b>	Half carry	on when the 16 bit addition instruction results in a carry.
<b>SP67</b>	Carry	when the 32 bit addition instruction results in a carry.
<b>SP70</b>	Sign	on anytime the value in the accumulator is negative.
<b>SP71</b>	Pointer reference error	on when the V-memory specified by a pointer (P) is not valid.
<b>SP73</b>	Overflow	on if overflow occurs in the accumulator when a signed addition or subtraction results in an incorrect sign bit.
<b>SP75</b>	Data error	on if a BCD number is expected and a non-BCD number is encountered.
<b>SP76</b>	Load zero	on when any instruction loads a value of zero into the accumulator.

**HSIO Pulse  
Output Relay**

<b>SP104</b>	Profile Complete	on when the pulse output profile is completed. (Mode 30)
--------------	------------------	--

**Communication  
Monitoring Relays**

<b>SP116</b>	CPU port busy Port 2	on when port 2 is the master and sending data.
<b>SP117</b>	Communications error Port 2	on when port 2 is the master and has a communication error.

**Equal Relays for  
HSIO Mode 10  
Counter Presets**

<b>SP540</b>	Current = target value	on when the counter current value equals the value in V2320 / V2321.
<b>SP541</b>	Current = target value	on when the counter current value equals the value in V2322 / V2323.
<b>SP542</b>	Current = target value	on when the counter current value equals the value in V2324 / V2325.
<b>SP543</b>	Current = target value	on when the counter current value equals the value in V2326 / V2327.
<b>SP544</b>	Current = target value	on when the counter current value equals the value in V2330 / V2331.
<b>SP545</b>	Current = target value	on when the counter current value equals the value in V2332 / V2333.
<b>SP546</b>	Current = target value	on when the counter current value equals the value in V2334 / V2335.
<b>SP547</b>	Current = target value	on when the counter current value equals the value in V2336 / V2337.
<b>SP550</b>	Current = target value	on when the counter current value equals the value in V2340 / V2341.
<b>SP551</b>	Current = target value	on when the counter current value equals the value in V2342 / V2343.
<b>SP552</b>	Current = target value	on when the counter current value equals the value in V2344 / V2345.
<b>SP553</b>	Current = target value	on when the counter current value equals the value in V2346 / V2347.
<b>SP554</b>	Current = target value	on when the counter current value equals the value in V2350 / V2351.
<b>SP555</b>	Current = target value	on when the counter current value equals the value in V2352 / V2353.
<b>SP556</b>	Current = target value	on when the counter current value equals the value in V2354 / V2355.
<b>SP557</b>	Current = target value	on when the counter current value equals the value in V2356 / V2357.

<b>SP560</b>	Current = target value	on when the counter current value equals the value in V2360 / V2361.
<b>SP561</b>	Current = target value	on when the counter current value equals the value in V2362 / V2363.
<b>SP562</b>	Current = target value	on when the counter current value equals the value in V2364 / V2365.
<b>SP563</b>	Current = target value	on when the counter current value equals the value in V2366 / V2367.
<b>SP564</b>	Current = target value	on when the counter current value equals the value in V2370 / V2371.
<b>SP565</b>	Current = target value	on when the counter current value equals the value in V2372 / V2373.
<b>SP566</b>	Current = target value	on when the counter current value equals the value in V2374 / V2375.
<b>SP567</b>	Current = target value	on when the counter current value equals the value in V2376 / V2377.

# DL05

## Product Weights

---

In This Appendix. . . .  
— Product Weight Table

---

## Product Weight Table

PLC	Weight
D0-05AR	0.60 lb. (272g)
D0-05DR	0.60 lb. (272g)
D0-05AD	0.58 lb. (263g)
D0-05DD	0.56 lb. (254g)
D0-05AA	0.60 lb. (272g)
D0-05DA	0.60 lb. (272g)
D0-05DR-D	0.56 lb. (254g)
D0-05DD-D	0.58 lb. (263g)

# European Union Directives (CE)

---

In This Appendix. . . .

- European Union (EU) Directives
- Basic EMC Installation Guidelines



## 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 January 1, 1997, 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 its 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, 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
- PLC**Direct** 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>

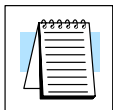
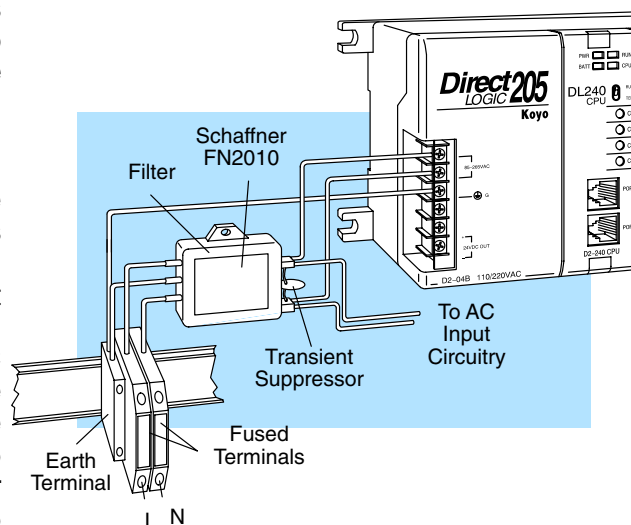
## Basic EMC Installation Guidelines

**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, 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/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

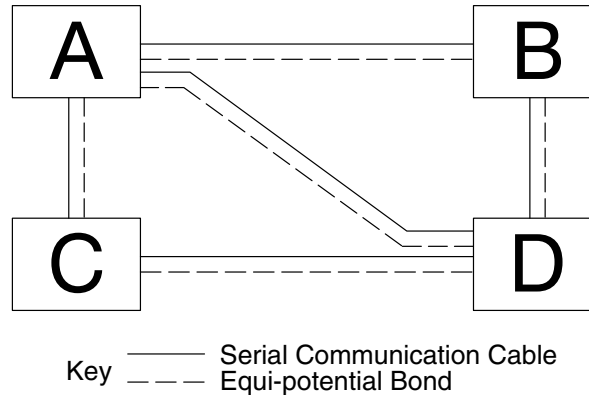
In order to comply with the fire risk requirements of the Low Voltage and Machinery Directive electrical standards EN 61010-1, and EN 60204-1, by limiting the power into “unlimited” mains circuits with power leads reversed, it is necessary to fuse both AC and DC supply inputs. You should also install 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

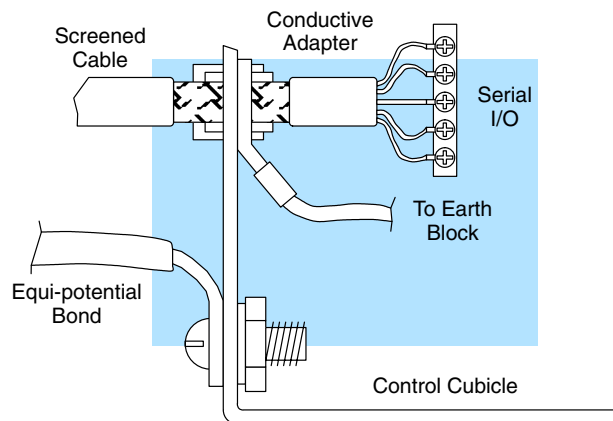
A heavy-duty star earth terminal block should be provided in every cubicle for the connection of all earth ground straps, protective earth ground connections, mains filter earth ground wires, and mechanical assembly earth ground connections. This should be installed to comply with safety and EMC requirements, 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.

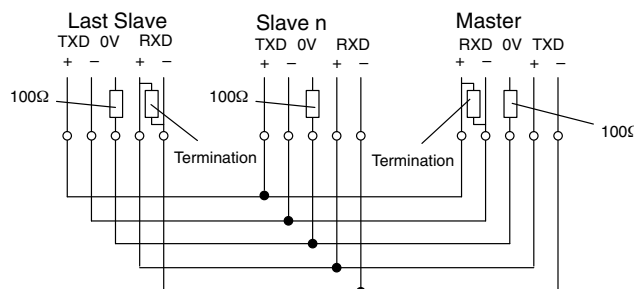
The recommendation is to use shielded cables as electrostatic “pipes” between apparatus and systems, and to run heavy gauge equi-potential bond wires alongside all shielded cables. When a shielded cable runs through the metallic wall of an enclosure or machine, it is recommended in IEC 1000-5-2 that 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 Commission'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 DL05, 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.
- The communication cable and all I/O cables must pass through suitable ferrite beads which must be mounted within the enclosure. The I/O cables can be bundled together and passed through the same ferrite.

Recommended ferrite beads and split cores are detailed below. These were used in extensive EMC tests and found to successfully attenuate radiated emissions to a high degree.

**For I/O Bundle**

Manufacturer	Mfg. Part No.	OD mm	ID mm	L mm	1 Turn L/25 MHz $\Omega$	1 Turn L/100 MHz $\Omega$	2 Turn L/25 MHz $\Omega$	2 Turn L/100 MHz $\Omega$
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
Würth Elektronik	742 700 9	17.5	9.5	28.5	153	210	649	632

**For Communication Cable**

RS Online	222-4416	26.6		16.6	72	132	301	547
Fair-Rite	0444167281	26.6	10.7	16.6	72	132	301	547
Richco	MTFC 231114-T	26.6	10.7	16.6	72	132	301	547

**Items Specific to the DL05**

- 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 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, 5–33
- Accumulating Timer instruction, 5–33
- Accumulator Stack Load Instructions, 5–43
- Add Binary instruction, 5–72, 5–74
- Add Double instruction, 5–64
- Add instruction, 5–63
- Agency approvals, 2–8
- Alarms, PID, 8–53
- And Double instruction, 5–56
- And If Equal instruction, 5–22
- And If Not Equal instruction, 5–22
- And Immediate instruction, 5–27
- And instruction, 5–11, 5–55
  - comparative, 5–25
- And Negative Differential instruction, 5–17
- And Not Immediate instruction, 5–27
- And Not instruction, 5–11
  - comparative, 5–25
- And Positive Differential instruction, 5–17
- And Store instruction, 5–12
- ASCII Constant instruction, 5–106, 5–108
- ASCII TO HEX instruction, 5–86
- Auxiliary Functions, 4–8
- Auxiliary functions, A–2

## B

- Bias freeze, 8–37

- Binary Coded Decimal instruction, 5–82, 5–84
- Binary instruction, 5–83
- Bit Operation Instructions, 5–78
- Boolean Instructions, 5–4, 5–9
- Bumpless transfer, 8–25

## C

- Cables
  - operator interfaces, 2–14
  - programming, 1–9
  - programming devices, 2–14
- Calculating Program Execution Time, 4–19
- Cascade control, 8–51
- Common terminals, 2–16
- Communication Port Specifications, 4–32
- Communication ports, 1–14
  - pinout diagrams, 4–4
- Communications problems, 9–7
- Comparative Boolean, 5–20
- Compare Double instruction, 5–62
- Compare instruction, 5–61
- Components, 1–7
- Connections
  - operator interface, 2–14
  - power input, 1–9
  - programming devices, 1–9, 2–14, 4–5
  - Solid State field devices, 2–17
  - toggle switches, 1–8
- Connector
  - common terminals, 2–16
  - diagram, 2–16

removal, 2–5

Control Output, 8–29

Control Relay Bit Map, 4–30

Converge Jump instruction, 7–23

Converge Stage instruction, 7–23

Convergence Stages, 7–19

Converting Number Formats

ASCII TO HEX, 5–86

Hex to ASCII, 5–87

Counter instruction, 5–36

Counter Instructions, 5–30

Counter Status Bit Map, 4–31

CPU

changing modes, 4–7

configuration, A–5

features, 4–2

hardware setup, 4–4

indicators, 9–6

instruction list, 5–2

Mode switch, 4–6

specifications, 4–3

status indicators, 4–6

CPU Indicator, 9–7

CPU Operation, 4–11

CPU Scan Time, 4–18

## D

Data Label instruction, 5–106, 5–108

Decrement Binary instruction, 5–71

Derivative term, 8–34

Diagnostics, 9–2

Dimensions, 2–6

DIN rail mounting, 2–9

Direct-acting loop, 8–33

DirectNET, 4–35

Divide Double instruction, 5–70

Divide instruction, 5–69

DL05 Micro PLC

front panel, 2–4

mounting guidelines, 2–6

unit dimensions, 2–6

Drum instruction, 6–2, 6–12

chart representation, 6–3

counter assignments, 6–6

drum control techniques, 6–10

EDRUM (Event Drum), 6–14

handheld programmer mnemonics, 6–16

overview of drum operation, 6–8

powerup state, 6–9

self-resetting, 6–11

step transition, 6–4

Drum sequencer programming, 1–12

## E

Emergency stop, 2–3

Enable Interrupt instruction, 5–104

Encode instruction, 5–77, 5–79, 5–81

End Instruction, 5–4

Environmental specifications, 2–8

Equal relays, 3–9, D–3, D–5

Error codes

code locations, 9–3

listing, B–2–B–9

pulse output errors, 3–43

Error term, 8–30

European Directives, F–2

Exclusive Or Double instruction, 5–60

Exclusive Or instruction, 5–59

Execution Times, C–3

## F

Fatal Errors, 9–2

Fault instruction, 5–107

Feed forward control, 8–47

For instruction, 5–97

Fuse protection, 2–10

## G

Goto Subroutine instruction, 5–99

Gray Code instruction, 5–89

## H

Handheld programmer, A-6  
 EEPROM operations, A-7  
 HEX TO ASCII instructions, 5-87  
 High-speed I/O  
   configuration, 3-5  
   discrete inputs with filter, 3-53  
   features, 3-2  
   high-speed counter, 3-6  
   high-speed interrupts, 3-45  
   I/O Point Usage, 3-4  
   modes, 3-4  
   programming, 3-11, 3-22, 3-41  
   pulse catch input, 3-50  
   pulse output, 3-25  
 Home search profile, 3-38

## I

I/O Response Time, 4-15  
 I/O Type Selection, 1-6  
 Increment Binary instruction, 5-71  
 Initial Stage instruction, 7-22  
 Initial Stages, 7-5  
 Instructions  
   accumulator / stack Load, 5-43  
   bit operation, 5-78  
   boolean, 5-9  
   capable of run time edits, 9-14  
   comparative boolean, 5-20  
   drum, 6-2, 6-12  
   execution times, C-2, C-3, C-6  
   immediate, 5-26  
   interrupt, 5-104  
   list of, 5-2  
   logical, 5-55  
   math, 5-63  
   message, 5-107  
   network, 5-114  
   number conversion, 5-83  
   program control, 5-97  
   stage, 7-21  
   stage programming, 7-2  
   table, 5-92  
   timer, counter, and shift register, 5-30  
 Integral term, 8-34  
 Interrupt Instructions, 5-104  
 Interrupt Return Conditional instruction, 5-104

Interrupt Return instruction, 5-104  
 Interrupts  
   external, 3-47  
   HSIO input, 3-45  
   timed, 3-47  
 Invert instruction, 5-85

## J

Jump instruction, 7-7, 7-22

## L

Load Address instruction, 5-51  
 Load Double instruction, 5-49  
 Load Formatted instruction, 5-50  
 Load instruction, 5-48  
 Load Label instruction, 5-91, 5-93  
 Logical Instructions, 5-55

## M

Maintenance, 9-2  
 Manual, organization, 1-4  
 Master Line Reset instruction, 5-102  
 Master Line Set instruction, 5-102  
 Math Instructions, 5-63  
 Memory  
   EEPROM, 1-13  
   FLASH, 1-13  
 Memory Map, 4-22, 4-28  
 Message Instructions, 5-107  
 MODBUS, 4-34, 4-41  
 Mode switch, 4-6  
 Modes  
   at power-up, 4-7  
   changing, 4-7  
 Motion control profile, 3-25  
 Mounting guidelines, 2-6  
   DIN rail, 2-9  
 Move instruction, 5-92  
 Move Memory Cartridge instruction, 5-91, 5-93

Multiply Binary instruction, 5–76  
Multiply Double instruction, 5–68  
Multiply instruction, 5–67

## N

Network Configuration and Connections, 4–32  
Network Diagrams, 4–32, 4–33  
Network Instructions, 5–114  
Network master operation, 4–41  
Network slave operation, 4–36  
Next instruction, 5–97  
Non–Fatal Errors, 9–2  
Normally Closed Contact, 5–4  
Not instruction, 5–14  
Not Jump instruction, 7–22  
Number Conversion Instructions, 5–83  
Numbering Systems, 4–20  
Numerical Constant instruction, 5–106, 5–108

## O

On/Off control, 8–49  
One shot, 5–14  
Or Double instruction, 5–58  
Or If Equal instruction, 5–21  
Or If Not Equal instruction, 5–21  
Or Immediate instruction, 5–26  
Or instruction, 5–10, 5–57  
    comparative, 5–24  
Or Negative Differential instruction, 5–16  
Or Not Immediate instruction, 5–26  
Or Not instruction, 5–10  
    comparative, 5–24  
Or Out Immediate instruction, 5–28  
Or Out instruction, 5–13  
Or Positive Differential instruction, 5–16  
Or Store instruction, 5–12  
Out Double instruction, 5–52  
Out Formatted instruction, 5–53

Out Immediate instruction, 5–28  
Out instruction, 5–13, 5–52  
Output Data Instructions, 5–43

## P

Panel layout, 2–7  
Part Numbers, 1–5, 1–6  
Password, 4–10, A–8  
Pause instruction, 5–19  
PID Loops  
    Alarms, process, 8–53  
    algorithms, 8–31  
    basic operation, 8–19  
    bibliography, 8–63  
    cascade control, 8–51  
    data configuration, 8–26  
    features, 8–2  
    feed forward control, 8–47  
    On/Off control, 8–49  
    Ramp/Soak generator, 8–57  
    sample rate, 8–13  
    scheduling, 8–13  
    setup parameters, 8–6  
    terminology, 8–4, 8–64  
    troubleshooting tips, 8–62  
    tuning procedure, 8–38  
PID loops, auto tuning, 8–38  
PLC Numbering Systems, 4–20  
Pop instruction, 5–53  
Position algorithm, 8–31  
Positive Differential instruction, 5–14  
Power indicator, 9–6  
Power wiring, 1–9  
Presets, 3–8  
    calculating values, 3–10  
    starting location, 3–9  
Print Message instruction, 5–110  
Process control, 8–17  
Product Weight Table, E–2  
Profiles  
    home search, 3–38  
    motion control, 3–25  
    registration, 3–30, 3–35  
    trapezoidal, 3–30, 3–32

velocity, 3–30, 3–40

Program Control Instructions, 5–97

Program Execution Time, 4–19

Program Mode, 4–12

Programming, concepts, 1–12

Programming Methods, 1–5  
examples, 1–10

Proportional term, 8–34

## Q

Quick Start, 1–7

## R

Ramp/soak generator, 8–57

Read from Network instruction, 5–114

Registration profile, 3–30, 3–35

Relay wiring, 2–19  
prolonging contact life, 2–21

Reset Immediate instruction, 5–29

Reset instruction, 5–18

Retentive Memory Ranges, 4–9

Reverse-acting loop, 8–33

Run Indicator, 9–7

Run Mode, 4–12

Run time edits, 9–14

## S

Safety guidelines, 2–2

Scratchpad memory, 1–10

Set Immediate instruction, 5–29

Set instruction, 5–18

Setpoint, 8–27

Shift Register instruction, 5–42

Shift Register Instructions, 5–30

Shift Right instruction, 5–80

Shuffle Digits instruction, 5–90

Sinking / sourcing I/O, 2–15

Special relays, 3–8, D–2

corresponding to error codes, 9–3

Specifications

CPU, 4–3

D0–05AA, 2–34

D0–05AD, 2–30

D0–05AR, 2–26

D0–05DA, 2–36

D0–05DD, 2–32

D0–05DD–D, 2–40

D0–05DR, 2–28

D0–05DR–D, 2–38

Discrete option modules, 2–42

environmental, 2–8

input power, 2–11

motion profiles, 3–28

Stage Control / Status Bit Map, 4–29

Stage Counter instruction, 5–38, 7–16

Stage instructions, 7–21

Stage programming, 1–12, 7–2

convergence, 7–19

emergency stop, 7–14

four steps to writing a stage program, 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, 7–25

stage instruction characteristics, 7–6

stage view, 7–18

state transition diagrams, 7–3

supervisor process, 7–17

timer inside stage, 7–13

Standard RLL Programming, 1–12

Status Indicators, 4–6, 9–6

Store If Equal instruction, 5–20

Store If Not Equal instruction, 5–20

Store immediate instruction, 5–26

Store instruction, 5–9

comparative, 5–23

Store Negative Differential instruction, 5–15

Store Not Immediate instruction, 5–26

Store Not instruction, 5–9

comparative, 5–23

Store Positive Differential instruction, 5–15  
Subroutine Return Conditional instruction, 5–99  
Subroutine Return instruction, 5–99  
Subtract Binary instruction, 5–73, 5–75  
Subtract double instruction, 5–66  
Subtract instruction, 5–65  
Sum instruction, 5–78  
System design, 1–11  
System V–memory, 4–26

## T

Table Instructions, 5–92  
Technical support, 1–2  
Time-proportioning control, 8–49  
Timer Fast instruction, 5–31  
Timer instruction, 5–31  
Timer Instructions, 5–30  
Timer Status Bit Map, 4–31  
Trapezoidal profile, 3–30, 3–32  
Troubleshooting, 9–2  
    communications, 9–7  
    electrical noise, 9–10  
    error codes, B–2  
    I/O points, 9–8  
    program debug, 9–11

Troubleshooting guide  
    HSIO Mode 20, 3–24  
    HSIO Mode 30, 3–43

## U

Up Down Counter instruction, 5–40

## V

V–memory, 4–26  
Velocity algorithm, 8–31  
Velocity profile, 3–30, 3–40

## W

Web site, 1–2  
Wiring  
    counter input, 3–7  
    counter outputs, 3–7  
    DC inputs, 2–22  
    DC Outputs, 2–23  
    drive inputs, 3–27  
    emergency stop, 2–3  
    encoder, 3–3  
    fuse protection, 2–12  
    High Speed I/O, 2–24  
    power input, 1–9, 2–10, 2–11  
    pulse output, 3–27

Informações sobre programação

[www.soliton.com.br](http://www.soliton.com.br) - e-mail: [soliton@soliton.com.br](mailto:soliton@soliton.com.br)

**SOLITON CONTROLES INDUSTRIAIS LTDA**

Rua Alfredo Pujol, 1010 - Santana - São Paulo - SP.

Tel:11 - 6950-1834 / Fax: 11 - 6979-8980 - e-mail: [vendas@soliton.com.br](mailto:vendas@soliton.com.br)