

# mitsubishi

PROGRAMMABLE CONTROLLERS

# MELSEC-FX

---

FX-Series PLC Training Manual using GX-Developer

---

FX1S  
FX1N  
FX2N  
FX2NC  
FX3U

***TRAINING MANUAL***

---

R18-0211-SLSASG-001-F

 **mitsubishi electric**



## Revisions

\* The manual number is noted at the lower left corner of the front cover.

Print Date	Manual Number	Revision
12/8/00		Rev * - Created new manual, internal only
1/30/01		Rev A - First Release Manual
11/8/02		Rev B – Updated manual to reference GX-Developer instead of GPP-WIN. Added Section 14.5 – Finding devices. Expanded Section 15.1 to discuss how to download comments. Corrected section 15.4 from Device Label to Alias
6/1/05		Rev C – Updated references to control products. Enhanced project ladder in Appendix. Added new updated screen captures.
2/25/06		Rev D – Updated to include FX3U. Section 2 removed references to obsolete PLCs, updated to include FX3U and new modules. Reordered Lessons 4 to 9. Lesson 8 now based on FX3U PLC trainer. Expanded coverage of high speed counter usage in Lesson 13.
3/24/06		Rev E – Corrected typos. Moved Lesson 2 power supplies to after discussion of module types.
6/16/06		Rev F – Corrected Project 4 code in Appendix. Corrected typos. Added programming exercises. Added basic ladder logic section in Lesson 10. Added ALT to Lesson 10, TTMR and HOUR to Lesson 12, RTC commands to Lesson 14. Replaced BIN and BCD with FLT and INT in Lesson 13. Added shift registers to Lesson 13.

Disclaimer: This manual does not imply guarantee or implementation right for industrial ownership or implementation of other rights. Mitsubishi Electric Corporation is not responsible for industrial ownership problems caused by use of the contents of this manual.



## Introduction

Thank you for purchasing the Mitsubishi FX-Series Programmable Logic Controller. Before using the equipment, please read this manual carefully to develop full familiarity with the functions and performance of the control you have purchased, so as to ensure correct use.

## Table of Contents

### LESSON 1 – Introduction and Overview

1.1	Course Objectives .....	1
1.2	Course Prerequisites .....	1
1.3	Course Duration.....	1
1.4	Course Description .....	2
1.5	Product Line Overview.....	3

### LESSON 2 – FX-Series Hardware Review

2.1	What is a dedicated PLC? .....	5
2.2	FX Line of PLCs.....	6
2.3	Hardware Components .....	8
2.4	Inputs.....	10
2.5	Outputs .....	12
2.6	Special Function Modules.....	14
2.7	High Speed Counter & Positioning Modules .....	15
2.8	Communication Modules and Option Boards.....	17
2.9	Network Modules & Option Boards .....	19
2.10	Miscellaneous .....	21
2.11	Power Supplies.....	21
2.12	Exercise – <i>Power Supply Calculation</i> .....	29
2.13	Memory Types .....	30

### LESSON 3 – Programming Equipment

3.1	Hand-Held Programming Units .....	33
3.2	Programming Software .....	33
3.3	GX-Developer Overview .....	34
3.4	File Format .....	37
3.5	Hardware Connection.....	37

### LESSON 4 – Number Systems

4.1	Binary Numbers .....	39
4.2	Hexadecimal Numbers.....	40
4.3	Octal Numbers.....	41
4.4	Binary Coded Decimal .....	42
4.5	Exercise – <i>Number Systems Conversion</i> .....	43

## **LESSON 5 – Numeric Data in PLCs**

5.1	Integer (16/32 Bit) .....	45
5.2	Decimal (16/23 Bit) .....	47

## **LESSON 6 – System Devices**

6.	System Devices .....	49
----	----------------------	----

## **LESSON 7 – Addressing**

7.1	Right Side Bus Rules of Addressing .....	53
7.2	FX3U Left Side Bus Addressing .....	53
7.3	Example .....	55
7.4	Exercise – <i>PLC Addressing</i> .....	56

## **LESSON 8 – Demo Kit Layout**

8.1	Addressing .....	55
8.2	Indicator Lights .....	56
8.3	Operator Interface .....	56

## **LESSON 9 – PLC Instruction Types**

9.1	Basic Instructions .....	59
9.2	STL (Step Ladder) Instructions .....	59
9.3	Applied Instructions .....	59

## **LESSON 10 – Basic Instructions**

10.1	Symbols .....	61
10.2	Ladder Basics .....	62
10.3	Common Instructions .....	63
10.4	Exercise – <i>Ladder Basics</i> .....	65

## **LESSON 11 – Develop and Edit Programs**

11.1	Launching GX-Developer .....	67
11.2	Creating a New Project .....	68
11.3	Editing the Ladder .....	69
11.4	Program Transfer .....	70
11.5	Online Editing .....	71
11.6	Monitor the Program Operation .....	72
11.7	Forcing Bits and Changing Registers .....	73
11.8	Exercise – <i>Contacts and Coils</i> .....	73

## LESSON 12 – Timers and Counters

12.1	Timers.....	75
12.2	Counters .....	76
12.3	Program Examples .....	79
12.4	Additional Timer Commands.....	81
12.5	Exercise – <i>Timers and Counters</i> .....	82
12.6	Exercise – <i>Conveyor Control</i> .....	82

## LESSON 13 – Applied Instructions

13.1	General Format.....	83
13.2	Data Transfer Instructions.....	84
13.3	Comparison Instructions .....	85
13.4	Exercise – <i>Parking Lot</i> .....	88
13.5	Exercise – <i>Conveyor Control Part 2</i> .....	88
13.6	Conversion Instructions .....	88
13.7	Increment and Decrement Instructions .....	89
13.8	Exercise – <i>INC and DEC</i> .....	89
13.9	Arithmetic Instructions.....	89
13.10	Exercise – <i>Binary Math</i> .....	90
13.11	Exercise – <i>Parking Lot Part 2</i> .....	90
13.12	Exercise – <i>Conveyor Control Part 3</i> .....	90
13.13	High-Speed Processing .....	90
13.14	TO/FROM Instructions .....	92
13.15	Exercise – <i>FX2N-5A Module Access</i> .....	93
13.16	Shift Registers .....	93
13.17	Exercise – <i>Bit Shift Register</i> .....	94
13.18	Program Flow Control.....	95

## LESSON 14 – Diagnostic Devices

14.1	Special M Relays .....	97
14.2	Special D Registers .....	98
14.3	Handy Troubleshooting Circuits.....	98
14.4	Real Time Clock Usage .....	99
14.5	Exercise – <i>Daylight Savings Time</i> .....	100
14.6	GX-Developer Diagnostics.....	101
14.7	Find/Replace Menu.....	102
14.8	Data Trace .....	107

<b>LESSON 15 – Documentation and Printing</b>
-----------------------------------------------

15.1	Comments .....	109
15.2	Statements.....	110
15.3	Notes .....	111
15.4	Device Labels .....	111
15.5	Viewing Documentation .....	111
15.6	Printing.....	112

<b>Appendix</b>	<b>p. 115</b>
-----------------	---------------



# LESSON 1 – Introduction & Overview

Welcome to the FX-Series programmers training course. This course is intended for designers and control engineers, responsible for developing application programs using the FX-Series controller. Apart from traditional product training in which the exclusive focus is on tools, a portion of this training is devoted to design with the intention of laying a foundation for a successful and short design and debugging cycle.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Identify the objectives of this course.
- ✓ Identify the objectives of each lesson.
- ✓ List the prerequisites and target audience of the course.

**Materials:** FX-Series PLC Training Manual

**Overview:** This lesson is an introduction to the course and its organization. This brief beginning is designed to give students a quick listing of the lessons, their sequence and their contents. An early opportunity is given to students to express opinions and needs regarding topics and issues that may not be covered in the courseware.

## **1.1 Course Objectives**

At the conclusion of this course, **you will be able to...**

- ✓ Identify the FX-Series hardware variations available, and understand their intended application
- ✓ Use the software tool GX-Developer, in conjunction with the FX-Series trainer, to develop, test, debug, and implement a symbolic ladder program for a machine control application.
- ✓ Understand the structure and basic operation our SFM (special function modules) used for operations such as AtoD and DtoA conversion, Networking, and High-Speed counters.

## **1.2 Course Prerequisites**

Students attending this class should have knowledge of basic electronics, and some exposure to industrial control concepts. Experience with PLC ladder or any computer programming language is also advantageous.

## **1.3 Course Duration**

This is a **3 day** training course.

## **1.4 Course Description**

### **LESSON 1 - Introduction to the Course**

This is a brief introduction to the course, and a breakdown of the lesson topics. It is also an opportunity for students to comment on what will and will not be covered.

### **LESSON 2 - Hardware Review**

This lesson discusses the hardware structure of the FX-Series logic controller, CPU module types and capabilities, Input and Output module types and characteristics.

### **LESSON 3 - Programming Equipment**

This lesson covers the hardware, software, and connections necessary to connect a laptop to a PLC. Alternatives to laptop programming are also covered.

### **LESSON 4 - Number Systems**

This lesson discusses the 4 different numbering systems used by our PLC systems: Binary, Octal, Hexadecimal, and BCD Binary Coded Decimal.

### **LESSON 5 - Numeric Data in PLCs**

This lesson explains how integers and decimals are manipulated in a PLC program.

### **LESSON 6 - System Devices**

This lesson covers the devices, such as X input devices, Y output devices, and M relays, that are used in the program instructions.

### **LESSON 7 – Addressing**

The rules of addressing, including limitations on the maximum number of I/O are discussed in this lesson

### **LESSON 8 - Demo Kit Layout**

The hardware kit that will run the programs written in this class is examined and explained.

### **LESSON 9 - PLC Instruction Types**

This is a discussion on the 3 types of FX programming instructions and their purposes.

### **LESSON 10 - Basic Instructions**

The contacts, coils and other basic building blocks are explained.

### **LESSON 11 - Developing and Editing Program**

This lesson reviews the process of starting a project, writing a simple project, transferring the project, and monitoring.

### **LESSON 12 - Timers and Counters**

This lesson reviews these 2 important devices and directs the student to write a program utilizing the timers.

### **LESSON 13 - Applied Instructions**

This lesson covers all special processing instructions: data manipulation instructions such as MOV, arithmetic instructions, comparison instructions, conversion instructions, logical operations, and TO/FROM instructions

**LESSON 14 - Diagnostic Devices** – The special relays and registers that can assist in troubleshooting and writing programs are discussed here. A brief discussion of the diagnostic capabilities of GX-Developer and some sample diagnostic ladder logic code is presented as well.

### **LESSON 15 - Documentation and Printing**

This lesson reviews the types of documentation that can be included in a program, as well as the different options available for printing information about a program.

## **1.5 Product Line Overview**

Mitsubishi offers **Modular** and **Micro** style controllers. A couple of the current models are shown below.

### ***Q-SERIES***



### ***FX-SERIES***

The FX-Series PLC will be covered in this class. There are many programming similarities between the FX Series and the rack based controllers. It incorporates inputs, outputs and power supply into one package!



# LESSON 2 – FX Series Hardware Review

This lesson discusses the hardware structure of the FX-Series logic controller. This includes a review of the different CPU module types, input/output modules and interfacing variations.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Explain the general sections of a PLC.
- ✓ Describe the different models of the FX family.
- ✓ Describe the characteristics of the I/O modules available.
- ✓ List some of the factors to be considered in specifying hardware.

**Materials:** FX-Series PLC Training Manual  
FX Family Brochure

## 2.1 What is a dedicated PLC?

A dedicated PLC is a microprocessor controlled computer that is designed specifically to **perform real-time industrial machine control**.

There are 3 general sections to all PLC's, **INPUT**, **CONTROLLER**, and **OUTPUT**.

- ❖ **INPUT** - The input section consists of integrated inputs, or extension inputs that machine input devices will be wired to, like limit switches, or transistor sensors. When the input voltage reaches its specified level the input becomes active. Once active, the input can be read by controller.
- ❖ **CONTROLLER** - The controller is the main unit. Several CPU modules are available depending on the demands of your application.
- ❖ **OUTPUTS** - The third section is the outputs. Based on the condition of the inputs, the controller will judge which outputs should turn on, to activate machine devices like lights, buzzers, relays, solenoids, or motors.

The 3 sections are controlled by custom sequencing software called ladder programming, which takes the place of hardwired circuits. The relationship between inputs and outputs is controlled by the logic in your ladder program. Because hard-wired circuits are now replaced by software logic, machine modifications and improvements, are much easier to complete.

## **2.2 The FX Line of PLCs**

Each of the PLCs in this family have certain characteristics in common:

- 1) An integrated power supply. Most of the PLCs in this line have a built in power supply that requires 100-240 VAC power. Several of the PLCs have a DC power version available as well.
- 2) Integrated I/O. The main unit has a varying number of inputs and outputs, dependent on the model chosen. The inputs are typically DC, although certain models have AC inputs as well. All models have relay and transistor output versions (except the FX0S), and several offer triac outputs.
- 3) Common Instructions. Although the higher-end PLCs have more instructions, all PLCs at least support a common instruction set of 20 basic and 35 applied instructions.
- 4) Built-in special functions. All FX PLCs have a built-in high speed counter (or several high speed counters) and a pulse train output.

### **The FX Series Legacy**

The F Series of PLCs was originally introduced in 1981 with the F PLC. The F was then improved and revised several times, resulting in the F1, F1J and F2 models. After the F2 came the FX Series PLCs. Some of these models were the FX, FX0, FX1, FX2, FX0S, FX0N, and FX2C, as well as the current offerings FX1S, FX1N, FX2N, FX2NC, and FX3U. The current families of FX Series PLCs are detailed below.

Sales of the Mitsubishi F Series PLCs have exceeded 6 million units worldwide!

### **FX1S**

This PLC model is one Mitsubishi's most advanced models. It has the small footprint like the FX0S, but far more capability. It has more I/O (up to 30), more internal devices, and has motion control capability. A small HMI (the FX1N-5DM) can be connected to the front and used to monitor and change timers, counters and data registers. This unit can use one expansion board, but has no expansion bus.

## **FX1N**

This model of PLC is one of Mitsubishi's most advanced. It provides the midrange number of I/O points similar to the FX0N, but has far more capability. It has more internal devices (like 1536 M relays and 235 counters) than the FX0N, and has motion control capability. Unlike the FX0N, the FX1N has a number of option boards that can be added to provide additional ports or allow the connection of FX0N communication modules. The FX1N-5DM can be connected to the FX1N as well. This CPU does have the expansion bus capability, but is limited to two expansion modules, which can be either discrete I/O extensions or special function modules.

## **FX2N**

This PLC type is currently one of Mitsubishi's most powerful processors. It is expandable and can control up to 256 I/O. It has 3072 internal relays, 256 timers, 234 counters, 8000 data registers, and up to 21 high-speed counters. All the special function modules available for the enhanced FX are available with this line, as well as several new modules. Modules for Profibus, CC-link, AS-Interface and I/O link are available, as well as an electronic cam switch module. Besides special function modules, the capabilities of this versatile PLC can be extended by small expansion boards that can be connected to the front of the PLC. These can give the user a second programming port, an RS-485 port, or RS-232 port. FX0N and FX modules can be used with this PLC.

## **FX2NC**

This PLC is to the FX2N as the FX2C is to the FX. Similar to the FX2N in all respects, save for no built-in real-time clock, this PLC uses distributed I/O. The result is a PLC main unit that is the size of a cigarette pack. Expansion boards cannot be used, but the full line of special function modules can still be connected.

## **FX3U**

The FX3U is Mitsubishi's latest and most powerful processor. It is highly expandable and can control up to a maximum of 384 I/O. It has more internal memory, a faster CPU, and a new left side expansion bus. All the special function modules available for the enhanced FX and FX2N are available with this line, as well as several new modules. New networking options include a full featured Ethernet module and a Profibus Master module. New communication boards and adapters, including a new USB expansion board, can be used to reach a maximum of 3 serial connections on this PLC. This processor comes with 64,000 steps of program memory.

This training course will make use of the FX3U series PLC mainly, however most of what is covered will apply to the entire FX series PLC line.

## 2.3 Hardware Components

### Main Units



FX3U  
(FX2N is very similar in appearance)



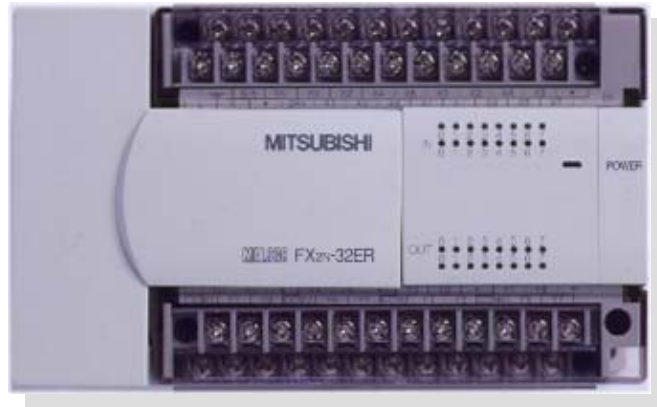
FX2NC Base Unit

**The main unit contains the CPU that provides the processing power that reads the inputs, solves the logic, and writes to the outputs. The main unit contains:**

- 1) An integrated power supply that provides power to the CPU, inputs, and a limited number of connected expansion or special function blocks
- 2) Integrated inputs. These can be either AC or DC, depending on the model selected. The largest FX3U main unit has 64 inputs.
- 3) Integrated outputs. These can be relays, transistors or triacs. The largest FX3U unit has 64 outputs. In the FX1S and FX1N the ratio of inputs to outputs is 3 / 2 or 4 / 3. In the FX2N and FX3U line the ratio is 1 / 1.
- 4) Programming port. This port uses RS-422 as its communication protocol. The PLC can be programmed through this port, or an HMI (Human-Machine Interface) can be connected as well.
- 5) Accessory connection ports. These can be used to connect memory module or an option board to the main unit.

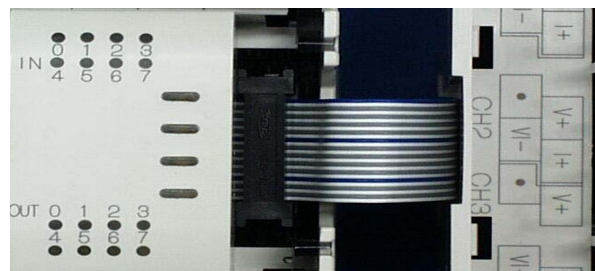


## Powered Extension Units



As was mentioned previously, the FX3U is very expandable. One way to increase I/O is through the use of powered expansion units. These units have:

- 1) An integrated power supply, so as not to task the power supply of the main unit. AC and DC are both available.
- 2) Integrated inputs. 24 VDC and 120 VAC units are available. The number of inputs is either 16 or 24, depending on the model selected.
- 3) Integrated outputs. The number of outputs is the same as the number of inputs (16 or 24). All three types of outputs (relay, transistor, triac) are available.



**Connecting I/O to a main unit**

## Unpowered Extension Blocks

The second way to expand the I/O of the FX3U system is through the use of unpowered extension blocks. These do not have a power supply, so they draw upon the power supply of the main unit or powered extension unit. As such, there is a limit (short of the I/O allocation limit for FX3Us) to the number of extension blocks that can be added. How to calculate this number will be covered later.

Unpowered extension blocks for the FX3U provide 16 I/O points. These points are either inputs or outputs, but not both. Inputs are 24 VDC, and all three output types are available. If the available models don't meet the needs of the system, units from the FX0N PLC line can be used, as well as from the FX PLC line (through the use of the FX2N-CNV-IF converter).



**Note:** Be aware of the terminology used. An **extension unit** is **powered**, while an **extension block** is **unpowered**.

## 2.4 Inputs

Input devices are the interface between PLC and machine. The base unit of an FX PLC has a number of integrated inputs available. If more are needed, input extension blocks and units are available to be connected.

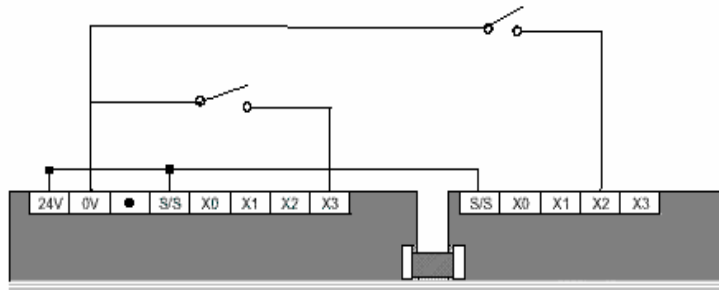
There are **2 different input types** ...

- DC Inputs
  - Fast response
  - 90% of new designs use this type of input
  - mostly 24VDC, but FX1N has 12VDC option
  - sink or source logic
- AC Inputs
  - Slow response
  - Easy to interface AC devices
  - All AC inputs are 120VAC



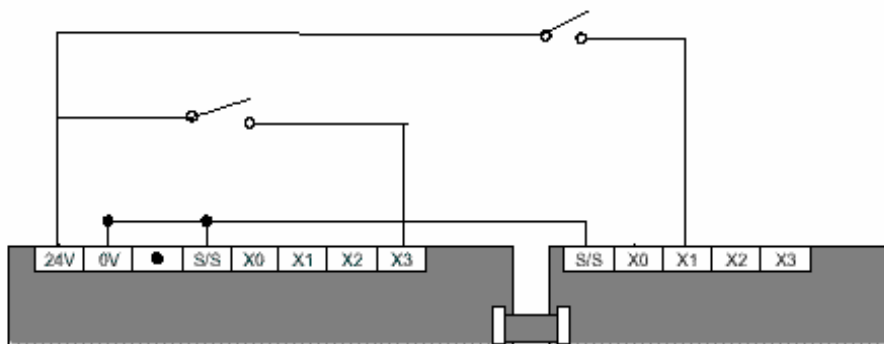
**SINK or SOURCE** logic refers to the voltage level that will cause the input to become active.

**SINK logic**: the input becomes active when connected to **GROUND**.  
The S/S pin is tied to +24V.



Use SINK for **NPN SENSORS**

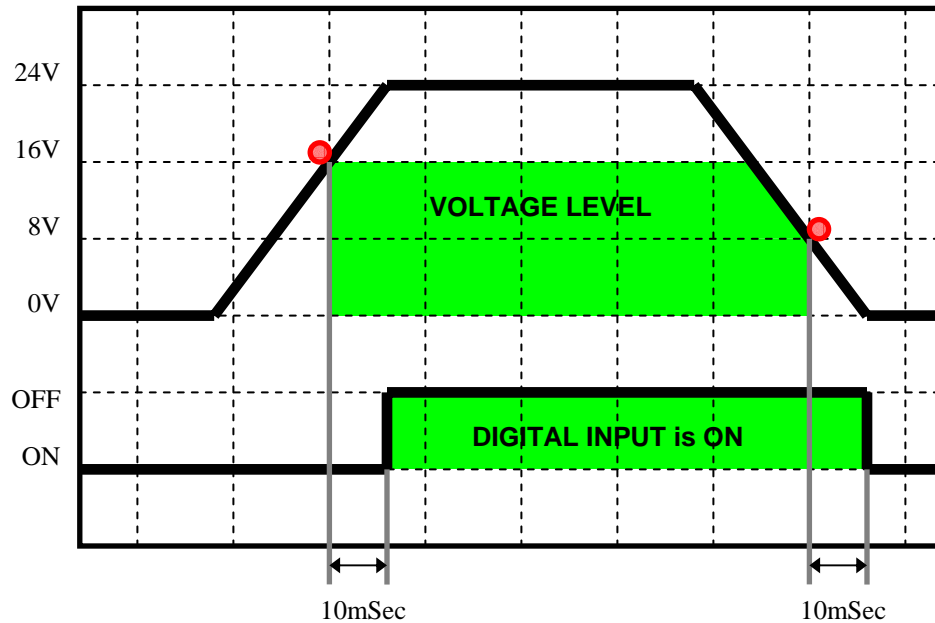
**SOURCE logic**: the input becomes active when connected to **+24VDC**  
The S/S pin is tied to GROUND



Use SOURCE for **PNP SENSORS**

The input trigger levels vary, depending on the module type. Generally, the input becomes active at the 2/3 level, then the input becomes inactive at the 1/3 level.

For example, a +24VDC input will become active when the input voltage reaches 16VDC, and the input turns off when the voltage reaches 8VDC.



The standard input point has a **10 millisecond input filter**. This is done deliberately for switch de-bounce. As a limit switch contacts close, the mechanical contacts actually bounce a little bit, causing the input to turn on and off rapidly for a short period of time. This 10msec delay allows the switch contacts time to stop bouncing.

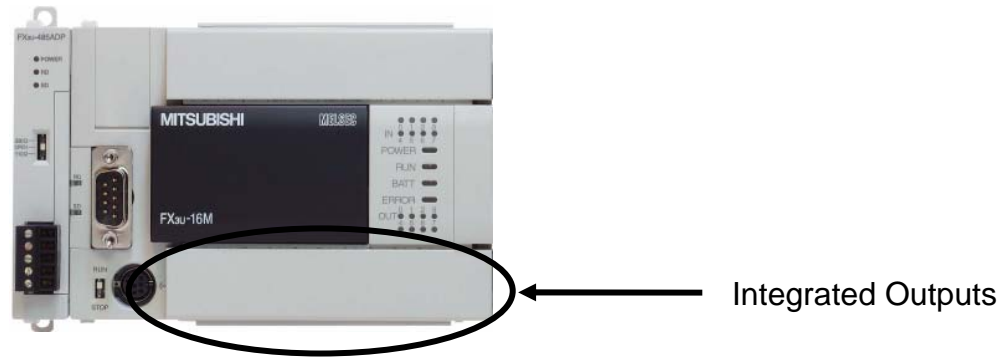
It is possible to adjust the input filter constant for the first 8 inputs. This is explained in Lesson 14.

Typically if you need to see more than 20 pulses per second, a high speed counter module or the built-in high speed inputs should be used.

Your base PLC choice will consist of 8, 16, 24, 32, 40, or 64 input points. Extension units offer either 16 or 24 inputs. Extension blocks offer 8 or 16 points. All modules are available in sink or source logic.

## 2.5 Outputs

The output device allows the PLC to control a machine. Integrated outputs are available on the base unit. If more outputs are required, extension blocks and units can extend the number of I/O.



There are 3 different output types ...

- Relay Outputs
- Triac Outputs
- Transistor Outputs

### **RELAY OUTPUTS**

Relays are dry contacts, so whatever you input on the common is switched out when the output is activated. This is the most common type of output module used. Loads up to 2 Amps, 100VAC~240VAC or 30VDC can be switched, with a maximum of 8 Amps per Common. Most base units and extension modules have 4 outputs per common.

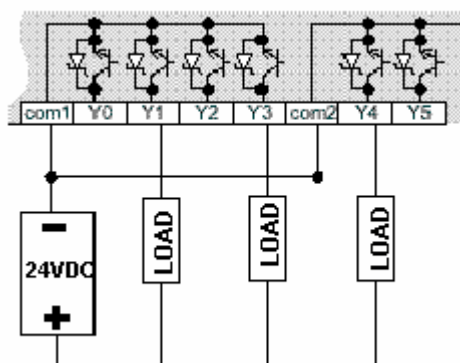
### **TRIAC OUTPUTS**

Triacs are solid state AC switches. When the output is active the module connects the load to the AC source. Load switching is up to 0.3Amps, up 240 VAC per point, with a max of 0.8 Amps per Common. Each base unit or extension module has up to 4 outputs per common.

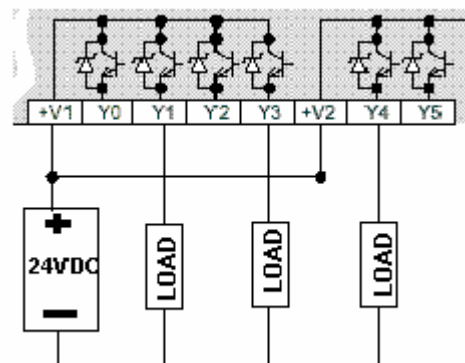
### **TRANSISTOR OUTPUTS**

Fast response is the main characteristic. Transistors are solid state DC switches. When the output is active, the module connects the load to the DC source. Load switching is up to 0.5Amps per point, up to 0.8 Amps per Common. Voltage that can be switched is 5VDC- 30 VDC. 4 points per common is typical.

#### **Sinking Outputs**



#### **Sourcing Outputs**



## 2.6 Special Function Modules

All of the modules that have been discussed thus far have been DISCRETE I/O modules. The inputs or outputs are either ON or OFF. This is acceptable if all the inputs in the PLC system are switches or simple sensors, and the outputs are or lights to turn off and on. If it is necessary to monitor or control a temperature, talk to a network, or control a positioning module, it is necessary to use a Special Function Module (referred to as SFM) to accomplish these tasks. Each of these modules has a range of internal addresses where the data is stored. The PLC has dedicated commands to access this data.

### Right Side Bus Analog Modules

There are 3 types: Analog Input modules, Analog Output modules and Combination Analog Input/Output modules. All are used with I/O points that have more states than just on or off. Examples of analog inputs would be a velocity reading or pressure reading. An example of an analog output would be the variable speed of a motor.

The input cards come in 2, 4, or 8 channels. These are the **FX2N-2AD**, **FX2N-4AD**, and **FX2N-8AD**. The output cards come in either 2 or 4 channels. These are the **FX2N-2DA** and **FX2N-4DA**. There are 2 combination cards. The **FX0N-3A** has 2 input channels and 1 output channel. The **FX2N-5A** has 4 input channels and 1 output channel. The FX3U family also has two dedicated analog modules, the **FX3U-4AD** and **FX3U-4DA**, which are higher resolution and faster processing modules.

All are based on varying current or voltage, usually  $-20\text{ mA}$  to  $+20\text{ mA}$ ,  $4-20\text{ mA}$  or  $-10\text{ V}$  to  $+10\text{ V}$ , as set by the programmer. Depending on the card, the module receives from the PLC or from the input, a raw number (range varies by module and type, see manuals for ranges), that is interpreted as the analog value read (if an analog input) or the analog value to be sent out (if an analog output)

### Programming Example

The PLC programmer wants to detect small changes in pressure to control a chemical mixing process. He has a sensor that has a range of 0 PSI to 300 PSI and generates a voltage of  $-10\text{ V}$  to  $+10\text{ V}$ , and the FX2N-4AD module, which sees  $-10\text{ V}$  as the number -2000, and  $+10\text{ V}$  as +2000.

Given this information, the programmer knows that at 0 PSI the sensor sends a voltage of  $-10\text{ V}$ , which results in a value of -2000 being written to the PLC. At 150 PSI, 0V is generated, resulting in a value of 0 being written to the PLC. At 300 PSI, the voltage is 10V and the PLC value is 2000. It is the responsibility of the programmer to know what pressure value equates to what PLC value, and to write the program to scale that value accordingly.



## **Temperature Input Modules**

These modules are similar to analog input modules, except for the type of input devices which can be connected. These modules have 4 input channels. With the **FX2N-4AD-TC**, thermocouple inputs are used for the detection of temperature changes, and work off of minute changes in voltage. Type J and Type K thermocouples are supported. On the **FX2N-4AD-PT**, the inputs are special platinum temperature sensors (PT-100 RTD). These sensors can detect very small temperature changes (.2°C to .3°C .36°F to .54°F). The **FX2N-8AD** module's inputs can also be configured for type K, J, or T thermocouples.

## **FX3U Left Side Adapter Bus Analog Modules**

One of the new features of the FX3U is a new high speed expansion bus on the left hand side of the PLC. A maximum of 4 analog modules can be used on this left side expansion bus. There are 4 analog modules available for this left side expansion bus. These modules include a 4-channel input module, a 4-channel output module, a 4-channel thermocouple input module, and a 4 channel RTD input module. These left side expansion bus modules do not require TO/FROM instructions. The data from the analog inputs is mapped directly into data registers in the PLC.



## **2.7 High Speed Counter & Positioning Modules**

There are many options for the FX Series of PLCs to perform high speed pulse inputs and pulse output motion control. Ordinary counters in the PLC are dependent on the scan time in 2 ways: 1) the updating of the input that is used as the counting input and 2) the updating of the accumulated value of the counter. This may be too slow for high speed counting applications. The FX Series PLC has built-in high speed counters, but the maximum counting frequency varies by CPU family, and this value drops as more high-speed counters are added. Also, inputs are limited to 24 VDC.

### **High Speed Counter Module**

The **FX2N-1HC** module provides the high speed counting ability, up to 50kHz, and has selectable inputs of 5, 12 or 24 VDC. It allows for 2 single phase inputs or one 2-phase input. It also has two integrated transistor outputs which can be controlled independently of one another by internal compare instructions.



## **Single Axis Positioning Modules**

Also known as the pulse generator, the **FX2N-1PG** module creates a pulse train output that can be used for motion control applications. This module will accept 24VDC inputs. This module can put out a pulse train at speeds up to 100KHz. The FX2N-1PG attaches to the right side expansion bus.

The **FX2N-10PG** module allows for 5VDC or 24VDC inputs. It offers a maximum output speed of 1MHz. This module also connects to the right side expansion bus.

The characteristics of the pulse train, such as frequency, and its OFF/ON status can be controlled by the PLC program or by the programmer setting parameters prior to operation.



## **FX3U SSCNETIII Motion Control Module**



The **FX3U-20SSC-H** module provides high-end motion control capabilities to the FX3U series PLCs. It is designed to use the MR-J3-B servo line and its' fiber-optic SSCNETIII motion network. Amplifiers can be up to 50 meters away from the PLC via the fiber optic cables. This module will control up to 2 axes of motion, with linear and circular interpolation. This module connects to the right side expansion bus.

## **FX3U Adapter Bus High Speed Counter Module**

The **FX3U-4HSX-ADP** module connects to the FX3U's left side expansion bus, and provides high speed counting ability, up to 200kHz. The FX3U-4HSX-ADP module does not require TO/FROM instructions. All data transmissions to and from this module are directly into the PLC's memory. A maximum of 2 high-speed counter modules can be used on an FX3U CPU. These modules must be located directly next to the CPU on the left hand side.





## **FX3U Adapter Bus High Speed Pulse Output Module**

The **FX3U-2HSY-ADP** module connects to the FX3U's left side expansion bus, and provides high speed pulse output ability, up to 200KHz. The FX3U-2HSY-ADP module does not require TO/FROM instructions. All data transmissions to and from this module are directly into the PLC's memory. A maximum of 2 high-speed output modules can be used on an FX3U CPU. These modules must be located directly next to the CPU on the left hand side.



## **2.8 Communication Modules and Option Boards**

Many times, more is required of a PLC system than monitoring inputs and controlling output devices. Data may need to be passed to another PLC or even a PC. A programmer may need to access the program to monitor an error or make program changes, but the front port is occupied by an HMI.

The FX family has several modules that can be added to add communications ability and solve problems like those stated above. Sometimes the solution requires the implementation of a network to get the required connectivity. The modules used for this will be discussed in the next section. In this section modules used to simply augment the communications ability of the PLC will be discussed.

The FX3U family of PLCs supports the addition of 2 serial ports via the option boards and left side bus ADP modules. If using a BD board, only one ADP module can be used. When not using the BD board, two ADP modules can be connected.

### **RS-232 Communication Interface Module(s)**

There are three modules that can be used to add an RS-232 port to the FX3U PLC, the **FX3U-232ADP**, the **FX2N-232IF** and the **FX3U-232-BD**.

For the FX1S, FX1N, FX2N, and FX2NC, the **FX2NC-232ADP** will work. The FX\*N-CNVD is required for PLCs other than the FX2NC.



The FX3U-232ADP (pictured left) can be connected to the left side of the PLC through the left side expansion bus. This module requires special programming in the PLC to configure the port. Data is transmitted and received through the use of the RS Instruction.

The FX2N-232IF (pictured right) connects to the right of the PLC, in the same way that other special function modules connect. PLC programming is necessary to initialize the module and set its parameters. The module can be configured to automatically convert data between the ASCII that is received or transmitted, and the binary or BCD data

that is used in the PLC. The RS instruction is not used; instead the module is controlled by TO/FROM statements (to be explained later). This module does not count as one of the two additional serial ports mentioned above.



### **RS-232 Option Board – FX3U-232-BD**

This board connects to a special port that is located on the left side of the PLC. This method saves the special function module space that would be occupied by the FX2N-232IF, or the adapter space used by the FX3U-232ADP.

Like the 232ADP, this board requires special programming to configure the port. If using an open protocol, the RS instruction is required to transmit and receive data.

The BD board can be used for dedicated communications protocols, requiring only the setup of the D8120 register.



### **RS-422 Option Board – FX3U-422-BD**

The HMI lines carried by Mitsubishi commonly connect to the FX3U PLC through its programming port. If the programmer needs to interface with the PLC program without disconnecting the HMI, this board is the simplest method to accomplish this goal. The part is plug-and-play, and adds a second programming port to the PLC. Note that an HMI will interface with the PLC through this port as well. Some setup of the D8120 register may be required.



## **USB Option Board – FX3U-USB-BD**

With recent updates to personal computers, a computer with a serial port is becoming rarer. This option board will add a standard USB port to the front of the PLC, and will allow the PC to connect to the PLC via a USB port. No USB to serial adapter is required, and the required drivers for the PC and a cable come with the board.



## **2.9 Network Modules and Option Boards**

One of the assets of the FX-series PLC line is the networking capability that can be added. This allows the PLC to control a far larger system, or to be controlled as part of a far larger system, than could be controlled by the main and extension racks alone. Networks and network modules that are available:

### **RS-485 Option Board & ADP Module**



The **FX3U-485-BD** adapter board connects to a special port that is on the left side of the PLC. This card provides several networking options. The most powerful of these is the N:N (also known as peer-to-peer) network. This is a multi-drop station network that allows the connection of up to 8 stations on one network. This module is also useful for the built-in Parallel Link mode, where data is automatically shared between 2 PLCs.



The **FX3U-485ADP** module allows the same functionality, but with a stronger signal rated for longer distances.

For the FX1S, FX1N, FX2N, and FX2NC, the adapter module for an RS485 connection is **FX2NC-485ADP**, and it will require the use of an FX\*N-CNV-BD converter board to connect to the PLC (except for FX2NC, connector is built in).

### **AS-I Master Module – FX2N-32ASI-M**

AS-I is a low-cost electromechanical connection system designed to operate over a two-wire cable carrying data and power over a distance of up to 100m, or more if repeaters are used. It is especially suitable for lower levels of plant automation where simple - often binary - field devices such as switches need to interoperate in a stand-alone local area automation network controlled by PLC or PC. AS-Interface is best seen as a digital replacement for traditional cable tree architectures.

This module allows the FX2N, FX2NC, or FX3U to use the AS-I (often pronounced as “azzi”) network to control and monitor up to 31 field devices.

For more information on AS-I see [www.as-interface.com](http://www.as-interface.com)

### **I/O Link – FX2N-16LNK-M**

I/O link is actually a distributed I/O system for Mitsubishi FX series PLCs. I/O modules can be placed at up to 200m from the main rack. They are addressed like standard I/O and are subject to the limitations of the CPU. Each I/O link master module can control up to 128 I/O points on up to 16 stations.

### **CC-Link Master Module – FX2N-16CCL-M**

CC-Link is primarily a high powered remote I/O network, although it can be used to connect local stations as well. It uses twisted-pair cabling. CC-link has a range of 1200 m, data transmission rate of 10Mbps at 100m, fast update times (3.9mS for 64 stations) and high I/O capability (2048).

This module makes an FX1N, FX2N, FX2NC or FX3U PLC act as a master on a CC-Link network.

### **CC-Link Interface Module – FX2N-32CCL**

The FX2N-32CCL allows the FX2N PLC to connect to the CC-Link network as a remote device station. The CC-link master (usually an A-series PLC) controls the FX2N PLC by writing data to the buffer memory of the FX2N-32CCL, which then writes data to the PLC.

### **Profibus Master Module – FX3U-64DP-M**

This module works only with the FX3U series. It is a full-featured master module for a Profibus network. It is now possible for an FX3U to run the Profibus network. Profibus networking supports maximum speeds of 12MB, and distances up to 4800 meters.

Profibus is an open network protocol, which many vendors make devices for. For more info on Profibus, go to [www.profibus.org](http://www.profibus.org).

### **Profibus Interface Module – FX0N-32NT-DP**

The addition of this Profibus module allows the PLC to be connected to a Profibus network. Profibus DP is supported. The PLC acts as a Profibus slave. Another PLC will be required as the network master.

### **Profibus Interface Module – FX2N-32DP-IF**

The addition of a Profibus module allows the PLC to be connected to a Profibus network. Profibus DP is supported. The PLC acts as a Profibus slave.

The FX2N-32DP-IF actually replaces the CPU in an FX setup. This module allows FX I/O and SFM modules to be configured as a remote I/O drop on a Profibus system.

## **Ethernet Interface Modules – FX2NC-ENET-ADP & FX3U-ENET**

Ethernet is the industry standard for computer-to-computer networking. Ethernet is beginning to make its way to factory floor devices.

The **FX2NC-ENET-ADP** can be connected to the FX1S, FX1N, FX2N, and FX2NC PLCs. All but the FX2NC will require the addition of the FX1N-CNV-BD or FX2N-CNV-BD. This is a simple serial to Ethernet gateway. GX-Developer v8.25 and later have support for the Ethernet connection to the FX Series PLCs via this module. Prior versions of the software required a software package called Serial-IP to simulate a COM port connection for an IP address.

The **FX3U-ENET** works with the FX3U series PLCs. It is a full-featured Ethernet module. The FX3U supports up to 8 active connections to the PLC. You can program the PLC via GX-Developer or retrieve data via MX-OPC Server. This module supports the sending and receiving of email messages. FX3U PLCs can communicate with other FX3U PLCs over Ethernet.

## **2.10 Miscellaneous Hardware**

### **Display Module – FX3U-7DM**

The FX3U-7DM Display Module connects to the FX3U and gives the operator a 4 line by 16 character text display screen. This unit mounts directly to the front of the PLC. It allows the operator to view the PLC clock, view or modify data addresses inside the PLC, perform error checks, show PLC status, as well as display user-defined text messages. The module can be protected with a password. A panel-mounting adapter is sold separately which will allow the unit to be mounted through the door of an enclosure.

This module is covered in more detail in Section 19 of the FX3U Series Manual – Hardware Edition, part number JY997D16501.

## **2.11 Power Supplies**

All PLCs in the FX family have an integrated power supply. The power supply accepts either 85 ~ 264 VAC or 24VDC (12 VDC on select FX1S models).

The FX power supply generates 2 types of bus power: 5VDC and 24VDC. The amount of current on each supply varies depending on model and size of the PLC. Check the hardware manual for the product you will be working with.

The 5 VDC runs on a bus that provides power to the CPU and extension blocks. Because there is a finite amount of power generated, this limits the number of extension blocks that can be connected.

The 24VDC current also runs to the 0V and 24V terminals on the PLC. This power can be used for discrete I/O extensions, and is also available for the powering of accessories such as sensors and HMIs. Be careful not to exceed the rated capacity of the power supply.

## How to compute power drain of the power supply

As has been stated before, the power supply can only support a certain number of extension block and special function blocks.

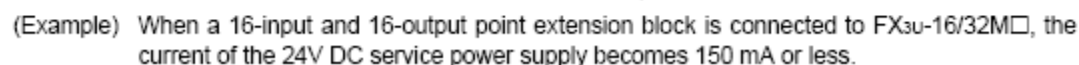
In the case of the FX3U, a power supply, **FX3U-1PSU-5V** can be added to offer additional bus voltage. This power supply creates 1A of 5VDC and 300mA of 24VDC. It is recommended to use this module after discrete I/O extensions as they require high amounts of 24VDC. Up to 2 can be used on a single FX3U system.

To determine if your system has a legal configuration, follow the steps below.

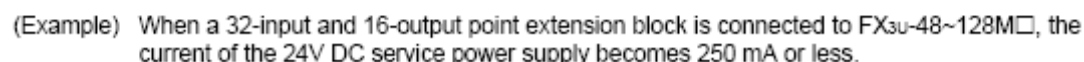
- 1) Total up the number of inputs and outputs you want to **add to the main unit**. Note that when a powered extension unit is used, all I/O points following it are calculated as a separate system. Each 16 input points require 100mA of 24VDC, and each 16 outputs require 150mA of 24VDC. Look at Table 1 or 2 for the PLC type or Table 3 or 4 for a powered extension unit. Cross-reference the input column with the output column. The number found is the remaining current.
- 2) Take note of the discrete I/O extension blocks, SFMs, expansion boards, and special adapters in the system. Look up the 5 VDC and 24 VDC current consumption ratings in Table 6. Where there is a listing for external 24VDC, this means the unit has terminals to connect an external power supply. In this case it does not need to be added to the internal consumption, unless the external wiring is connected to the 24VDC output terminals on the PLC.
- 3) Look at Table 5 to determine the amount of 5 VDC current available. Tables A1 and A2 are FX3U main units, and table D1 is powered extension units.
- 4) Add the module consumption and subtract from the available power supply.

NOTE: These tables are in the FX3U Hardware Manual (part number JY997D16501), Chapter 6.

FX3U-16MR/ES, FX3U-16MT/ES, FX3U-16MT/ESS, FX3U-32MR/ES, FX3U-32MT/ES, FX3U-32MT/ESS



FX3U-48MR/ES, FX3U-48MT/ES, FX3U-48MT/ESS, FX3U-64MR/ES, FX3U-64MT/ES, FX3U-64MT/ESS,  
FX3U-80MR/ES, FX3U-80MT/ES, FX3U-80MT/ESS, FX3U-128MR/ES, FX3U-128MT/ES, FX3U-128MT/ESS



**Output**

24	25				
16	100	50	0		
8	175	125	75	25	
0	250	200	150	100	50

(Example) 150 mA or less when 16 input points are added

**Input**

Number of added points

Number of added points

[illegible]

TABLE 5: 24VDC and 5VDC Supply Capacities

## FX3U CPU Units (AC Powered)

No.	Type	Input/output		Output current (mA)	
		Number of input/ output points [points]	Input/output [points]	5V DC power supply	24V DC service power supply
AC power supply/24V DC input/relay output type					
A1	FX3U-16MR/ES	16	8/8	500	400
	FX3U-32MR/ES	32	16/16		600
	FX3U-48MR/ES	48	24/24		
	FX3U-64MR/ES	64	32/32		
	FX3U-80MR/ES	80	40/40		
	FX3U-128MR/ES	128	64/64		
AC power supply/24V DC input/transistor output type					
A1	FX3U-16MT/ES	16	8/8	500	400
	FX3U-16MT/ESS	16	8/8		600
	FX3U-32MT/ES	32	16/16		
	FX3U-32MT/ESS	32	16/16		
	FX3U-48MT/ES	48	24/24		
	FX3U-48MT/ESS	48	24/24		
	FX3U-64MT/ES	64	32/32		
	FX3U-64MT/ESS	64	32/32		
	FX3U-80MT/ES	80	40/40		
	FX3U-80MT/ESS	80	40/40		
	FX3U-128MT/ES	128	64/64		
	FX3U-128MT/ESS	128	64/64		

## FX3U CPU Units (DC Powered)

No.	Type	Input/output		Output current (mA)	
		Number of input/ output points [points]	Input/output [points]	5V DC power supply	Power supply capacity for internal 24V DC
DC power supply/24V DC input/relay output type					
A2	FX3U-16MR/DS	16	8/8	500	400*1
	FX3U-32MR/DS	32	16/16		
	FX3U-48MR/DS	48	24/24		600*2
	FX3U-64MR/DS	64	32/32		
	FX3U-80MR/DS	80	40/40		
DC power supply/24V DC input/transistor output type					
A2	FX3U-16MT/DS	16	8/8	500	400*1
	FX3U-16MT/DSS	16	8/8		
	FX3U-32MT/DS	32	16/16		
	FX3U-32MT/DSS	32	16/16		
	FX3U-48MT/DS	48	24/24		600*2
	FX3U-48MT/DSS	48	24/24		
	FX3U-64MT/DS	64	32/32		
	FX3U-64MT/DSS	64	32/32		
	FX3U-80MT/DS	80	40/40		
	FX3U-80MT/DSS	80	40/40		



## FX2N Powered Extension Units

No.	Type	Input/output		Output current (mA)	
		Number of input/output points [points]	Input/output [points]	5V DC power supply	24V DC service power supply
<b>D1</b>	FX2N-32ER-ES/UL	32	16/16	690	250
	FX2N-32ET-ESS/UL	32	16/16		
	FX2N-32ER	32	16/16		
	FX2N-32ES	32	16/16		
	FX2N-32ET	32	16/16		
	FX2N-48ER-ES/UL	48	24/24		460
	FX2N-48ET-ESS/UL	48	24/24		
	FX2N-48ER	48	24/24		
	FX2N-48ES	48	24/24		
	FX2N-48ET	48	24/24		
	FX2N-48ER-DS	48	24/24		-
	FX2N-48ET-DSS	48	24/24		
	FX2N-48ER-D	48	24/24		
	FX2N-48ET-D	48	24/24		

TABLE 6: Expansion Devices

## FX3U Option Boards

No.	Type	Number of input/output occupied points	Current consumed (mA)	
			5V DC	Internal 24V DC
<b>B1</b>	FX3U-232-BD	—	20	—
	FX3U-422-BD	—	20*1	—
	FX3U-485-BD	—	40	—
	FX3U-USB-BD	—	15	—
	FX3U-CNV-BD	—	—	—

## FX3U Adapter Bus Modules

No.	Type	Number of input/output occupied points	Current consumed (mA)		
			5V DC	Internal 24V DC	External 24V DC
<b>C1</b>	FX3U-4HSX-ADP	—	30	30	0
	FX3U-2HSY-ADP	—	30	60	0
<b>C2</b>	FX3U-4AD-ADP	—	15	0	40
	FX3U-4DA-ADP	—	15	0	150
	FX3U-4AD-PT-ADP	—	15	0	50
	FX3U-4AD-TC-ADP	—	15	0	45
<b>C3</b>	FX3U-232ADP	—	30	0	0
	FX3U-485ADP	—	20	0	0

## FX2N Unpowered Extension Blocks

No.	Type	Number of input/ output points	Current consumed (mA)		
			5V DC	Internal 24V DC	External 24V DC
D2	Types for addition of input/output				
	FX2N-8ER-ES/UL	16	—	125	0
	FX2N-8ER	16	—	125	0
	Types for addition of input				
	FX2N-8EX-ES/UL	8	—	50	0
	FX2N-8EX	8	—	50	0
	FX2N-8EX-UA1/UL	8	—	50	0
	FX2N-16EX-ES/UL	16	—	100	0
	FX2N-16EX	16	—	100	0
	FX2N-16EX-C	16	—	100	0
	FX2N-16EXL-C	16	—	100	0
	Types for addition of output				
	FX2N-8EYR-ES/UL	8	—	75	0
	FX2N-8EYT-ESS/UL	8	—	75	0
	FX2N-8EYR	8	—	75	0
	FX2N-8EYT	8	—	75	0
	FX2N-8EYT-H	8	—	75	0
	FX2N-16EYR-ES/UL	16	—	150	0
	FX2N-16EYT-ESS/UL	16	—	150	0
	FX2N-16EYR	16	—	150	0
FX2N-16EYS	16	—	150	0	
FX2N-16EYT	16	—	150	0	
FX2N-16EYT-C	16	—	150	0	

## Special Function Modules

No.	Type	Number of input/occupied output points	Current consumed (mA)		
			5V DC	Internal 24V DC	External 24V DC
<b>E1</b>	FX3U-4AD	8	110	0	90
	FX3U-4DA	8	120	0	160
	FX3U-20SSC-H	8	100	0	220
<b>E2</b>	FX2N-2AD	8	20	50 <sup>18</sup>	0
	FX2N-2DA	8	30	85 <sup>18</sup>	0
	FX2N-4AD	8	30	0	55
	FX2N-4DA	8	30	0	200
	FX2N-4AD-TC	8	30	0	50
	FX2N-4AD-PT	8	30	0	50
	FX2N-8AD	8	50	0	80
	FX2N-5A	8	70	0	90
	FX2N-2LC	8	70	0	55
	FX2N-1HC	8	90	0	0
	FX2N-1PG(-E)	8	55	0	40
	FX2N-10PG	8	120	0	70 <sup>11</sup>
	FX2N-232IF	8	40	0	80
	FX2N-16CCL-M	8 <sup>12</sup>	0	0	150
	FX2N-32CCL	8	130	0	50
	FX2N-64CL-M	8 <sup>13</sup>	190	Supplied from power supply for CC-Link/LT	
	FX2N-16LNK-M	0 <sup>14</sup>	200	0	90
	FX2N-32ASI-M	8 <sup>15</sup>	150	0	70
<b>E3</b>	FX0N-3A	8	30	90 <sup>16</sup>	0

## Special Function Modules (Continued)

No.	Type	Number of input/ occupied output points	Current consumed (mA)		
			5V DC	Internal 24V DC	External 24V DC
<b>E3</b>	FX2N-10GM	8	—	—	5
	FX2N-20GM	8	—	—	10
	FX2N-1RM(-E)-SET	8	—	—	5

## FX3U Display Module

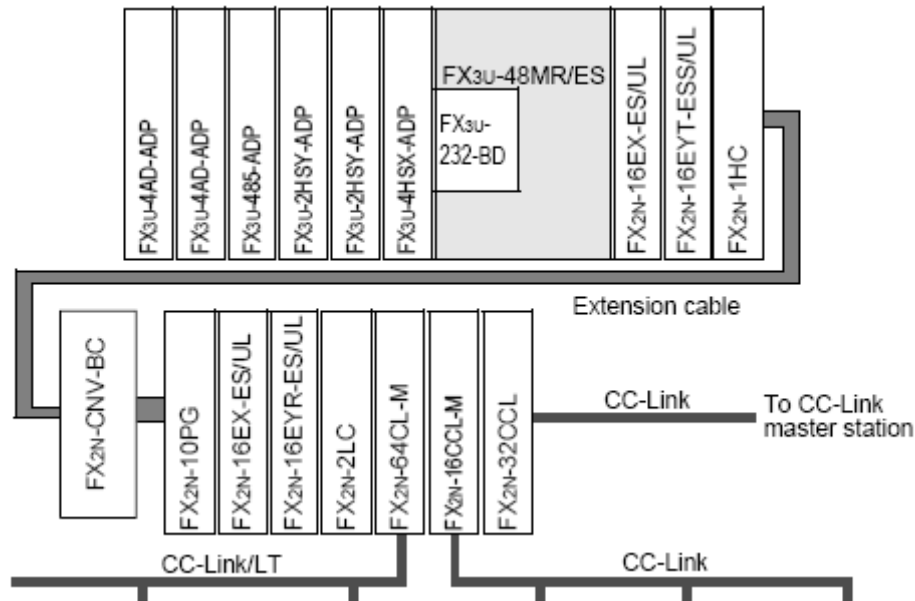
No.	Type	Number of input/ occupied output points	Current consumed (mA)		
			5V DC	Internal 24V DC	External 24V DC
<b>G1</b>	FX3U-7DM	—	20	0	0

Worksheet for Exercise 2.12

	Classification	Number of connected units	Type	Number of input/output points [points]	Capacity of built-in power supply	
					5V DC power supply[mA]	24V DC service power supply[mA]
				1-1	1-2	1-3
With built-in power supply	<div>A</div> <div>Main unit</div>	1	FX3U-48MR/ES	48	500	600

	Classification	Number of connected units	Type	Number of input/output (occupied) points [points]	Calculation of current consumption of built-in power supply	
					5V DC power supply[mA]	24V DC power supply[mA]
Enter the products connected to the main unit.	<b>B</b> Expansion board	1	FX3U-	—		—
	<b>C</b> Special adapter	10	FX3U-	—		
			FX3U-	—		
			FX3U-	—		
			FX3U-	—		
			FX3U-	—		
			FX3U-	—		
			FX3U-	—		
			FX3U-	—		
			FX3U-	—		
			FX3U-	—		
	<b>D2</b> Input/output extension block	—	FX2N-		—	
			FX2N-		—	
			FX2N-		—	
			FX2N-		—	
			FX2N-		—	
			FX2N-		—	
			FX2N-		—	
			FX2N-		—	
			FX2N-		—	
			FX2N-		—	
			FX2N-		—	
	<b>E</b> Special function unit/block	8	FX0N/FX2N-			
			FX0N/FX2N-			
			FX0N/FX2N-			
			FX0N/FX2N-			
			FX0N/FX2N-			
			FX0N/FX2N-			
	<b>G</b> Display module	1	FX3U-7DM			
				<b>2-1</b>	<b>2-2</b>	<b>2-3</b>
Calculate the totals.						

## 2.12 EXERCISE Power Supply Calculation



Using the worksheet on the previous page, combined with the preceding tables, use the steps as discussed previously, determine the power consumption of the system above.

1. Is this a valid configuration? \_\_\_\_\_
2. If not, why not? \_\_\_\_\_
3. If not, how can it be corrected? \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_

## 2.13 Memory Types

The FX3U comes with enough internal RAM to hold a program of 64K steps. The 64K memory in the FX3U is NOT expandable.

The FX2N comes with 8K steps of program memory. By using an 8K RAM memory module, the PLC's memory can be expanded to 16K steps.

The FX2NC comes with 8K steps of program memory. By using a 16K memory module, the PLC's memory can be expanded to 16K steps.

The FX1N comes with 8K of program memory, which is not expandable.

The FX1S comes with 2K of program memory, which is not expandable.

A cassette that contains non-volatile memory can add portability to a program. The program is saved in the cassette, and when the cassette is connected to another PLC, the program in the cassette overrides the program in the PLC RAM. The program in the RAM is retained; the PLC simply uses the program on the cassette instead of the memory. A special memory card with a program loader option allows the memory card to backup or overwrite the program in the PLC.

Note that since the PLC uses the cassette instead of the PLC RAM, the cassette memory is **not cumulative** with the memory of the PLC.

Depending on the PLC type, there are 4 types of program memory. **RAM**, **EPROM**, **EEPROM** and **FLROM** modules are available. Each type has its advantages and disadvantages. Your PLC type, the application, and the level of security you require will dictate which type of memory you use.

### **RAM (Random Access Memory)**

RAM memory is volatile, which means it relies on a battery to keep the program in memory. It is easy to make changes to a program that is kept in RAM memory. On-line program changes are allowed with RAM.

### **EPROM (Erasable Programmable Read Only Memory)**

EPROM memory is permanent; it retains the program memory without a battery. It is difficult to make changes to programs stored in EPROM, because an ultraviolet light is required to erase it. An EPROM burner is required. Online changes are NOT allowed

### **EEPROM (Electrically Erasable Programmable ROM)**

EEPROM is permanent memory also; the program is retained with no battery connected. It is easy to make changes to EEPROM, because it is electrically erasable. On-line changes are allowed with FX1N and newer.

## FLROM (Flash ROM)

FLROM is permanent memory also; the program is retained with no battery required. FLROM operates similarly to EEPROM, as it is electrically erasable and can be overwritten many times. Only the FX3U uses Flash ROM.

The chart below details which types of memory modules are available for each PLC type.

PLC Type	RAM	EPROM	EEPROM	FLROM
<b>FX1S</b>	No	No	8K w/loader	No
<b>FX1N</b>	No	No	8K w/loader	No
<b>FX2N</b>	8k*	8k	4k, 8k	No
<b>FX2NC</b>	No	No	4k, 16k, RTC*	No
<b>FX3U</b>	No	No	No	16k, 64k, 64k w/loader*

Notes:

- FX2NC does not have an internal real time clock. A memory module with a real time clock must be used if real time clock functionality is required
- FX-RAM-8 is only used to extend FX2N memory from 8K to 16K
- FX3U-FLROM-64L must be used if program loader functionality desired on FX3U

The questions to ask are ...

- 1) Which PLC type are you using and what options are available?
- 2) Do you want the program to be retained, even if battery power is lost?
- 3) Do you want to be able to change the program easily?





# LESSON 3 – Programming Equipment

This lesson discusses the hardware and software requirements to program a PLC. The student will also be shown how to connect the system together. Alternatives to using a laptop to program are also covered.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ List the hardware required to program a PLC with a laptop.
- ✓ Describe alternatives to using a laptop for programming.
- ✓ Describe how to connect a PLC system to a laptop.
- ✓ List the types of software used to program a PLC.

**Materials:** FX-Series PLC Training Manual

## **3.1 Hand-Held Programming Units**

Most programmers will use a laptop computer for making changes. Due to issues of expense or security this may not always be the case. Convenience may also be a factor: who wants to find and connect a computer and cable, and wait for the boot-up, etc. when all you want to do is change a timer, add a contact or make an address change.

The **FX-10P-E** and **FX-20P-E** can be useful for situations like these. They are handheld LCD display programmers that connect directly to the PLC. These units allow for programming changes and monitoring.

**FX-10P-E** – This unit permits on-line program changes only. The display is 2 lines by 16 characters in size. It can read programs from the PLC, write to the PLC, monitor the program, and change the states of devices (forcing bits).

**FX-20P-E** – This unit is a more powerful version of the FX-10P-E. The display is backlit and has a size of 4 lines by 16 characters. It supports both online and offline programming.

## **3.2 Programming Software**

There are 2 software packages that can be useful in the programming and troubleshooting of all Mitsubishi PLCs. These packages are GX-Developer and GX-Simulator. Some users may be familiar with some of the previous software packages, including MEDOC (DOS application), FX-WIN, and GPP-Win (previous version of GX-Developer). This class will deal with the latest version of GX-Developer.

**GX-DEVELOPER** – This is Windows based software (95, 98, NT, 2000, XP). It can be used to program all PLC lines, including FX-series, Q-series, A-series and the Motion A-series PLCs. This software, which replaces the previous Windows package, GPP-WIN, has a large number of troubleshooting and diagnostic

features, as well as the ability to easily communicate over networks. It also has import capabilities to bring in programs written in older software packages.

**GX-DEVELOPER-FX** – This is Windows based software (95, 98, NT, 2000, XP), is based on the GX-Developer software package. This software has all the features of GX-Developer, but only supports the entire FX Series and its functionality. This package is offered at a discounted price versus the entire GX-Developer package, making it a cost saving option for anyone who does not need to program the rack-based PLC controllers.

**GX-SIMULATOR** – This Windows based software isn't used to program PLCs, but to assist in program troubleshooting. This software will actually act as a PLC, allowing a GX-Developer program to be tested, without having to download to an actual PLC system. This software is not included with GX-Developer, it must be purchased separately. It must be loaded on the same computer that GX-Developer is installed on.

Using GX-Simulator in troubleshooting allows the programmer to debug roughly 90% of the problems in a program before hardware is ever connected. It has the ability to simulate discrete and analog I/O, serial and network communications, and special function modules. You can build timing charts and simulate inputs such as a run signal based on the run output turning on automatically. You can step the processor through the system step by step, partial execution, or skip steps.

### **3.3 GX-Developer Overview**

This section will provide an overview of things to remember when installing and using GX-Developer. This not intended to be a full tour of GX-Developer's features. That will be accomplished during the coursework.

#### **INSTALLATION**

**IMPORTANT:** Before installing GX-Developer, make sure to remove any previous version of GX-Developer or GPP-WIN using the Add/Remove Program utility in Windows Control Panel. Do NOT delete the directories and attempt to reinstall. The uninstall process will not erase any of the PLC programs that have been previously created.

When Windows does the Program Remove, it will display a message that states some elements could not be removed, please remove manually. These are the previously created PLC programs, so their removal must be done manually at your discretion.

The CD should have an automatic menu loaded when inserted into your PC. If it does not, run the program by double clicking on the autorun.exe file located in the root directory of your software CD. You can also open the Windows Start Menu, select Run, and type or browse to X:\autorun.exe, where X is your CD drive letter.

It is very important that the prompts are read and responded to, because this is the only opportunity to install the Import From MELSEC MEDOC features. When this prompt appears you must click on each check box to install, otherwise the programmer will be unable to import MEDOC programs in the future without first reinstalling GX-Developer.

After installing GX-Developer you can browse the CD-ROM. All the manuals pertaining to GX-Developer are available in .PDF format, along with a free Acrobat viewer. The programmer is encouraged to install the viewer and copy the manuals to the hard drive.

Make sure to register the software after installation. This ensures that the programmer will get information about updates and qualifies the software for a free upgrade if a new version is released within a certain time period after purchase.

## PART NUMBER & VERSION

When ordering GX-Developer, the Part Number is **GX-DEV-C#**, where # is a number of licensed users: 1, 5, 10, 25, and 50. For the FX-only version, the part numbers is **GX-DEV-FX-C#**.

The software version installed can be found within GX-Developer by going to the Help menu and selecting Product Information. If the part number begins with SW2 ~ SW5, then the software was actually GPP-Win. Starting with SW6, the software was renamed GX-Developer.

## FEATURES

### *Multiple windows*

It is possible to have multiple windows open. Thus different windows showing different sections of code and various monitoring windows can be open at once.

### *Import from other formats*

Programs written in MEDOC, GPPA, and FX-WIN can be imported into GX-Developer.

### *Workspace setup is saved*

Save and Save As preserves the last state of the program, including all open and positioned windows. Thus it isn't necessary to constantly recreate the desired work environment every time the program is reopened.

### *Entry Ladder Monitor*

A new feature to GX-Developer, the Entry Data Ladder allows the programmer to copy rungs from different sections of the program into one screen for easy monitoring.

### *Local Device Monitor*

A new feature to GX-Developer, this monitor allows the programmer to monitor the states of local devices (*used with the QCPU only*)

## **CAUTIONS**

### *Importing from MEDOC*

GX-Developer writes a temporary file during the import process. If a floppy disk is write-protected or doesn't have enough space, the import will fail. Copying the original files to the hard drive before importing is recommended.

### *Importing Documentation*

Documentation will not import if it has foreign characters (like the tilde ~).

### *Copy and Paste*

Cannot copy and paste between GX-Developer and other Windows applications, with the exception of comments. These can be copied from the Comment table to an Excel spreadsheet.

### *Multiple Projects*

Only 1 GX-Developer project can be open per session. If copying and pasting between multiple GX-Developer projects is required, multiple sessions of GX-Developer must be opened. This can be done from the File menu, Start New GX-Developer session, or by starting GX-Developer a second time from the Start Menu.

### *Read Only Files*

GX-Developer cannot open read only files. If a project has been archived to a CD, the CD is read only, and all files on the CD will be read only. If these files are copied to a hard drive, they will still be marked as read only. You will need to change these files to be read/write in the attributes tab in Windows Explorer before GX-Developer will see them as a project. This applies to the project folder, all subfolders and all files.

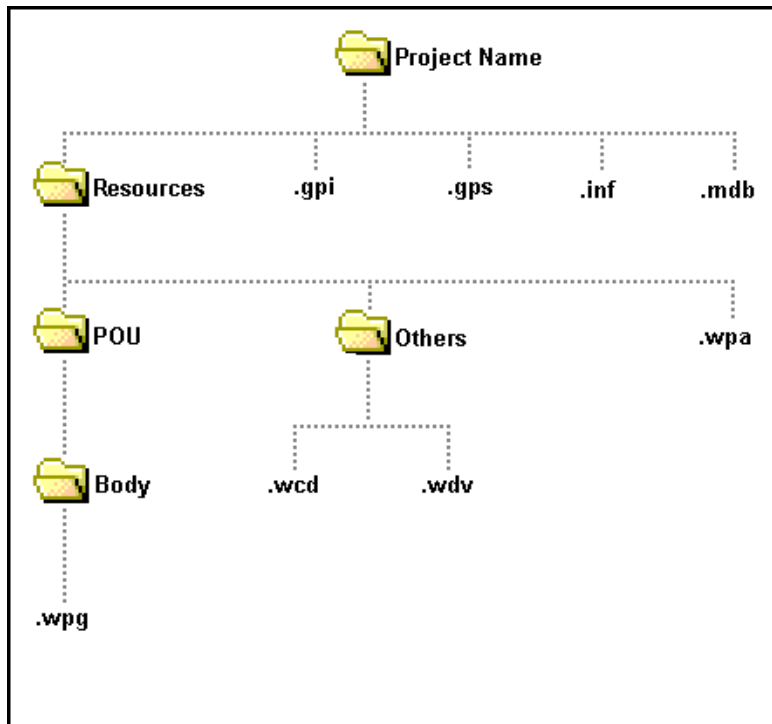
### *Zipping the program for distribution*

A GX-Developer program has a special format of folders and placement of files. While it is possible to manually recreate this format, when emailing a program it is advisable to do a Save As and zip the whole folder (selecting the include subfolders option if using WinZip). This will preserve the format.

### *Disk Defragmentation Utilities*

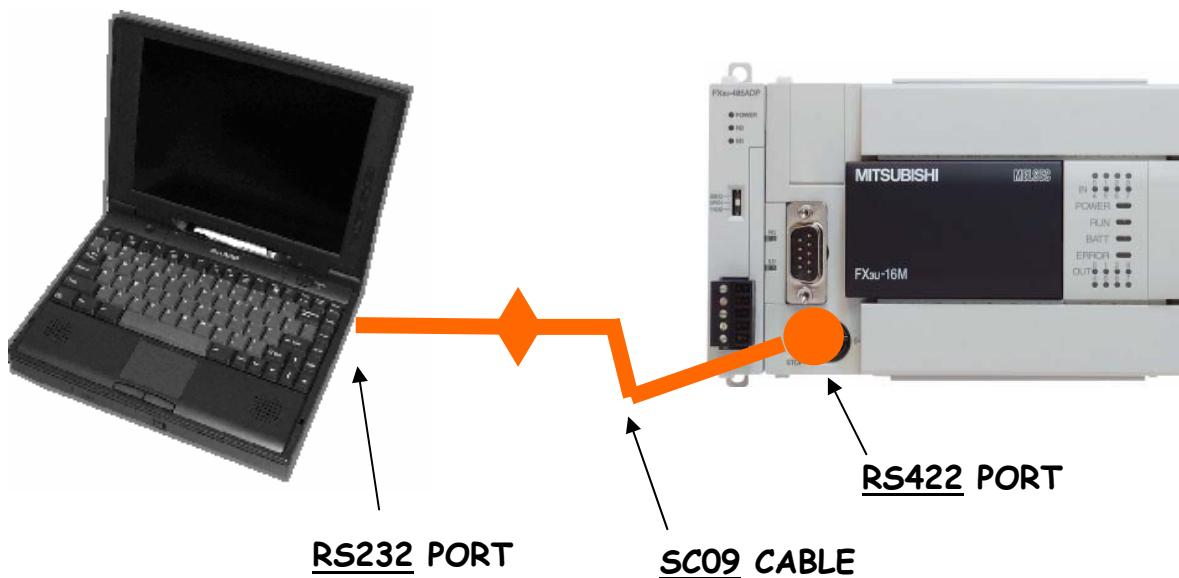
Use of a 3<sup>rd</sup> party disk defragmentation utility could corrupt the GX-Developer license. The GPPW directory should be excluded from the disk defragmentation process. The defragmentation utility which comes with Windows can be run without restriction.

### 3.4 File Format



### 3.5 Hardware Connection

The **SC09** cable is used to connect the PLC to a personal computer for program development. The circular 8 pin port on the PLC CPU module uses the RS422 standard of communication. Most personal computers only have a RS-232 communication port. For this reason, the SC09 cable has a conversion circuit built into the connector housing. The cable includes hardware that converts from RS422 to RS232.



If the PC does not have an RS232 serial port, there are a couple of options available.

- On the FX3U series PLC, you can install an **FX3U-USB-BD** and connect to the PLC directly from a USB port.
- For FX1S, FX1N, FX2N, FX2NC, and FX3U PLCs, you can use a cable with USB to serial converter built in. Mitsubishi offers such a cable, part number **FX-USB-AW** for sale. This cable has the 8-pin round DIN programming plug as found on all current FX-Series PLCs. It will not work with Q-Series, A-Series, or older FX-Series with the 25-pin D-SUB connector.
- There are many third parties who offer USB to Serial adapters for sale. Mitsubishi has tested only a few of these, and recommends the **Keyspan USA-19(W or HS)** adapters or **CP Technologies CP-US-03**. These can be used with the standard SC09 cable, and are available from computer or office supply stores.

# LESSON 4 – Numbering Systems

The PLC uses several numbering systems besides the Base 10 decimal system. An understanding of these other systems is crucial to successful programming.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Name the different numbering systems.
- ✓ Describe how the different systems represent numbers.
- ✓ Convert between number systems.

**Materials:** FX-Series PLC Training Manual

## 4.1 Binary Numbers

In the binary number system, each digit is called a binary unit, or **bit** for short. Binary is a Base 2 numbering system, meaning there are only 2 possible values for each digit. Each **bit** can have a value of only '0' or '1'.

A group of **4 bits** is called a **NIBBLE**

A group of **8 bits** is called a **BYTE**

A group of **16 bits** is called a **WORD**

The position of a bit, in a byte or word, determines its *value*. Starting from the right side, bit number 0 has a value of '1'. As you move left, the bit value doubles with each position. Bit 1 has a value of 2, bit 2 has a value of 4, bit 3 has a value of 8, etc.

The following example shows a BYTE:

Bit Value	128	64	32	16	8	4	2	1
	0	0	0	0	0	0	0	0
Bit Number	7	6	5	4	3	2	1	0

To convert from binary to decimal, just add the bit values of the bits that are set to '1'.

Bit Value	128	64	32	16	8	4	2	1
	0	0	0	0	0	0	0	1
Bit Number	7	6	5	4	3	2	1	0

<u>Binary Word</u>	<u>Decimal Value</u>
0000 0001 .....	1
0000 0010 .....	2
0000 0100 .....	4
0000 1000 .....	8
0000 0011 .....	3
0000 0101 .....	5
0000 0110 .....	6

With 4 bits you can count from 0 to 15 ...

0000 0000 .....	0	
0000 1111 .....	15	(8+4+2+1=15)

## 4.2 Hexadecimal Numbers

Hexadecimal is a Base 16 number system, meaning each digit has 16 possible values. Each digit then represents a number from 0 to 15. This is accomplished by using letters for values greater than 9.

From 0 to 9 the count is the same as decimal ...

<b>DECIMAL</b>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
<b>HEXADECIMAL</b>	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Then starting with 10, in hexadecimal, letters are used.

Hexadecimal is also considered a **shorthand** method of writing **BINARY**. Each Hexadecimal digit represents 4 binary bits of data.

<b>BINARY</b>	0000	0010	0011	0100	1000	1001	1010	1011	1111
<b>HEXADECIMAL</b>	0	2	3	4	8	9	A	B	F



### 4.3 Octal Numbers

Octal is a Base 8 numbering system, meaning there are 8 possible values. The numbers for the octal system are **0 ~ 7**.

In decimal, when the count passes 9, 19, etc. the count restarts at 0, but the tens digit is incremented by one (i.e. after 9 comes 10, after 19 comes 20).

In the same way, when the count passes 7 in octal, the count restarts at 0 and the tens digit is incremented. Thus after 7 comes 10, after 17 comes 20.

<b>DECIMAL</b>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
<b>OCTAL</b>	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	21	22

Octal is also a **shorthand** method of writing **BINARY**.  
Each Octal digit represents 3 binary bits of data.

<b>BINARY</b>	000	010	011	100	001	101	110	111
<b>OCTAL</b>	0	2	3	4	1	5	6	7

Putting the charts for hexadecimal and octal together, it becomes easy to convert between hex and octal:

Convert 349AFh to Octal

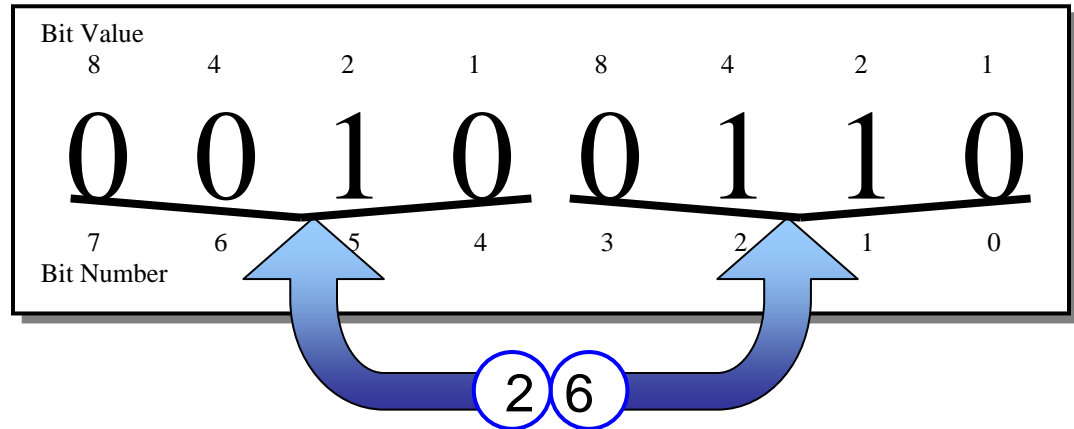
HEXADECIMAL	3	4	9	A	F		
BINARY	0011	0100	1001	1010	1111		
(Regroup into 3s)	00	110	100	100	110	101	111
OCTAL	0	6	4	4	6	5	7

Thus 349AFh is 644,657o

## 4.4 Binary Coded Decimal

Binary coded decimal has the same counting sequence as Decimal, 0~9, but has the same format as binary. Break down each decimal digit into 4 binary bits. When converting BCD to Binary, break down each decimal digit into **4 binary bits**.

Decimal 26 =



BCD was developed with the use of decimal devices in mind, like thumbwheels and seven segment displays. Decimal devices only count from 0~9, and require the use of 4 binary bits to do so.

DECIMAL	2	9	12	30
BCD	0000 0010	0000 1001	0001 0010	0011 0000

The difference between Binary and BCD is apparent when converting from decimal.

Converting decimal 12 to binary, bits 3 (value of 8), and 2 (value of 4) are '1'.  
Converting BCD 12 to binary, bits 4 (value of 8), and 1 (value of 4) are '1'.

In the FX-Series PLC there are dedicated commands to convert between BCD and BINARY.

The **BCD** command converts from **BINARY** to **BCD**.  
The **BIN** command converts from **BCD** to **BINARY**.

## 4.5 EXERCISE      Number Systems Conversion

In this exercise, convert the following numbers to the given number system.

- |                         |      |         |        |
|-------------------------|------|---------|--------|
| ❶ Convert decimal 2 to  | HEX= | BINARY= | OCTAL= |
| ❷ Convert decimal 10 to | HEX= | BINARY= | OCTAL= |
| ❸ Convert decimal 16 to | HEX= | BINARY= | OCTAL= |
| ❹ Convert decimal 28 to | HEX= | BINARY= | OCTAL= |
| ❺ Convert decimal 6 to  | BCD= |         |        |
| ❻ Convert decimal 16 to | BCD= |         |        |
| ❼ Convert decimal 35 to | BCD= |         |        |



# LESSON 5 – Numeric Data in PLCs

Most PLC applications will require the handling of data, whether manipulating counter and timer values, reading data from a Special Function Module and processing then information, or high-level mathematical computations. It is critical that the programmer understand how the PLC ‘sees’ and handles the different types of data that can be encountered.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Describe how a PLC handles integer and decimal numbers.

**Materials:** FX-Series PLC Training Manual

## 5.1 Integer Handling

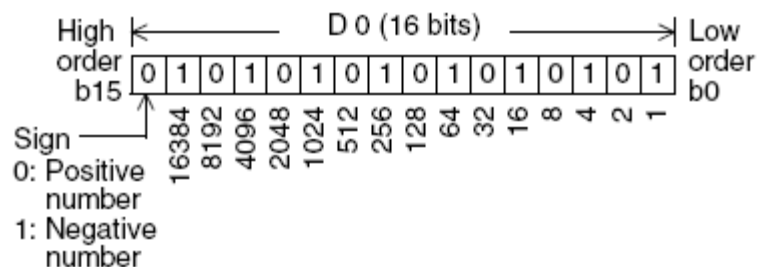
A very important fact to note is that the PLC handles only integer data by default. This means that it sees numbers only as whole numbers...a 1, 10, -2, etc. The PLC doesn't recognize fractional numbers like 3.14159. Attempting to enter a number like this will cause an error. If a mathematical operation like 5 divided by 3 (answer is 1.667) is performed, the PLC will drop the decimal part and give an answer of 1. The remainder, in this example 2, is stored in the data register following the destination register of the math instruction.

### 16 BIT NUMBERS

Integers in PLCs are 16 bit numbers, unless otherwise declared by programming. Remembering back to the binary number lesson, this means that the numeric range for integers is: 0000 0000 0000 0000 to 1111 1111 1111 1111. When converted to decimal, this means the integer range for the PLC is 0 to 65,535.

The integer range is actually -32,768 to + 32,767. This is due to the fact that the bit that is farthest to the left (bit 15) is used by the PLC as the sign bit. This bit is also known as the Most Significant Bit (MSB). If it is a 1, the number is negative, if a 0 it is positive. Thus the actual largest positive number is 0111 1111 1111 1111 which is 32,767. If the program is incrementing and goes above 32,767 it goes to -32,768. If the program is decrementing a number and goes below -32,768, it goes to 32,767.

1000 0000 0000 0000 = -32,768. Why is this?



The PLC uses a numbering format known as two's complement to display negative numbers. Two's complement is easy to calculate:

- ❶ Change all 1's to 0's and all 0's to 1's. This new number is known as the complement.
- ❷ Add a 1 to the number

0111 1111 1111 1111	this is 32,767
1000 0000 0000 0000	this is the complement (the sign bit is not included in the complement but needs to be a 1 for this number to be negative)
<u>                  1                  </u>	add the 1
1000 0000 0000 0001	this is -32,767.

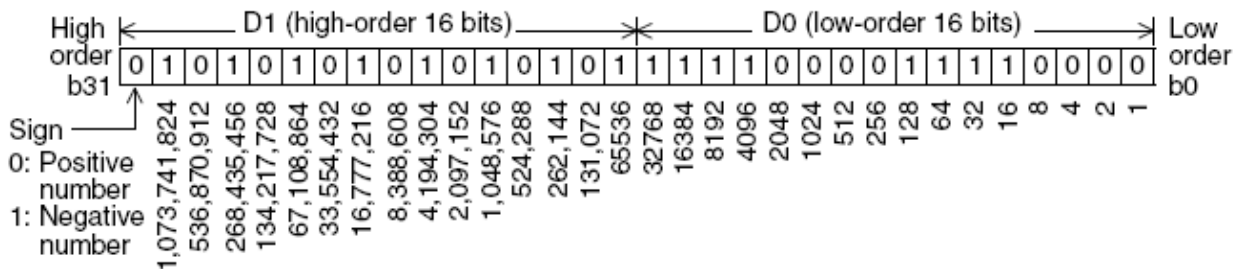
1000 0000 0000 0000 is 1 less than -32,767, therefore it is -32,768.

The **NEG** command can be used to perform a two's complement on either 16-bit or 32-bit data to change a negative to a positive.

## 32 BIT NUMBERS

As stated before, integers are 16 bit by default. When certain commands, to be discussed later, are used, it is possible to have 32 bit numbers. When this is done, the PLC looks at two 16 bit registers as 1 large register. Bit 15 is no longer considered as the MSB. The PLC now considers bit 32 to be the MSB. This allows the PLC to display integers with the range -2,147,483,648 to 2,147,483,647 (except when using scientific notation, see below).

When 32 bit instructions are used, it is important to note that the number occupies both the destination register and the following register. Take this into account when writing the program because overwriting the second register can have unpredictable consequence on the data.



## 5.2 Decimal Handling

As mentioned above, the default method for handling decimals is to drop them. This restriction can be avoided through the setting of the float flag (M8023) and the use of the floating point instructions in section 5.1 of the FX3U Programming Manual.

There are 2 formats for displaying decimal numbers: Scientific Notation and Floating Point Format.

### SCIENTIFIC NOTATION

Scientific Notation uses 2 registers to store the mantissa and the exponent. The mantissa is the first 4 significant digits of a number, and the exponent shows the position of the decimal. This format cannot be used in calculations, but is useful for displaying data.

**Example:** 1,238,900 would be displayed as  $1238 \times 10^3$ . 1238 is the mantissa and  $10^3$  which indicates the decimal is 3 places to the right, is the exponent.

A number between 0 and 1 or 0 and  $-1$  is represented by a negative exponent. The exponent shows how many places to the left of the mantissa to locate the decimal.

**Example:** .00123 would be displayed as  $123 \times 10^{-5}$ .

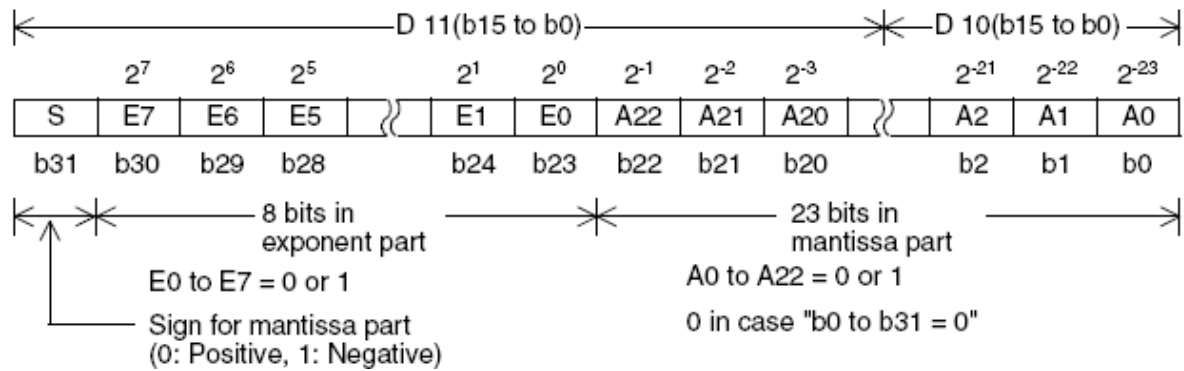
This format allows number outside of the normal 32 bit range ( $\sim \pm 2$  billion) to be displayed. The number range is  $9999 \times 10^{35}$  to  $-9999 \times 10^{35}$ . The trade off is a loss of precision, only 4 significant digits.

The method for storing a scientific notation number: The mantissa is stored at register D, and the exponent is stored at D+1. In the above examples, if 1,238,900 were to be stored in D0 and .00123 was stored at D2, the data registers would appear as follows:

D0	1238
D1	3
D2	123
D3	-5

### FLOATING POINT

Similar to the Scientific Notation format, this format displays the number in register D and register D+1. The mantissa occupies all 16 bits of D and the first 7 bits of D+1. The exponent occupies the last 9 bits of D+1, with bit 15 acting as a sign bit.



It is not possible to monitor and interpret the values in D and D+1 for Floating Point format in the same way that Scientific Notation can be monitored. The representation of floating point numbers follows a special format recommended by I.E.E.E. (Institute of Electrical and Electronic Engineers).

The main advantage to using this format is the accuracy over Scientific Notation. The number pi (~3.1415926) appears as 3.141592 (7 significant digits) in floating point format, and as  $3142 \times 10^{-3}$  in scientific notation.



# LESSON 6 – System Devices

To write a program for a PLC, it is necessary to be familiar with the devices that are used in the instructions. An overview is provided here, with more detailed information to follow in later lessons.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Name and describe the devices used to make a program.

**Materials:** FX-Series PLC Training Manual

A common question when discussing system devices is the number of each that is available to use. This varies depending on the CPU. Make sure to check the user's manual for the model of CPU under question.

## **X – Physical Inputs**

X addresses are used to designate physical inputs.

## **Y – Physical Outputs**

Y devices are used to designate physical outputs.

## **M – Auxiliary Relays**

**M** relays are internal bit devices. They are internal bits that can be used for any function needed. When an M device coil is energized, the corresponding M device contact becomes active.

In GX-Developer, it is possible to configure M bits to be battery-backed. This means the bits will maintain their current state (Off or On) in the event of power loss.

There is a group of M devices that have dedicated functions. They are **M8000 ~ M8511**. The meanings are described in tables in chapter 36 of the FX3U Programming Manual.

## **S – State Relays**

**S** relays are internal bit devices. They are used in STL programming to indicate which step, or section of ladder logic code, is active. If STL programming is not used, these bits can be utilized in the same manner as M bits.

In GX-Developer, it is possible to configure M bits to be battery-backed. This means the bits will maintain their current state (Off or On) in the event of power loss.

If STL programming is utilized in conjunction with the IST instruction (Initial State) causes certain state relays to have special operations. 2 examples are: S0 is the manual operation return state and S2 is the automatic operation return state.

One last use of state relays is as a fault annunciator. Through programming techniques described in chapter 4.4 of the FX3U Programming Manual, **S900 ~ S999** can be used as user defined fault indicators.

## T - Timers

**T** devices are timer devices by default timers are either 100msec time increments, 10msec time increments, or 1msec time increments depending on the timer address. Most timers, depending on their address, are non-retentive, meaning they do not hold their current value if the input conditions stop conducting. In the FX2N, FX, FX2NC, timers with addresses T246 and above are **retentive**. This means the timer hold its value until it is reset. Retentive timers have a time base of either 100msec or 1msec, depending on the address.

When a timer reaches the associated preset value the T device coil is energized, the corresponding T device contact becomes active also. All timers are 16 bit, meaning the maximum preset is +32767. Valid presets are K values and D data registers. For a 100msec timer the maximum time is 3276.7 seconds.

In GX-Developer, it is possible to assign a range of timers to be battery-backed. This means the timers will hold their accumulated value in the event of a power loss – provided the logic that drives the rung is battery-backed as well. Otherwise the timer will be reset.

## C - Counters

**C** devices are counter devices. The standard is all counters are retentive, holding their current count until reset. In GX-Developer you can assign counters to be battery-backed. When a counter reaches the associated preset value, the C device coil is energized, the corresponding C device contacts become active also. Counters can be 16 or 32 bit, meaning the maximum count range is –32768 to +32767 (16 bit) or -2,147,483,648 to 2,147,483,647 (32 bit). Negative presets are not very useful, as will be explained in more detail in Lesson 12. Valid presets are K values and D data registers.

There are 3 types of counters: 16 bit up counters, 32 bit up/down counters, and 32 bit high-speed counters. Within the high-speed counter category there are 1 phase, 2 phase, and A/B phase counters.

Counters will be covered in more detail in the chapter on Timers and Counters.

## D – Data Registers

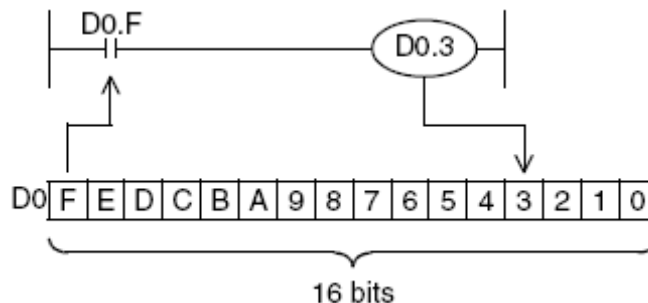
**D** devices are data registers. Data registers can be used for any purpose. All data registers are 16 bit, meaning the limits of numeric data is from -32768 to 32767. In ladder you can perform 32 bit operations, in that case 2 consecutive D registers are used together and the maximum numeric value can be 2,147,483,647 to -2,147,483,648.

In GX-Developer, it is possible to configure D devices to be battery-backed. This means the bits will maintain their current value in the event of power loss. It is possible to retain data in all registers by turning on special relay M8033.

File registers are data registers that are stored with the program, rather than with the PLC. They are declared in the Parameters section in groups of 500. Each group of 500 declared registers reduces the amount of program steps by 500.

Data registers **D8000~D8511** are dedicated for PLC diagnosis and special functions. Their meanings can be seen in the FX3U Programming Manual chapter 36.

In FX3U, the PLC has the ability to directly access bits within a word. By placing a decimal point between the word address and bit address, you can reference the status of a single bit within a word. An example would be D100.0 which references the least significant bit in word D100.



## R and ER – File Registers and Extension File Registers

**R** registers are file registers. File registers are an extended set of data registers which are battery backed. They are stored in the PLC's internal memory, and can be accessed from within the ladder logic program.

**ER** registers are extension file registers. Extension file registers are only available if a memory cassette is installed in the PLC, as the ER data is stored in the memory cassette. There are dedicated commands in the PLC for access to data in the extension file registers. ER registers are accessed by copying their contents into the R file registers.

## V and Z – Index Registers

Index registers are indicated by the symbols V and Z. V and Z can both be used for 16 bit applications, while only Z can be used with 32 bit instructions. Values stored in an index register are used as offsets to specified devices. There are 8 available V indexes (V0-V7) and 8 available Z indexes (Z0-Z7). Indexes are used by attaching their address as a suffix to an address in the program. If V or Z are referenced without a number, the number 0 is implied.

If V0 has the value of K2, then D10V is  $D10 + 2 = D12$ .

If Z2 has a value of K8, then Y1Z2 = Y11. Remember X and Y address in octal, so the offset number is converted to octal and added to the base address. Because 8 in decimal is 10 in octal, the address is increased by 10.

These devices are useful for accessing a large amount of information without the use of a large number of rungs.

## P - Pointers

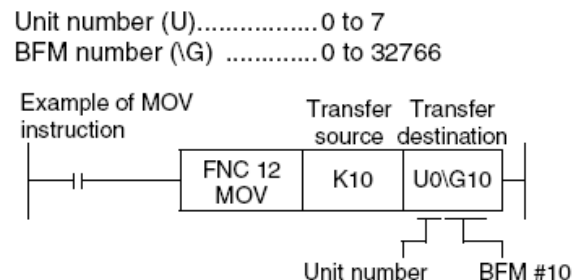
P addresses are pointers. Pointers are used with jump, call, and interrupt instructions to alter program flow. Call and JMP instructions cause the program scan to stop in one area and to move to another, either by calling a subroutine or by moving to another location in the same program. The pointer is used as 1) the destination of a jump or call instruction and 2) as the name of a jump point or subroutine.

## K, H, and E – Numerical Constants

K, H and E are used to indicate a numeric constant. A PLC instruction will not recognize a “1” written in Relay Ladder code, but would understand “K1” or “H1”. A “K” indicates the constant is a decimal constant. The “H” designates a hexadecimal constant. An “E” indicates a floating point numeric value. Constants can be either 16 bit (-32768 to 32767) or 32 bits (-2,147,483,648 to 2,147,483,647) in decimal.

## U\G – Buffer Memory Access (FX3U Only)

The U#G# addressing system allows direct access to the buffer memory addresses located within a special function module. The U number specifies the SFM number, and the G number specifies the buffer memory address.



# LESSON 7 - Addressing

Controlling and monitoring I/O in a PLC program requires knowledge of the address of the point to be controlled. The same applies to Special Function Modules. This chapter explains how the addresses of the modules and I/O point in a system are determined

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Correctly address a discrete input or output point.
- ✓ Determine the address of a Special Function Module.
- ✓ Describe the I/O limitations of an FX3U system

**Materials:** FX-Series PLC Training Manual

## **7.1 Right Side Bus Rules of Addressing**

- 1) Addressing of inputs and outputs is in Octal (X0-X7, X10-X17, etc.)
- 2) Addressing for both inputs and outputs start at 0 (X0 and Y0)
- 3) Addressing is consecutive
- 4) SFMs are addressed as Blocks 0 – 7. The first SFM encountered to the right of the processor is block 0; the next one is block 1, etc.
- 5) A maximum of 8 SFMs can be connected to a main unit
- 6) SFMs do not affect the addressing of I/O modules, and vice versa.
- 7) FX3U PLC cannot have more than 128 inputs and 128 outputs directly connected. It is possible to extend to 384 I/O via network modules.
- 8) SFMs use 8 points of I/O each. This is deducted off the 256 I/O max. Thus an FX3U with one SFM has a maximum I/O capability of 248 I/O. The Maximum number of inputs possible is still 128, and the maximum number of outputs is still 128, as long as 248 combined I/O is not exceeded.

## **7.2 FX3U Left Side Adapter Bus Rules of Addressing**

The FX3U series adds a new adapter bus on the left side of the PLC. This bus has certain special adapter modules (ADP modules) which can be used to add functionality to the PLC.

There are 4 different types of modules available.

- Analog I/O Modules (maximum 4 per CPU)
- High Speed Pulse Input Module (maximum 2 per CPU)
- High Speed Pulse Output Module (maximum 2 per CPU)
- Serial Communications Module (maximum 2 per CPU or 1 with a BD board)

### **Analog Modules**

The data is directly input into PLC memory addresses. The modules are numbered out from the CPU to the left.

- 1<sup>st</sup> uses M8260-M8269 and D8260-D8269
- 2<sup>nd</sup> uses M8270-M8279 and D8270-D8279
- 3<sup>rd</sup> uses M8280-M8289 and D8280-D8289
- 4<sup>th</sup> uses M8290-M8299 and D8290-D8299

### **High Speed Pulse Input Modules**

High speed pulse input modules are numbered from the CPU out to the left.

- 1<sup>st</sup> unit uses X0, X1, X2, X6
- 2<sup>nd</sup> unit uses X3, X4, X5, X7
- High speed counters C235-C255 are used

Note that these are the same addresses used by the first 8 inputs on the PLC. Only the 4HSX module OR the built-in I/O should be used. Do NOT wire both.

### **High Speed Pulse Output Modules**

High speed pulse input modules are numbered from the CPU out to the left.

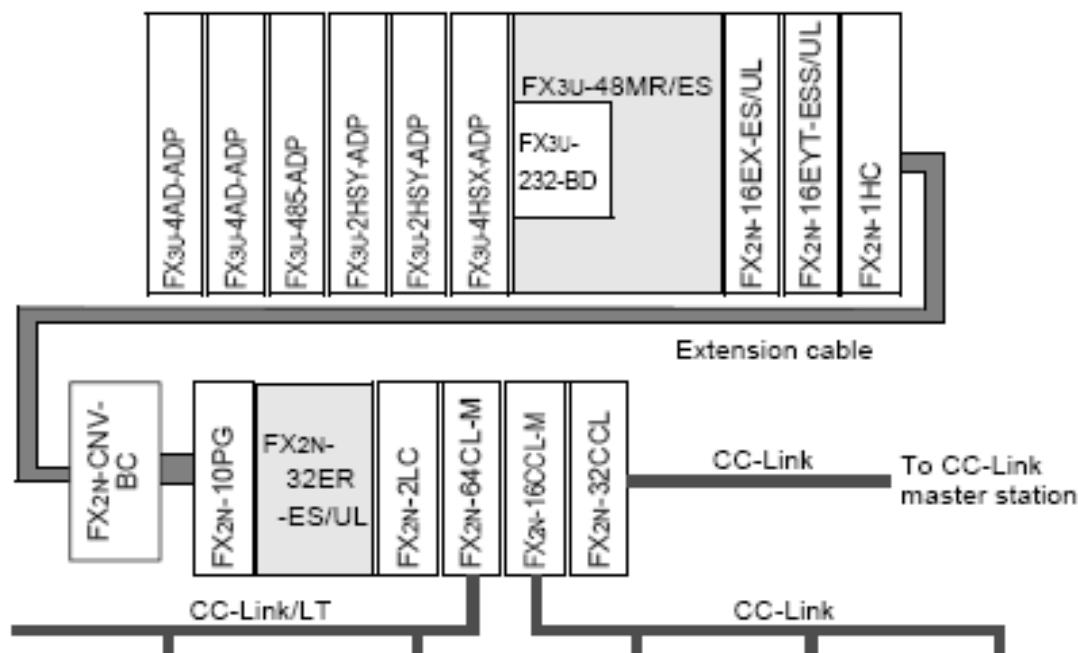
- 1<sup>st</sup> unit uses Y0, Y1, Y4, Y5
- 2<sup>nd</sup> unit uses Y2, Y3, Y6, Y7

Note that these are the same addresses used by the first 8 outputs on the PLC. Only the 2HSY module OR the built-in I/O should be used. Do NOT wire both.

### **Serial Communications Modules**

Serial communication modules do not have input or output addresses. Programming information for these modules is covered in the FX Communications Manual, part number JY997D16901.

## 7.3 Example



FX3U-48MR/ES      X00-X07, X10-X17, X20-X27  
                          Y00-Y07, Y10-Y17, Y20-Y27

Left Side Bus:      from the CPU out to the left

FX3U-232-BD board not addressed  
 FX3U-4HSX-ADP      X00, X01, X02, X06  
 FX3U-2HSY-ADP      Y00, Y01, Y04, Y05  
 FX3U-2HSY-ADP      Y02, Y03, Y06, Y07  
 FX3U-485-ADP not addressed  
 FX3U-4AD-ADP      M8260-M8269, D8260-D8269  
 FX3U-4AD-ADP      M8270-M8279, D8270-D8279

Right Side Bus:      from the CPU out to the right

FX2N-16EX-ES/UL      X30-X37, X40-X47  
 FX2N-16EYT-ESS/UL      Y30-Y37, Y40-Y47  
 FX2N-1HC      SFM #0  
 FX2N-10PG      SFM #1  
 FX2N-32ER-ES/UL      X50-X57, X60-X67  
                          Y50-Y57, Y60-Y67  
 FX2N-2LC      SFM #2  
 FX2N-64CL-M      SFM #3  
 FX2N-16CCL-M      SFM #4  
 FX2N-32CCL      SFM #5

## 7.4 EXERCISE      PLC Addressing

1) A PLC system consists of an FX3U-64MR (32 I/ 32 O), 1 FX3U-4AD-ADP, an 8 point input module, 2 16-point output modules, 2 SFMs, a 16 point input module, and another SFM. Determine the addressing.

2) ARE THE FOLLOWING SYSTEMS LEGAL? WHY OR WHY NOT?

- A. A 64 I/O main unit (32 I/32 O), [4] 8 point input modules, [6] 8 point output modules, and [9] SFMs.
- B. A 128 I/O main unit (64 I/64 O), [2] SFMs, [2] 48 I/O Extension Units (24 I/24 O each), [1] 16 point input module
- C. A 128 I/O main unit (64 I/64 O), [3] SFMs, [2] 48 I/O Extension Units (24 I/24 O each), [1] 16 point input module
- D. An 80 I/O main unit (40 I/40 O), [2] 48 I/O Extension Units (24 I/24 O each), [4] 16 point output modules, [1] 8 point input module
- E. An FX3U 64 I/O main unit, [3] FX3U-4AD-ADP, [1] FX3U-4DA-ADP, [1] FX3U-4AD-PT-ADP, [2] FX3U-4HSX-ADP, [2] FX3U-485-ADP, [1] FX3U-USB-BD



# LESSON 8 – Demo Kit Layout

Now that all the necessary background has been covered, it is time to take out and set up the hardware. The following section explains the demo kit and provides a brief tour of its features.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Name the different parts of the kit.
- ✓ Know how the parts work together.

**Materials:** FX-Series PLC Training Manual  
FX3U Training Demo Kit



## 8.1 Addressing

This demo kit has an **FX3U-16MR/ES**, **FX3U-USB-BD**, **FX3U-4AD-ADP**, **FX3U-7DM**, and an **FX2N-5A**. The FX3U has 8 inputs, addressed X0 – X7, and 8 outputs, addressed Y0 – Y7.

The FX2N-5A, a 4 channel analog input and 1 channel analog output module, being the first special function module to the right of the main unit, is block #0.

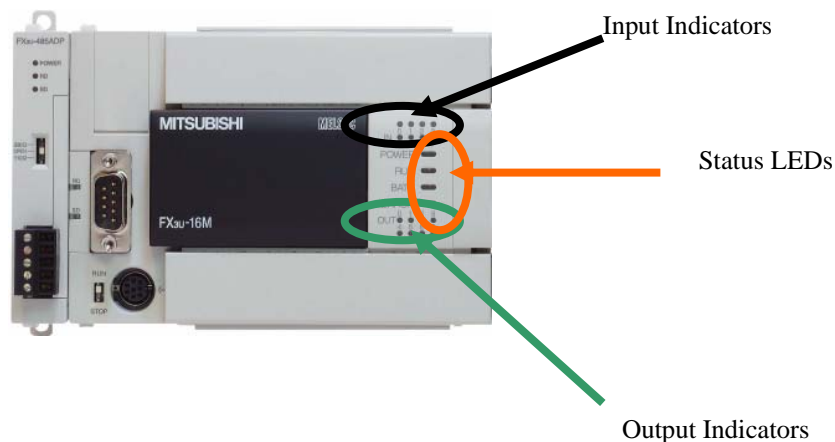
The FX3U-4AD-ADP is the first analog module to the left of the PLC. As such, it uses the addresses M8260 – M8269 and D8260 – D8269.

## 8.2 Indicator Lights

The PLC has indicator LEDs for its inputs and outputs. These lights are on the right side of the main unit, one set for inputs, one set for outputs. When X0 is true, LED 0 in the input section is on, when Y0 is one LED 0 in the output section is on, etc.

There are 4 LEDs on the far right side of the main unit. These are status lights.

- Top light indicates that power is being supplied to the PLC.
- The 2<sup>nd</sup> LED is on when the PLC is in RUN mode, off in STOP mode.
- The 3<sup>rd</sup> light comes on when the battery voltage is low
- The 4<sup>th</sup> LED has 2 purposes
  - When blinking indicates there is an error in the program
  - When on steady, indicates a CPU problem, such as removing the memory cassette while the unit was still powered.



## 8.3 Operator Interface

The exercises we will be doing will involve the use of the GOT as a means of PLC interface. There are screens loaded in the GOT to simulate discrete I/O, analog modules, and the various programming exercises planned for this class. As such, there are no hard-wired inputs and outputs on this trainer. All data required will be accessible from the GOT terminal screen.

# LESSON 9 – PLC Instruction Types

To write a program for a PLC, it is necessary to be familiar with the instructions that make up the PLC Instruction Set. An overview is provided here, with more detailed information to follow in later lessons.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Describe the 3 different types of instructions.

**Materials:** FX-Series PLC Training Manual

All three of these instruction types can be found in great detail in the FX Programming Manual for the appropriate model of PLC.

Part Number	Series Of PLC Covered
JY992D48301	FX, FX0, FX0S, FX0N, FX2C
JY992D88101	FX1S, FX1N, FX2N, FX2NC
JY997D16601	FX3U

## 9.1 Basic Program Instructions

This category refers to the 4 basic bit devices (X, Y, M, and S), the timer device, the counter device, and operation that pertain to these devices. This includes SET, RESET, and the PULSE functions. These instructions will make up about 80 ~ 90 percent of a program.

## 9.2 Step Ladder Instructions

Step ladder instructions are used in Step Ladder Programming (STL). This programming is similar to SFC programming in that it implements a flow chart, but the flow chart is implied, and the code is actually created in ladder logic. When programming in ladder logic, the STL contact is a common instruction used to check if a state is active. S relays are used to indicate states.

STL Programming is not typically covered in this class. However, an excellent explanation and examples can be found in Chapter 34.2 of the FX3U Programming Manual.

## 9.3 Applied Instructions

These instructions are the 'specialist' instructions of the FX line. These instructions allow the PLC to perform complex data manipulations, mathematical operations, and communications. Most applied instructions work on the 16 bit or 32 bit word level.



# LESSON 10 – Basic Instructions

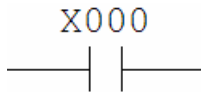
Basic instructions are bit control instructions, typically they make up 90% of the ladder program. They are used to confirm input status, manipulate outputs, bit shifts, and master control for nesting contacts.

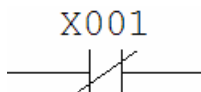
**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

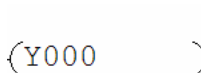
- ✓ Name the most common basic instructions.
- ✓ Know the format of the instructions and what they do.


**Materials:** FX-Series PLC Training Manual  
FX3U Programming Manual – Basic and Applied Instructions

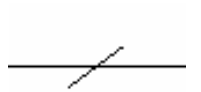
## 10.1 Symbols

 **X000** **NORMALLY OPEN CONTACT.** This symbol conducts when the associated device is energized. In instruction mode, the mnemonic is **LD**, which stands for LOAD. This symbol occupies 1 step of program space.

 **X001** **NORMALLY CLOSED CONTACT.** This symbol conducts when the associated device is de-energized. In instruction mode the mnemonic is **LDI**, for LOAD INVERSE. This symbol occupies 1 step of program space.

 **(Y000)** **COIL CONTROL.** This symbol always appears just before the right vertical ladder rail. It becomes energized when the logic before it conducts. When energized, the output with the same address becomes active. In instruction mode the mnemonic is **OUT**, for OUTPUT ACTIVATE. This symbol occupies 1 step of program space, unless being used for a timer or counter instruction, when it can occupy up to 5 steps.

 **[END]** **BRACKET CONTROL.** This symbol usually appears just before the right vertical ladder rail when used for bit control. This symbol is typically used for word device commands; however there are a few bit instructions that use the brackets as well. It becomes energized when the logic before it conducts. This symbol occupies multiple steps of program space depending on the command used.

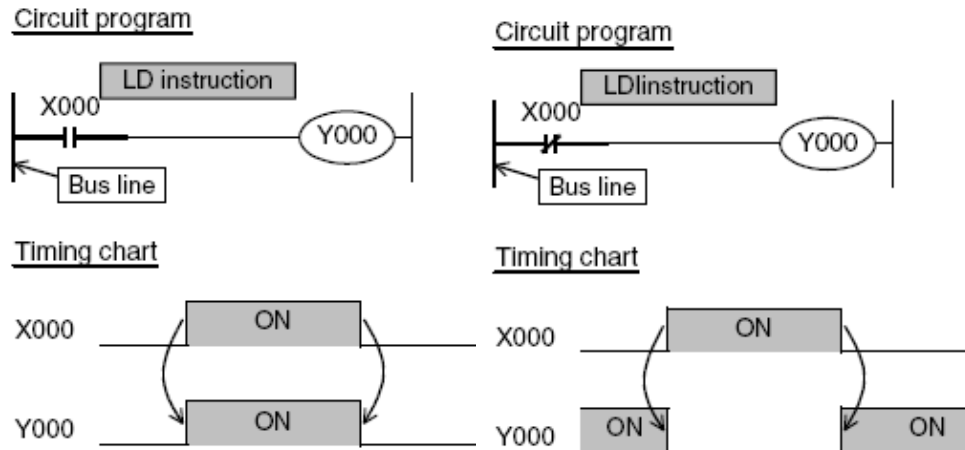
 **INVERT.** This symbol inverts the state of all logic before it. If the logic is true (positive) at the point of the invert, the output of the invert is false (negative). If the logic is false, the invert output is true.

It is important that the above concepts are clear before moving on. The symbols are used to indicate the device in its non-actuated state. When it's stated that a device is conducting, that means electricity is allowed to flow through.

For example, a light switch is usually in the off position (non-actuated), no current is flowing (the switch is open) until someone turns it on (actuates it and energizes it). At this point, electricity starts to flow (the switch is conducting) and the lights turn on. A light switch is a normally open contact.

The normally closed contact is the opposite in every respect. Current flows until the switch is actuated. A common example of this is an E-stop. It allows current to flow until an operator hits the switch in an emergency. The switch is actuated and the current flow stops.

### Example: Normally Open & Normally Closed Contacts



## 10.2 Ladder Basics

The alignment of contacts and coils determine how a rung of ladder logic is processed.

If two or more devices are positioned in series (one after another), they are operated as an AND instruction. In order for power to flow through the rung, all the conditions must be made true.



If the devices are in parallel, they are operated as an OR condition. OR conditions provide multiple paths for the power to flow.



AND/OR conditions can be combined to create complex rungs of logic.



Note: A rung must have an input condition, and an output to be a complete circuit. If a command is to be always on, there is a bit which can be placed in front of it, addressed M8000. This bit is always on. Connecting an output coil or bracketed instruction directly to the left power rail is not allowed.

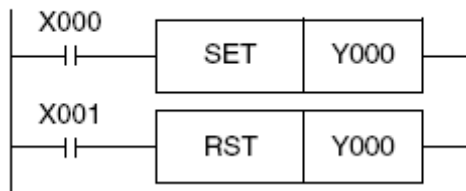
### 10.3 Common Instructions

**SET – Bit Set**

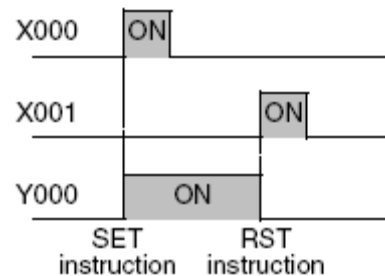
**RST – Bit Reset**

The **SET** instruction latches the device on. The **RST** (reset) instruction releases a latch.

Circuit program



Timing chart

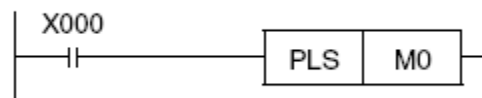


**PLS – Rising Edge Pulse**

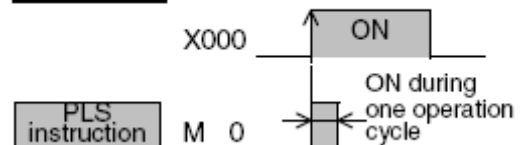
**PLF – Falling Edge Pulse**

The **PLS** instruction triggers on the positive edge of the input condition, while the **PLF** instruction triggers on the negative edge of the input condition. Both result in the **pulsing of the specified device ON for just 1 ladder scan**.

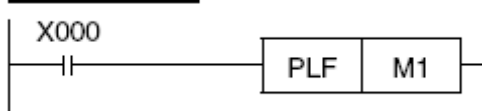
Circuit program



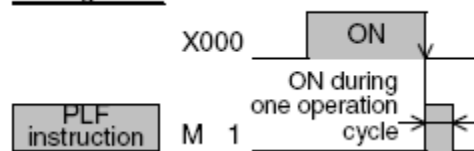
Timing chart



Circuit program



Timing chart



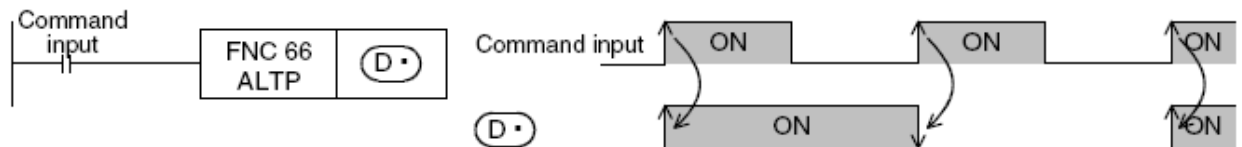
Another method for pulsing contacts is the rising edge and falling edge pulse bits. These bits activate only for one scan similar to the PLS and PLF instructions.

**X003 RISING EDGE PULSE.** The rising edge pulse is a one scan pulse based on the address shown above the contact. Instead of using X003 as an input and then a PLS instruction on an M bit, this command issues a one scan pulse to the logic following it.

**X004 FALLING EDGE PULSE.** The falling edge pulse is a one scan pulse based on the address shown above the contact. Instead of using X004 as an input and then a PLF instruction on an M bit, this command issues a one scan pulse to the logic following it.

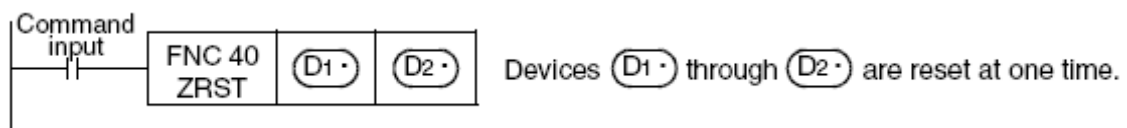
### ALT – Toggle Bit

The **ALT** command toggles the state of the specified bit. If the bit is on, it is turned OFF. If it off, the bit is turned ON.




### ZRST – Zone Reset

The **ZRST** (zone reset) command is used to reset a range of addresses. Rather than reset a single bit as RST does, the starting and ending addresses are specified. The range of addresses can be bits, words, timers, or counters. When word addresses are specified, they are reset to a value of 0. The second parameter (end address) must be higher than the first parameter (start address).





## 10.4 EXERCISE      Ladder Basics

- 1) X1 turns on and sets Y3. What happens to Y3 when X1 turns off?
  
- 2) What type of symbol would you use to represent a standard E-Stop in ladder logic? (Yes, E-stops are commonly hard-wired, but are often referred to in other parts of a program for various reasons)
  
- 3) List the common basic symbols (like ) and describe what they do.
  
- 4) List the common basic instructions (like **PLS**) and describe what they do.
  
- 5) What is required for a rung to be a complete circuit?
  
- 6) On the main conveyor, a sensor (input X2) checks for the presence of a certain package. When one is detected, the sensor switches on and turns on a pusher (output Y7). The pusher stays on until the package is detected by the side conveyor sensor (X3). When Y7 turns off, the pusher automatically retracts. Write the logic to accomplish this.

NOTE: As the package is pushed, it leaves the detection area of the main sensor before being detected by the side conveyor sensor.



# LESSON 11 – Develop & Edit Programs

Now is the time to put to work some of the knowledge that has been covered so far. In this section, we will open GX-Developer, write a simple program, download it to the PLC and test its operation, as well as investigate some of the tools in the GX-Developer software.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Launch GX-Developer.
- ✓ Enter instructions to write a small program.
- ✓ Transfer the program between the PLC and the laptop
- ✓ Do Online editing
- ✓ Monitor the Program
- ✓ Change values in the program with the software
- ✓ Monitor values in data registers

**Materials:** FX-Series PLC Training Manual  
FX-series Demo Kit

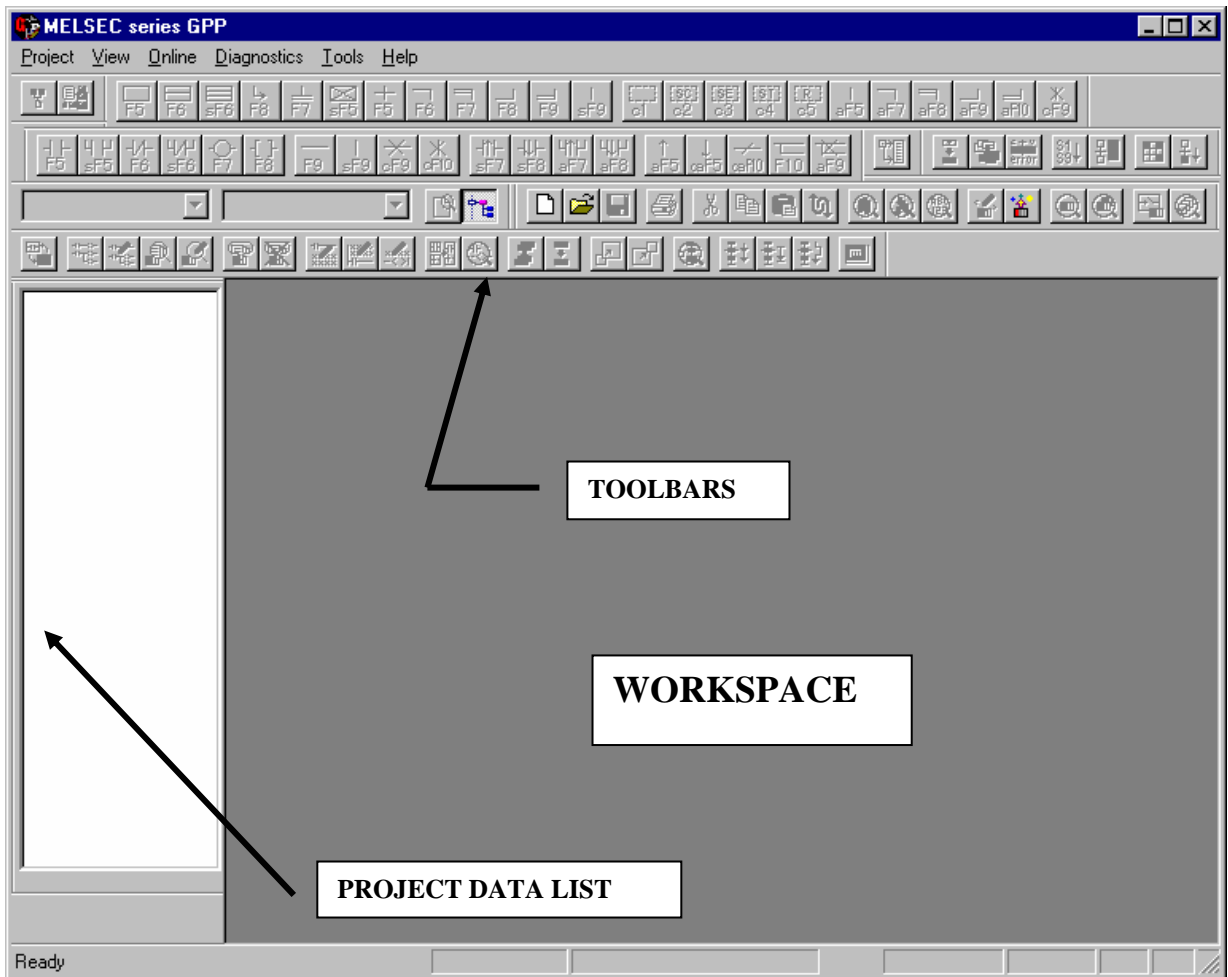
## **11.1 Launching GX-Developer**

GX-Developer is **WINDOWS based** ladder programming/monitoring software. The **serial port** is used to communicate between the PC and PLC. GX-Developer will run under Windows 95, 98, NT, and 2000.

GX-Developer can be started in one of 2 ways:

- 1) Double-clicking on the icon if one is present on your desktop or
- 2) Selecting it under the Start Menu. The default is Start -> Programs -> MELSEC Application -> GX-Developer for Windows

Once started, a screen similar to the one on the next page will appear:



The dark gray area is the workspace where the ladder logic will appear. Most of the toolbars that are open are but rarely used. These can be closed down to make the workspace larger:

- 1) Under View, select Toolbar
- 2) Check Standard and LD symbol, and uncheck the rest
- 3) Under View find Project Data List
- 4) Deselect Project Data List until required.

## 11.2 Creating a New Project

- 1) Three ways to create a new project
  - Under the Project menu, select New
  - Press Ctrl-N
  - Click the toolbar button for new document (blank white sheet of paper)
- 2) Select PLC series from dropdown menu. For this class select FXCPU.
- 3) Select PLC type from dropdown menu. For this class select FX3U(C).
- 4) If desired, a pathname and project name, as well as a short title can be defined now. These do not need to be set at this point. This information can be assigned when the project is saved later.
- 5) Press OK.

## 11.3 Editing the Ladder

We will now enter a basic ladder logic program. Follow the steps below to create this simple ladder program.

- 1) Click on the Normally Open (NO) Contact button
- 2) Type X10 into the dialog box that pops up and click OK
- 3) Double-click inside the placement box
- 4) The dialog box has a dropdown menu on the left. Select the Normally Closed (NC) symbol. Type X11 in the textbox and press OK.
- 5) Press the number 7 key. Enter Y0 in the textbox and press OK.

NOTE: The 7 key may not work depending on the selected keyboard layout. Under Tools menu, in Customize Keys, the keyboard shortcuts can be set to one of 3 standard sets. 7 is the key in MEDOC for a coil. In GPPQ and GPPA the key is F7. The default keyboard on a new installation of GX-Developer is GPPQ format. This can be changed to whichever keyboard layout a user is most familiar and most comfortable with.

You have just written an entire rung using each of the 3 ways to enter symbols. Now we will look at editing the program.

- 6) Click on the rung. Go to the Edit pull-down menu.
- 7) Select Delete Line. Rung disappears.
- 8) Return to Edit and Select Undo. Rung Reappears.
- 9) Right click on X10. Select Delete Row. X10 disappears.
- 10) Right click on X11. Select Delete Row. X11 disappears.
- 11) Right click on rung and select Undo. X11 reappears.
- 12) Right click on rung and select Undo.

**Notice that Undo is grayed out. There is only 1 level for Undo.**

Insert Rung and Insert Row add space for a new rung or new contact. Put X10 back into the rung.

Notice that the rung is highlighted in gray. This means that the rung doesn't yet exist in the program. It exists only in the program's scratchpad. Go to the Convert menu and select Convert (you can access convert by right-clicking in the workspace or by pressing the F4 shortcut key). The gray highlight disappears. If the logic drawn is not complete or has errors, the convert will fail with an error message. The software will not allow saving or downloading of programs with unconverted code in them.

Click on the View pull-down menu. Select Instruction List. The ladder logic diagram disappears and is replaced by abbreviations and the addresses:

```
0    LD  X10
1    ANI X11
2    OUT Y0
3    END
```

This is the ladder logic diagram written out in instruction list, which is the program format that the PLC actually understands. Ladder can be displayed again by going back to the View menu. Instead of Instruction List, Ladder is now displayed.

You should now save the program by clicking on the Save icon in the toolbar, or by pressing Ctrl-S. Enter the Project Name as FXPROG1. Click Yes on the dialog box to save a copy of your project.

## **11.4 Program Transfer**

To transfer a program to the PLC, the PLC must be stopped. This can be done by setting the key switch on the CPU to the STOP position, or by the software. If the PLC is in RUN, a remote stop can be performed. If an attempt is made to download to a PLC which is running, the software will automatically prompt to remote stop the PLC, and then it will download the program, and prompt to remote start the PLC.

- 1) Go to the Online menu and select Write to PLC.
- 2) Select Param + Prog.
- 3) Click on Execute.

A progress bar appears to show the program download process. A dialog box pops up to indicate the download is finished. Press OK. Return the PLC to RUN mode.

Two other options in the Online menu are Read from PLC and Verify.

Read from PLC uploads the program from the PLC and displays it in GX-Developer. This would be used so that you can make changes to the program in the PLC.

- 4) Select Read from PLC.
- 5) Select Param + Prog.
- 6) Click on Execute.
- 7) When done, click on OK and then close the box.

Verify with PLC compares the program that is open in GX-Developer with the program in the PLC. This is especially useful in an environment where several employees have the ability to make changes to a program. This prevents the programmer from inadvertently writing over previous changes made by a co-worker

- 8) Select Verify with PLC.
- 9) Select Param + Prog.
- 10) Execute.
- 11) Click on the X in the right hand corner to close the verify screens.

GX-Developer will do a comparison and list any unmatched items.

## Special Shortcut

It may appear that this very small program takes a long time to download. This is because no matter how many ladder logic steps are in the program, GX-Developer always downloads at least 8000 steps for the FX2N. It is possible to speed the process considerably. Be careful if using this function to ensure all code including the END statement is downloaded.

- 1) Take note of the final step number next to the rung with the END statement.
- 2) Go to the Online menu and select Write to PLC.
- 3) Select Main.
- 4) Click on Program tab.
- 5) Click Step Range Specification.
- 6) In the End textbox, enter the final step number.
- 7) Click on Execute.

## 11.5 Online Editing

So far we have written a small program and then downloaded it to the PLC. Since the program change was done in the computer only, this is called Offline Editing. If the computer is connected to the PLC it is possible to change the program directly in the PLC, avoiding the need to download or stop the PLC. This is called Online Editing.

Note: FX PLCs prior to the FX2N must have RAM memory to do online editing. The FX2N and above can do online editing to RAM and EEPROMs.

- 1) Click on the Online pull down menu
- 2) Go to 'Monitor'
- 3) Click on Monitor(Write Mode)
- 4) Change the rung so that it looks like the one below



Note: You will need to convert the rung that you create.

At this point the new program exists in the PLC, but not on your hard drive. This is a good time to save your project.

The circuit you created is called a **LATCH and HOLD** circuit and is very common. X10 is a momentary contact, such as a pushbutton, that starts a machine. Without the branch, the machine would only run as long as the pushbutton was held down. Now Y0 turns on and keeps the branch true, even though X10 is off. It is necessary to toggle the stop button, X11, to turn the machine off.

## 11.6 Monitor the Program Operation

It is possible to view what's happening in the program, and to check the states of program bits, in GX-Developer. This process of viewing is called **Monitoring the program**.

- 1) Click on the 'Online' pull down menu
- 2) Go to 'Monitor'
- 3) Select 'Monitor Mode'

A small box will pop up, indicating the mode (run or stop) of the PLC and the average scan time for the program.

You should notice that X10 and Y0 are not highlighted, and X11 is highlighted. This indicates whether an input device is conducting or not, or an output device is energized. A highlighted contact is conducting, and a highlighted coil is energized. X10 and Y0, which are checking to see their input and output respectively are on, aren't true. X11 on the other hand, which is checking the input to see that it is off, is true.

Turn X10 on by toggling the switch. When the switch is toggled on, the bit in the program highlights. Toggle X11 off. Notice that the bit is not highlighted anymore. Set X11 and X10 so that both are highlighted. When all the contacts on a rung are highlighted, we say that the rung is **TRUE**. When a rung is true, the output turns on. Notice that Y0 is highlighted on the screen and that the Y0 bulb is lit on the trainer.

### **Entry Ladder Monitor**

This feature allows the programmer to monitor multiple rungs in a non-sequential manner.

- 1) Copy and Paste the above rung 3 times. Change the addresses of the contacts and coils to create 4 separate rungs
- 2) Put the program into Monitor Mode
- 3) Go to Online → Monitor → Entry Data Ladder
- 4) Go to Window → Tile Horizontally
- 5) In the bottom window click on the 4<sup>th</sup> rung to highlight it
- 6) Click and hold on the highlighted rung, drag it to the upper screen and release
- 7) Repeat steps 5 and 6 for the 1<sup>st</sup> rung and the 3<sup>rd</sup> rung.
- 8) Click on the upper window to make it the active window and maximize it
- 9) Put the window into Monitor Mode
- 10) Toggle switches and watch the results



## 11.7 Forcing Bits and Changing Registers

It can often be helpful to run sections of PLC code while writing a program. This allows the programmer to test parts of the code while the program is small enough to make changes easily. This can be done without the use of switches and other devices; all that's required is the PLC and GX-Developer. This is called **FORCING**.

- 1) Put the PLC in Run Mode
- 2) Open program and put GX-Developer into Monitor (Write) mode
- 3) Holding the Shift Key down, double click on X10
- 4) After observing the change, do a shift and double click again.

Notice that Y0 comes on, while X10 is off. In the FX-Series, real world inputs can only be forced on for a single scan. Real world outputs that are used in ladder logic can only be forced on for one scan as well. Any address with a physical input will revert back to its real world input state at the beginning of the next PLC scan when the physical inputs are read. Outputs used in the PLC code will revert to the program controlled output state upon next PLC scan. Internal bits, like M relays, can be forced on and stay on, as long as they are not being controlled by the program in the PLC.

This is the easiest way to turn contacts and relays off and on. This isn't recommended when the PLC is connected to a running system, however. There is no message that warns that the change is about to happen, and dangerous results could occur.

- 1) Go to Online → Monitor → Entry Data Monitor → Device Test
- 2) Enter X10 into Device textbox in the Bit Device section
- 3) Click Force ON

It is possible to enter numbers into data registers through this dialog box as well.

- 1) Enter D0 into the Device textbox in the Word Device section
- 2) Enter 10 into the Setting Value textbox
- 3) Click the Set command button

Check to see that a 10 has been entered into data register D0.

- 1) Go to Online → Monitor → Device Batch Monitor
- 2) Enter D0 into the Device textbox
- 3) Click on the Start Monitor button

## 11.8 EXERCISE      **Contacts and Coils**

Please find Project #1 in the appendix. This project is intended to give the student practice in entering and controlling ladder logic.



# LESSON 12 – Timers and Counters

Timers and counters are a standard part of a PLC program. This section will cover the various types of timers and counters available in the FX-Series PLCs as well as how to code them. Exercises will allow the user to demonstrate their understanding of the concepts.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Describe the different types of timers
- ✓ Know the availability of timers and counter
- ✓ Describe the format for timer and counter instructions
- ✓ Describe timer and counter limitations
- ✓ List the types of presets available to timers and counters
- ✓ Write a program using timers

**Materials:** FX-Series PLC Training Manual  
FX-series Demo Kit  
FX3U Programming Manual – Basic and Applied Instructions

## 12.1 Timers

### Availability

FX3U PLCs have 512 timers.

FX1N, FX2N, and FX2NC PLCs have 256 timers.

FX1S PLCs have 64 timers. The time base is dependent upon the address used in the timer instruction. In the FX1S, a special relay bit (address M8028) can be set to convert 31 of the 100ms timers to 10ms timers.

### Types

- 100ms (.1 second)
- 10ms (.01 second)
- 1ms (.001 second)

Time Base	PLC Timer Address		
	FX1S	FX1N/FX2N/FX2NC	FX3U
100ms	0-62	0-199	0-199
10ms	32-62	200-245	200-245
1ms (Retentive)	-	246-249	246-249
100ms (Retentive)	-	250-255	250-255
1ms	63	-	256-511

## Presets

Preset is the length of time the timer runs before finishing. The preset indicates units of time bases. Thus T0 with a value of 50 runs for 5 seconds (50 x .1 seconds = 5 sec).

The preset must be a number between 1 and 32,767, because timers are 16 bit registers.

Timer presets can be either a K constant, or a variable, such as a data register. Having a D device as the preset allows an operator to make changes to the timing duration from an HMI, or allows the program to change the preset based on the ladder logic.

Timers only time up.

## Reset (Retentive)

The accumulated value of a timer returns to 0 when the input conditions of the timer rung become false. This is not the case with retentive timers. To return a retentive timer accumulated value to 0 it is necessary to use the RST T# instruction.

Non-retentive timers will lose their accumulated values at power down unless they have been declared battery-back in PLC parameters.

## 12.2 Counters

### Availability

FX2N, FX2NC, and FX3U PLCs have 256 counters.

FX1N PLC has 256 counters.

FX1S PLCs have 45 counters.

Counter Type	PLC Counter Address		
	FX1S	FX1N	FX2N/FX2NC/FX3U
16-Bit	0-15	0-15	0-99
16-Bit Latched	16-31	16-199	100-199
32-Bit Bi-directional	-	200-219	200-219
32-Bit Bi-directional Latched	-	220-234	220-234
High Speed Counters	235-254	235-255	235-255

Note: Although counters C235 to C255 (21 points) are all high speed counters, they share the same range of high speed inputs. Therefore, if an input is already being used by a high speed counter, it cannot be used for any other high speed counters.

Note: FX1S does not have C239, C240, C243, C245, C248, C250, or C253.

## **16 bit counters**

### **Presets**

Presets are the number of times the rung driving the counter has to go through a FALSE to TRUE state transition before turning on.

16 bit counters have a range of 1 to 32,767.

Counter presets can be either a K constant, or a variable, such as a data or file register. Having a D device as the preset allows an operator to make changes to the counter preset from an HMI.

The accumulated value of the timer never goes above the preset value. Once the counter coil has turned on, it will remain on until reset. Even the use of the Decrement instruction to reduce the count will not deactivate the counter coil.

### **Counting direction**

16 bit counters only count up.

### **Reset**

The accumulated value of a counter returns to 0 when the RST C# instruction is activated.

Counters addressed from C100 ~ C199 are latched counters and retain their count even at power down. C0 ~ C99 will lose their counts at power down unless they have been declared as battery-backed in PLC Parameters

### **Limitations**

Counter negative number presets are not permitted.

The only way to make a counter count down is to use the DECP instruction *prior to the counter reaching its preset*. Once the counter coil is turned on, the only way to turn it off is to use the RST instruction. The counter will appear to count down, but the counter coil will not turn off.

Using the INCP instruction will make the counter count up and count pass the preset. However, the counter coil will not be activated by reaching the preset with the INCP instruction. The final count must come from a FALSE to TRUE transition of the counter rung.

## **32 bit counters**

### **Presets**

Presets are the number of times the rung driving the counter has to go through a FALSE to TRUE state transition before turning on.

32 Bit counters have a range of –2,147,483,648 to 2,147,483,647.

Counter presets can be either a K constant, or a variable, such as a data or file register. Having a D device as the preset allows an operator to make changes to the counter preset from an HMI, or allows the preset to be changed during operation of the logic.

The accumulated value of the 32 bit counter can exceed the preset value, or go below 0. Once the counter coil has turned on, it will remain on until reset, or the counter counts back down. Using the Decrement instruction to reduce the count will not deactivate the counter coil.

### **Counting direction**

32 bit counters can count up or down. Counting direction for counter C### is based upon the status of relay M8###. If M8### is off, the counter counts up. If M8### is on, the counter counts down.

i.e. For C201, direction is determined by M8201.

### **Reset**

The accumulated value of a counter returns to 0 when the RST C# instruction is activated.

32 bit counters addressed from C220 ~ C234 are latched counters and retain their count even at power down. C200 ~ C219 will lose their counts at power down unless they have been declared as battery-backed in PLC Parameters

### **Limitations**

Using the INCP instruction will make the counter count up and count pass the preset. However, the counter coil will not be activated by reaching the preset with the INCP instruction. The final count must come from a FALSE to TRUE transition of the counter rung.

If a 32 bit counter has a negative preset, the logic works somewhat differently than what may be expected. If the counter coil will activate only when the preset is reached by counting **UP** to it from a **SMALLER** number

i.e. Counter C200 has a preset of –10. The as the counter counts down from –9 to –10, the associated coil will not turn on. If the counter counts down to –11 and then counts up to –10, then the coil will turn on.

## 12.3 Program Examples

Start a new program and enter the following timer delay circuit.



When developing logic, try to use the words **AND** and **OR**, for example ...

When X12 **AND** X13 are on, **OR** when X14 is on, then timer T0 should start timing.

**TIMERS by default are non-retentive.** That means that they do not hold their value if the input circuit opens. Close the X14 switch and monitor the timer value (at the bottom of the monitor screen). When X14 opens, notice the timer value returns to zero.

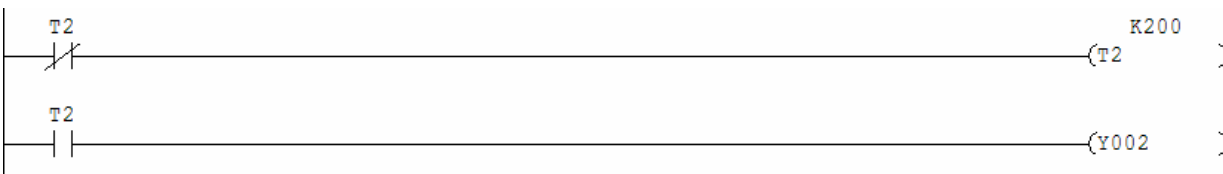
Timers T0~T199 are 100 millisecond timers, T200~T245 are 10 millisecond timers, for this CPU. The K value is a multiplier. K40 means 40 x 100 milliseconds = 4 seconds. After the timer reaches 40, the T0 contact will close.

Add the following logic.



Turn ON X12 **AND** X13. When the timer reaches 40 then the T0 contact will conduct, turning on the output Y1.

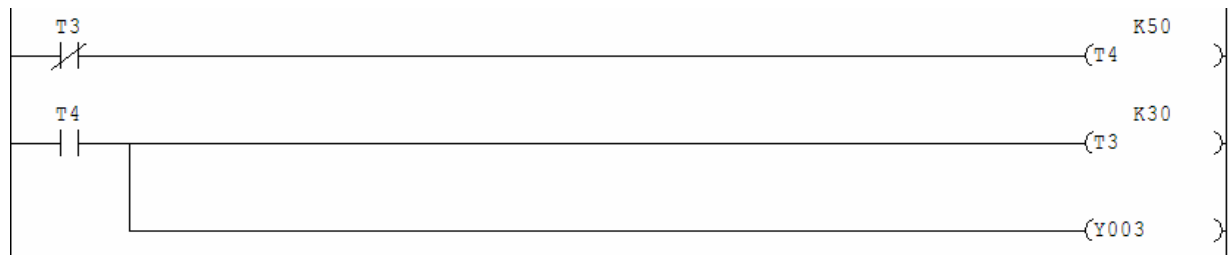
Here is a pulse timer.



When the PLC starts running, the T2 normally closed contact is conducting. This causes the T2 timer to start counting up to 20 seconds (T2=100msec timer). When the timer reaches 20 seconds the coil becomes active, which activates the corresponding contacts. Since this example uses a normally closed contact, when the coil activates the contact opens, automatically resetting the timer. When the timer resets, the T2 coil turns off, causing the normally closed contact to conduct, which starts the timer counting again.

The result is the T2 normally open contact will conduct for 1 ladder scan every 20 seconds creating a timed pulse.

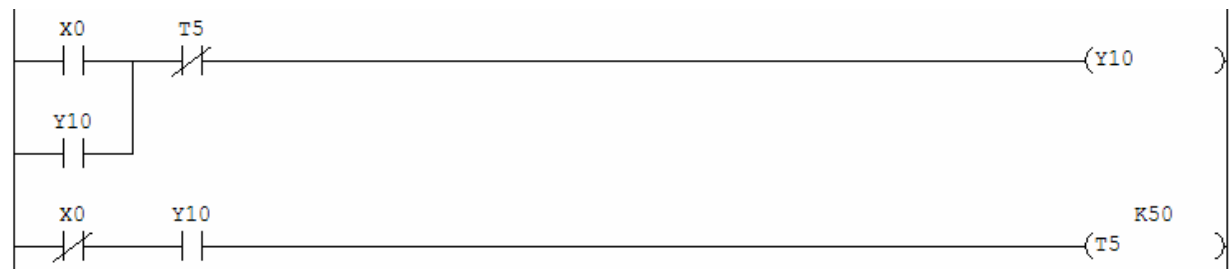
Here is a Flip-flop circuit.



Initially the T3 contact conducts, T4 coil counts up to 5 seconds. When T3 completes, T4 coil comes on. T4 contact conducts, causing the T3 coil to count up to 3 seconds. At 3 seconds the T3 coil becomes active, opening the T3 contact, which then resets T4 coil.

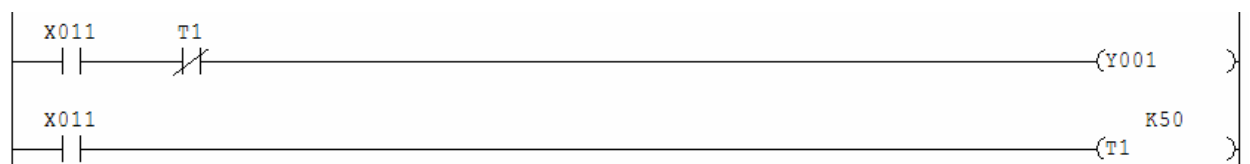
The result is Y3 will be OFF for 5 seconds and ON for 3 seconds.

Another useful timing circuit is an off-delay timer.



In this example, X0 is our run signal. Once X0 is on, we want to turn on Y10, and we want Y10 to remain on for 5 seconds after X0 turns off.

Sometimes, an output needs to be on for a specific amount of time, no matter how long the input condition stays on. This is typically called a one-shot timer.



In the example above, input X11 starts the output Y1. Once the output is energized, a timer starts. Once the timer has completed, it turns off the output. The output will not turn on again until after the input X11 has turned off and the timer is reset.



A quick calculation shows that the longest time duration that can be handled by a timer is  $(32,767 \times .1 \text{ sec} / 60) = 54.36$  minutes. What happens if it is necessary to run a timer longer than this? The answer uses a combination of timers and counters together like the program below:



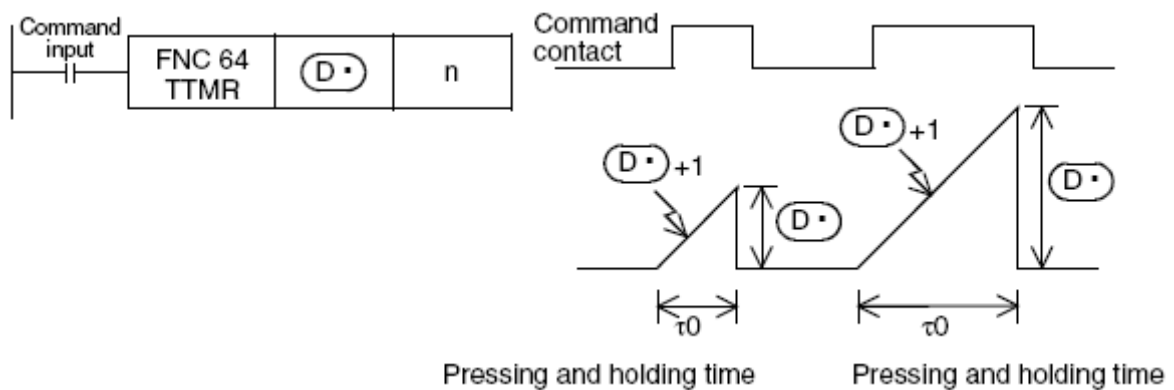
T0 runs for 1 minute. After 1 minute, T0 causes C0 to increment. After 60 increments, or 1 hour, C1 increments. After 24 hours C1 would go true and set a day counter.

Note: The program shown above is not complete. What needs to be added to make it work properly?

## 12.4 Additional Timer Commands

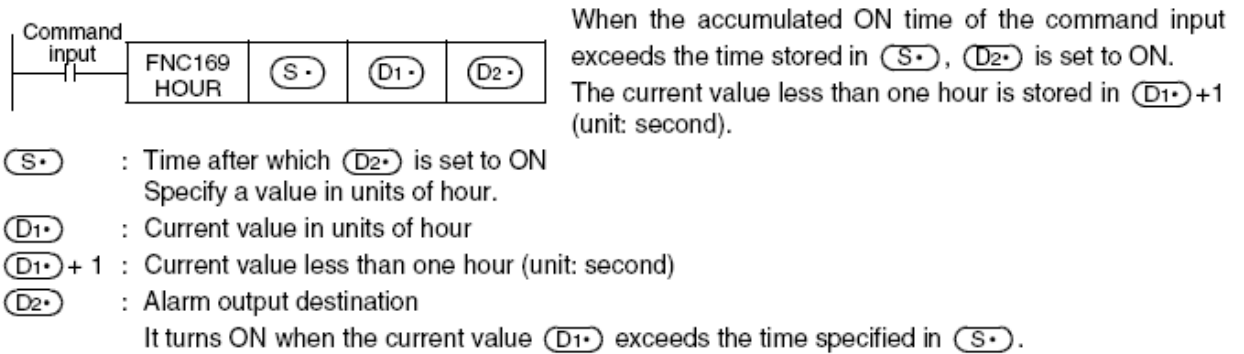
There are some other handy timer instructions within the FX Series command set. These are a teaching timer and an hour meter command.

The **TTMR** command measures the amount of time in seconds which its input condition is on. It amplifies this value based on a magnification setting and stores this value in a data register. Once in the data register, it can be used as a preset to another timer instruction or in any word commands.



The magnification takes the number of seconds and multiplies it by  $10^N$ , where  $n$  is 0, 1, or 2. To store the actual number of seconds, the magnification will be K0. To make a preset for a 100msec timer, you would enter K1 in the magnification parameter. This will take the number of seconds and multiply it by 10 to get the number of 100msec increments. For a 10msec timer, the number of seconds would need to be multiplied by 100, or  $10^2$ . The magnification would be K2.

The **HOUR** instruction is a built-in hour meter. The function times the number of seconds the function has been active. It allows the operator to set a value for the number of hours which will turn on the output indicated in the final parameter.



It is recommended that the addresses used to store the current value in hours and seconds should be in the retentive range in the PLC so they are not lost when the PLC is powered off or reset. Timing will continue until the maximum number is reached in the data register. This function can be coded as DHOUR to use 32-bit registers and store longer time frames.

## 12.5 EXERCISE Timers and Counters

Please find Project #2 in the appendix. This project is intended to give the student exposure to entering timers and counters, as well as exposure to timer/counter behavior.

## 12.6 EXERCISE Conveyor Control

Write a program that uses:

X10 as the Start button (momentary contact)  
 X11 as the Stop button (momentary contact)  
 M0 is the latch contact (remain on for entire cycle, shut off after last conveyor off)

When the start button is pushed, turn on outputs Y0 to Y7 in sequential order. These outputs are 8 conveyors that must be turned on in order. Each conveyor comes on 1 second after the previous conveyor comes on. When all conveyors have been running for 5 seconds, turn the conveyors off, in the reverse that they were turned on, one at a time, one second apart.

# LESSON 13 – Applied Instructions

These instructions are the ‘specialist’ instructions of the FX line. These instructions allow the PLC to perform complex data manipulations, mathematical operations, and communications. Most applied instructions work on the 16 bit or 32 bit word level.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Name the most common applied instructions.
- ✓ Describe the format of the instructions and what they do.
- ✓ Write a program using comparison statements

**Materials:** FX-Series PLC Training Manual  
FX-Series Demo Kit

## 13.1 General Format

Applied instructions are always drawn with the bracket symbol. The number of parameters varies among the various instructions. Most applied instructions follow the following format:

—[ADD      SRC1      SRC2      DEST      ]

Example

—[ADD      D0      K2      D300      ]

This instruction adds the contents of source 1 (D0) to the contents of source 2 (K2) and places the result in D300.

It is possible to use a source as the destination:

Example

—[ADD      D0      D300      D300      ]

In this case, the second source and the destination are the same register. If D0 contains 9 and D300 contains 200 prior to the execution of the instruction, then when the instruction is executed, 9 is added to 200, and the result of 209 is placed in D300.

Applied instructions by default are 16 bit instructions. If 32 bit data manipulation is desired, it is necessary to add a “D” to the front of the instruction.

Example:     **MOV** transfers 16 bits of data  
              **DMOV** transfers 32 bits of data

Most of these instructions continue to execute as long as the input conditions are TRUE. Sometimes this is not what is wanted. Where the programmer may want to increment a data register (with **INC**) by 1 when an input is made, INC will increment every scan as long as the input is on! This could be hundreds of times every second!

To avoid this, it is possible to set up basic instructions so they execute only once, on the FALSE to TRUE state change. To do this, add a “P” to the end of the instruction.

Examples:

      / **D0 D1 D20** divides D0 by D1 and places the result into D20 every scan  
      / **P D0 D1 D20** does the division once when the input conditions are TRUE

## 13.2 Data Transfer Instructions

The data transfer instructions must be placed in the end of the rung. When the input criterion is true, the data transfer instruction is performed.

Move Function	16-Bit Data	32-Bit Data	Floating Point
1 to 1	MOV	DMOV	DEMOV
Many to Many	BMOV	-	-
1 to Many	FMOV	DFMOV	-

**MOV – Data Move**

**DMOV – 32-Bit Move**

**DEMOV – Floating Point Move**

The **MOV** move command moves data from the source (D1) to the destination (D2). The command actually copies the data, so after the command is executed, both registers contain the same data. Two variants of this command are **DMOV**, which moves a 32-bit value, and **DEMOV**, which moves a 32-bit floating point value.



## BMOV – Block Move

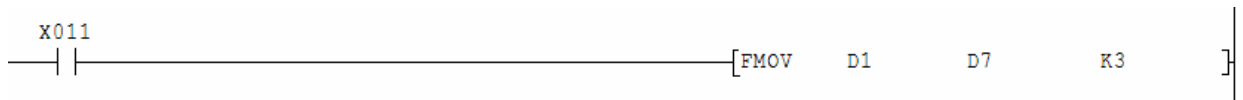
The **BMOV** block move command moves data 3 words of data starting from the source (D1) to the 3 words of data starting at the destination (D7). The command actually copies the data, so after the command is executed, registers D1-D3 contain the same data as registers D7 – D9. This command is not available specifically for 32-bit data, so the number of registers to move should be doubled. Since it moves raw data, there is no difference in the formatting.



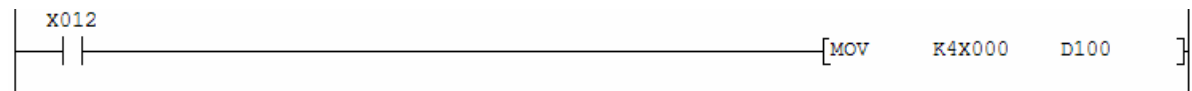
## FMOV – Fill Move

### DFMOV – 32-Bit Fill Move

The **FMOV** fill move command moves the data that is in source (D1) into the 3 words starting at the destination (D7). The command actually copies the data. If D1 held the value 13, then D7, D8, D9 all contain the value 13 after the execution of this command. The **DFMOV** command uses 32-bit numbers, so the source is D0+D1, and the destinations are D7+D8, D9+D10, and D11+D12.



For any of the move commands, a group of bits can be moved to a word using a prefix of Kx, where X is a number between 1 and 8. This command below will move 16 bits (4 nibbles of 4 bits each) into the 16-bit register D100.



The **K value** determines how many bits will be converted.

K1 = 4 bits

K2 = 8 bits

K3 = 12 bits

K4 = 16 bits

K5 = 20 bits

K6 = 24 bits

K7 = 28 bits

K8 = 32 bits

## 13.3 Comparison Instructions

A variety of commands exist to compare one piece of data to one or a range of values. These commands are available for 16-bit, 32-bit, and floating point data.

Compare Function	16-Bit Data	32-Bit Data	Floating Point
One to One	CMP	DCMP	DECMP
One to Range	ZCP	DZCP	DEZCP

**CMP – Compare**  
**DCMP – 32-Bit Compare**  
**DECMP – Floating Point Compare**

The **CMP** compare instruction must be placed at the end of the rung. Its destination is a group of 3 bit devices, which can be Y outputs, M relays or S relays. The 3 result bits tell whether source data 1 is greater than, less than, or equal to source data 2.



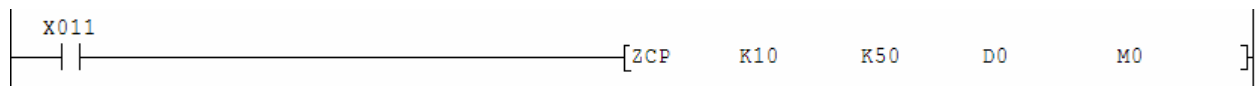
The instruction turns on M0, M1 or M2 based on the following criteria:

- M0 if the value in D0 is greater than K10
- M1 if the value in D0 equals K10
- M2 if the value in D0 is less than K10

Two variants of this command are **DCMP**, which compares two 32-bit values to, and **DECMP**, which compares two 32-bit floating point values.

**ZCP – Zone Compare**  
**DZCP – 32-Bit Zone Compare**  
**DEZCP – Floating Point Zone Compare**

The **ZCP** zone compare instruction must be placed at the end of the rung. Its destination is a group of 3 bit devices, which can be Y outputs, M relays or S relays. The 3 result bits tell whether source data 3 is greater than, less than, or within the range specified by source data 1 and 2.



The instruction turns on M0, M1 or M2 based on the following criteria:

- M0 if the value in D0 is less than K10
- M1 if the value in D0 is  $K10 \leq D0 \leq K50$
- M2 if the value in D0 is greater than K50

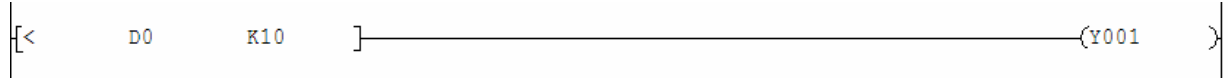
The zone compare takes S3 (D0) and compares to the range specified by S1 (K10) and S2 (K50). The instruction checks to see if S3 is below the range, inside the range, or above the range.

S1 needs to be less than S2. If it is not, the PLC will still accept the instruction, but it will not function properly.

Two variants of this command are **DZCP**, which compares a 32-bit value to 32-bit values, and **DEZCP**, which compares a 32-bit floating point value to two 32-bit floating point values.

## INLINE COMPARISONS

Unlike all the function blocks discussed so far, the inline compare instructions can be placed anywhere within the rung. They function in the same way as an input contact. These instructions are only available on the FX1S, FX1N, FX2N(C) and FX3U PLCs. These commands were not available on legacy FX PLCs.



If the data register (D0) is less than the decimal constant (K10), the circuit conducts.

Comparison Function	16-Bit Data	32-Bit Data
Greater Than	>	D>
Equal To	=	D=
Less Than	<	D<
Greater Than Equal	>=	D>=
Not Equal	<>	D<>
Less Than Equal	<=	D<=

Imagine the operator is between Source 1 and Source 2. The above instruction actually reads: Check if D0 < K10 (the value in D0 is less than 10)

The inline comparison instructions support a prefix of **D** so any of the comparisons can be done on 32-bit numbers.

It is possible to look at the accumulated value of a timer or counter and do a comparison. This allows you to turn on different outputs at different times or count values, but only using one timer or counter. The rung below shows how to turn on output Y1 when T10 has reached 6 seconds.



### Further note on all compares

The compare, zone compare and inline compare instructions shown above are 16 bit instructions. As described at the beginning of the chapter, if a 32 bit comparison is required, it is necessary to add a "D" to the instruction, this we have **DCMP**, **DZCP**, and for inline compares, **D=**, **D>**, etc.

This is crucial to remember in the case of up/down and high speed counters. These counters are 32 bit instructions and will not function properly with a standard comparison instruction.

### 13.4 EXERCISE      Parking Lot

Write a program for the following parking lot control application:

X10 indicates a car coming in

X11 indicates a car going out

Y0 is a sign which turns on to indicate the lot is full

C200 (bi-directional 32-bit counter) will keep track of the number of cars in lot

D0 will store the maximum number of cars (for this exercise write 10 to D0)

When a car comes in, count that car. When a car goes out, reduce the current count by 1. When the lot is full we need to turn on the Lot Full sign.

### 13.5 EXERCISE      Conveyor Control Part 2

Rewrite the Conveyor Control program, using ONLY ONE timer and using compare statements to control the outputs.

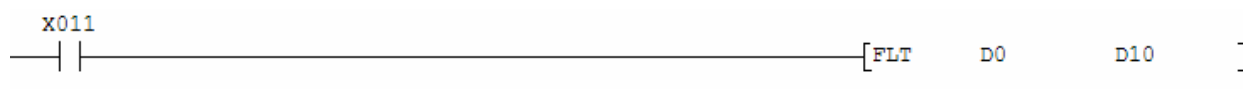
### 13.6 Conversion Instructions

The conversion instructions are placed at the end of the rung. Commands are available to convert data to and from many numerical formats and ASCII strings. The ones which will be covered in this class convert integer numbers to floating point numbers and back.

**FLT – Integer to Floating Point**

**DFLT – 32-Bit Integer to Floating Point**

The **FLT** command converts from a 16-bit integer to a 32-bit floating point number. **DFLT** would convert a 32-bit integer to a 32-bit floating point number.



The command above would convert an integer value stored in D0 to a floating point number and use D10 and D11 to store this number.

**INT – Floating Point to 16-Bit Integer**

**DINT – Floating Point to 32-Bit Integer**

The **INT** command converts from a 32-bit floating point number to a 16-bit integer number. Decimal places are lost, only the integer portion is converted. **DINT** would convert a 32-bit floating point number to a 32-bit integer number.



The command above would convert a floating point number stored in D10 and D11 and store it as a 16-bit integer in D1.



## 13.7 Increment and Decrement Instructions

**INC – Increment**

**DINC – 32-Bit Increment**

**DEC – Decrement**

**DDEC – 32-Bit Decrement**

The increment and decrement instructions simply add or subtract 1 from the data in a 16-bit data register. These instructions execute nearly twice as fast as ADD or SUB instructions, and are not subjected to the limitations of a counter. These commands can be coded to operate on 32-bit numbers as **DINC/DINCP** and **DDEC/DDECP**.

Since this operation occurs every scan, **INCP/DECP** are almost always used for most applications.



When X11 is on, register D1 is increased by one every program scan.



When X11 is on, register D1 is decreased by one every program scan.

## 13.8 EXERCISE      **INC and DEC**

Please find Project #3 in the appendix. This project is intended to demonstrate the difference between INC/DEC and INCP/DECP.

## 13.9 Arithmetic Instructions

The arithmetic instructions must be placed in the end of the rung. When X1 conducts the addition instruction is performed.



When X1 conducts, the integer constant 30 (K30) is added to the existing value of D0 and the result placed in D10 every PLC scan.

Math Function	16-Bit Data	32-Bit Data	Floating Point
Addition	ADD	DADD	DEADD
Subtraction	SUB	DSUB	DESUB
Multiplication	MUL	DMUL	DEMUL
Division	DIV	DDIV	DEDIV
Square Root	SQR	DSQR	DESQR

Note how all 32-bit commands were prefixed with a **D**. All floating point commands are also prefixed with a **D** since they operate on 32-bit numbers, and the commands all begin with an **E** to indicate floating point format.

Integer square root requires a little more explanation. The square root calculated will be a rounded number, unless floating point math is being done. If a rounded answer is produced, special relay M8021 is turned on.

### **13.10 EXERCISE      Binary Math**

Please find Project #4 in the appendix. This project is intended to give the student practice in using the arithmetic instructions.

### **13.11 EXERCISE      Parking Lot Part 2**

This is a modification of the parking lot program from earlier. Add logic to count the total number of cars today, and store that number in D10. Add logic to count how much money should be in the cash drawer at the end of the day, assuming each car pays \$5.00 to park, and store that total in D12. There will also be a reset button for the manager to reset the car count and cash total at the end of the day. The reset button is addressed M10.

### **13.12 EXERCISE      Conveyor Control Part 3**

Take the Conveyor Control program and modify it to: Use the HMI to set the number of times the program cycles. The number must be a value between 5 and 15. If the number is not a value between 5 and 15, the program will not start. Display the number of cycles elapsed on the HMI display.

### **13.13 High Speed Processing**

The FX3U has built-in high-speed counters. These counters are either hardware counters that execute independent of the PLC scan, or software counters that execute as interrupts when they count. Thus they function independent of the program scan. This allows the high-speed counters to count at speeds of up to 60 kHz.

While the use of high-speed counters is somewhat beyond the scope of this class, all the important information on their use is in section 4.7.1 of the FX3U Programming Manual.

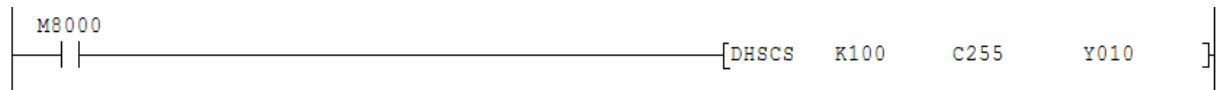
The counter number used varies based on the type of input and the physical inputs used to connect it. The chart in Section 4.7.1 of the FX3U Programming Manual shows the various high speed counters as well as their associated formats and inputs.

High speed counters are 32 bit counters, thus 32 bit “D” type instructions are required when using them.

While it is possible to use **SET**, **RESET** and the **comparison instructions** with high speed counters, these instructions are scan dependent and limit the benefits of high speed counters. To obtain full benefit, use the following high speed instructions.

### HSCS – High Speed Counter Set

This instruction functions like the standard SET instruction. When the counter reaches a specified value, a bit is set. This instruction uses an interrupt and is scan independent.



This instruction sets Y10 when the value of counter 255 equals 100.

### HSCR – High Speed Counter Reset

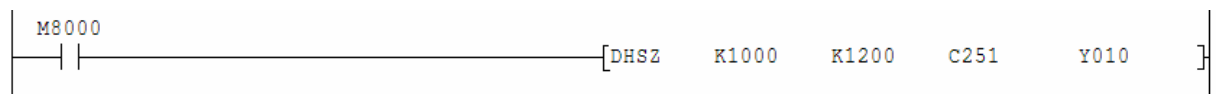
This instruction functions like the standard RESET instruction. When the counter reaches a specified value, a bit is reset. This instruction uses an interrupt and is scan independent.



This instruction resets Y10 when the value of counter 255 equals 200.

### HSZ – High Speed Zone Compare

This instruction functions like the standard ZONE COMPARE instruction. Depending on the count in relation to the range, one of 3 bits will turn on.



This instruction sets:

- Y10 when  $C251 < 1000$ .
- Y11 when  $1000 \leq C251 \leq 1200$
- Y12 when  $C251 > 1200$

Notice that DHSCS, DHSCR, and DHSZ are used as the operation instead of HSCS, HSCR, and HSZ. High speed counters are 32 bit devices, so it is necessary to add the 'D' to the instruction to make it use 32 bit number.

## 13.14 TO/FROM Instructions

The different types of SFMs were discussed earlier in the lesson. These modules increase the capabilities of the PLCs. They can provide analog signal functionality, high speed counters, or network connections just to name a few of the options available.

For the vast majority of the SFMs it is necessary to put logic in the ladder logic program to pass information between the CPU and the SFM. This is accomplished through the use of **TO** and **FROM** statements.

To understand TO/FROM instructions, it is important to understand the concept of the Buffer Memory Location (BFM). Inside each SFM, there are many memory locations that have a specific function. For example with the FX2N-4DA module BFM #0 holds the output mode and BFM #1 holds the digital value for channel 1.

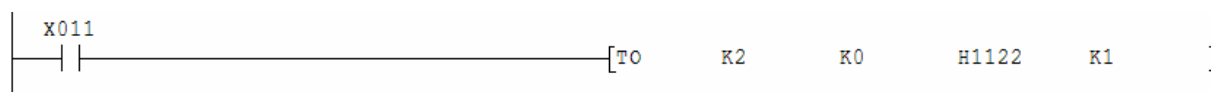
Note: Before attempting to program a SFM, it is absolutely necessary to have the manual for that module. With out the manual it is impossible to know what parameters need to be set, what BFMs to send the data to, or what BFMs to receive the data from.

The **TO** instruction is used to send data from the CPU to the SFM. Typically this data will be parameters that guide the functioning of the module or output data for an analog output module or network communication module.

The format of the instruction:

—[ **TO**        **SFM**        **BFM**        **SRC**        **COUNT**       ]

Example:



This rung writes 1 word of data, the constant H1122, to BFM #0 (K0), in the 3<sup>rd</sup> SFM encountered. If the module is an FX2N-4DA, this sets channels 1 and 2 to output 0 – 20mA, and channels 3 and 4 to output 4 – 20mA.

The **FROM** statement has the same format as the TO statement. The FROM statement is used to move data from a BFM in an SFM to the PLC's memory. An analog input module stores converted data from its inputs in various BFMs. The FROM statement moves the data from the BFMs in the module to destination devices in the CPU, where operations can be performed.

The format of the instruction:

—[ **FROM**        **SFM**        **BFM**        **DEST**        **COUNT**       ]

## 13.15 EXERCISE FX2N-5A Module Access

The FX3U demo cases have installed an FX2N-5A analog module. Write the TO/FROM instructions required to communicate to this module and populate the data fields as shown on the GOT screen. The list of buffer memories for the FX2N-5A module can be found in the FX2N-5A Hardware Manual.

NOTE: the averaging times value from the GOT should be written to all 4 averaging times values in the module.

## 13.16 Shift Registers

The FX3U contains numerous commands for shifting bits within a word or words within a table. There are commands to shift bits or words to the left or right. The size of the data in the shift, size of data to be shifted, as well as the source for the original and data to shift in can all be defined.

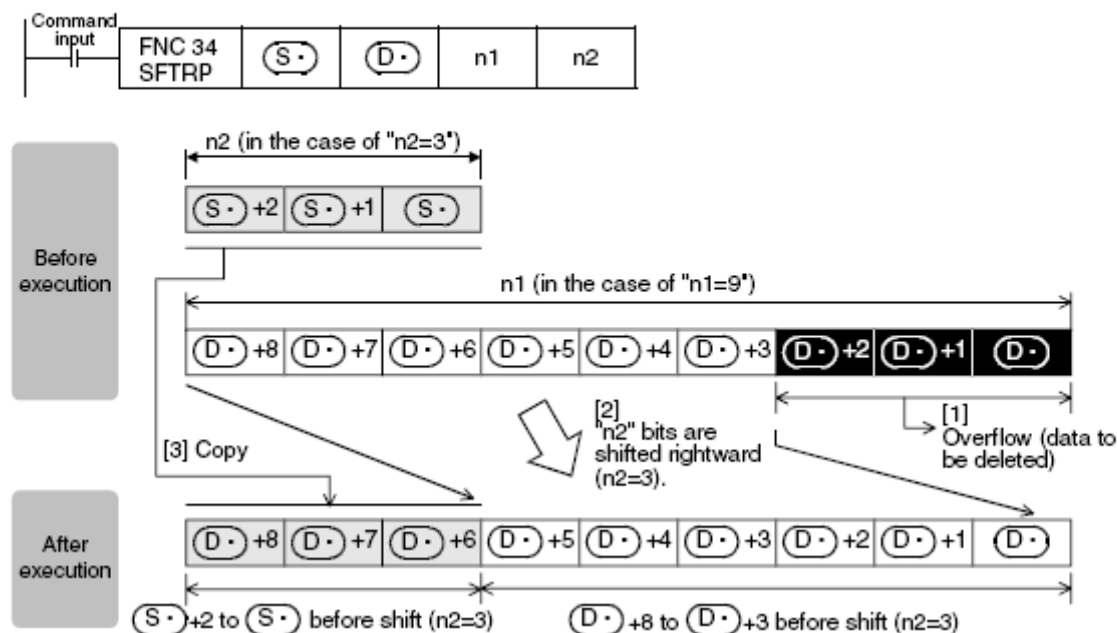
### SFTL – Bit Shift Left

### SFTR – Bit Shift Right

The Bit Shift Left and Bit Shift Right commands will move bits within a word (or multiple words). The command takes 4 parameters. The first parameter is the location of the data to be shifted into the register(s). The second parameter is the first address of the shift data. The third parameter is length of the shift data. The fourth parameter is the number of bits left or right to shift the data.

For "n1" bits (shift register length) starting from (D•), "n2" bits are shifted rightward ([1] and [2] shown below).

After shift, "n2" bits from (S•) are transferred to "n2" bits from (D•) + n1 - n2 ([3] shown below).

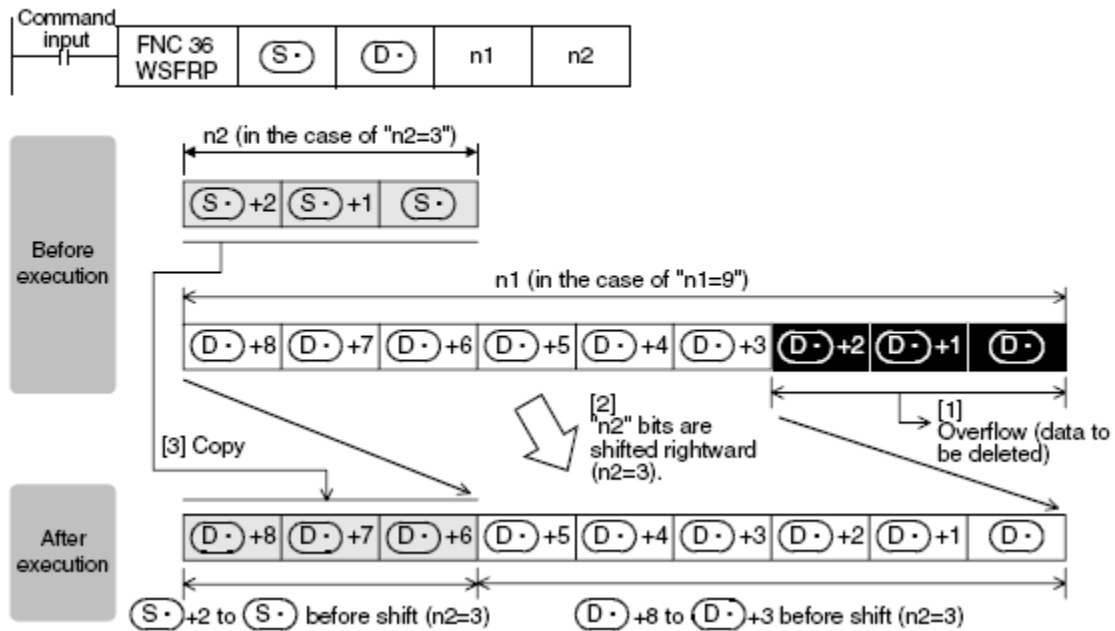


## WSFL – Word Shift Left

## WSFR – Word Shift Right

The Word Shift Left and Word Shift Right commands will move a word (or multiple words) of data in the same way as shown above for the bit shift instructions. The command takes the same 4 parameters. The first parameter is the location of the data to be shifted into the register(s). The second parameter is the first address of the shift data. The third parameter is length of the shift data. The fourth parameter is the number of words left or right to shift the data.

For "n1" word devices starting from (D•), "n2" words are shifted rightward ([1] and [2] shown below). After shift, "n2" words starting from (S•) are shifted to "n2" words starting from [(D•) + n1 - n2] ([3] shown below).



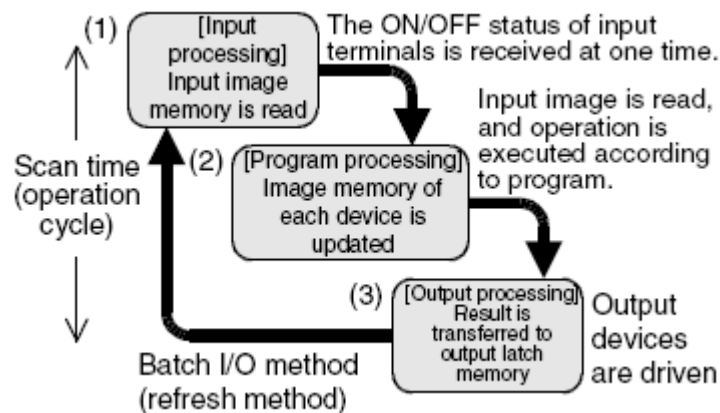
### 13.17 EXERCISE Bit Shift Register

Write a bit shift register to move bits across the 8 outputs on the PLC. Use the discrete I/O simulator screen for this exercise. Use X10 to trigger a shift of one bit into the shift register. Use X11 as the source bit to shift in.

The end result should be that when X11 is on and X10 is pressed, an on bit should be shifted into the register. If X11 is off when X10 is pressed, an off bit should be shifted into the register. Y0 thru Y7 are your output bits. Use a shift left or shift right command.

## 13.18 Program Flow Control

By default the PLC program processes top to bottom and left to right. At the beginning of the program, the actual inputs on the system are read into an image memory. The program is then processed, with modifications being made to the internal image memory. Once the END instruction is reached, the outputs are updated based on the image memory. Then the process begins again.



Sometimes this program flow needs to be controlled. There are several options for controlling the flow of the program. These methods include jumps and subroutine calls.

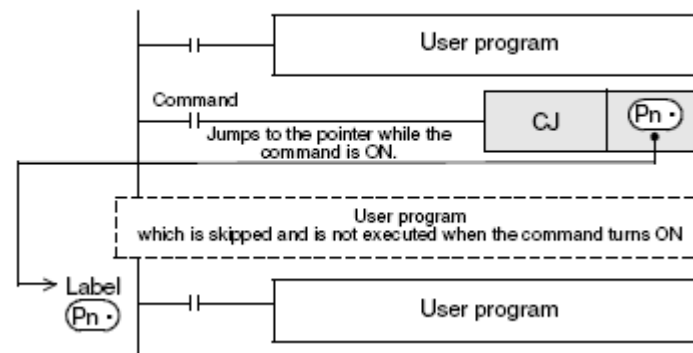
These commands use the pointer addresses mentioned earlier. To set a pointer, double click in the left margin of the ladder before the power rail and enter an address beginning with P. This is the pointer. When referencing this location in a jump or subroutine, call it by its Pxx address.

NOTE: Address P63 is reserved. It is a jump to the END instruction. Do NOT code a P63 in your program, or an error will be generated.

### CJ – Conditional Jump

The conditional jump command allows a section of logic to be skipped. The command is activated when the conditions leading up to the CJ instruction are true. When the jump is executed, all code between the CJ command and the pointer is not executed.

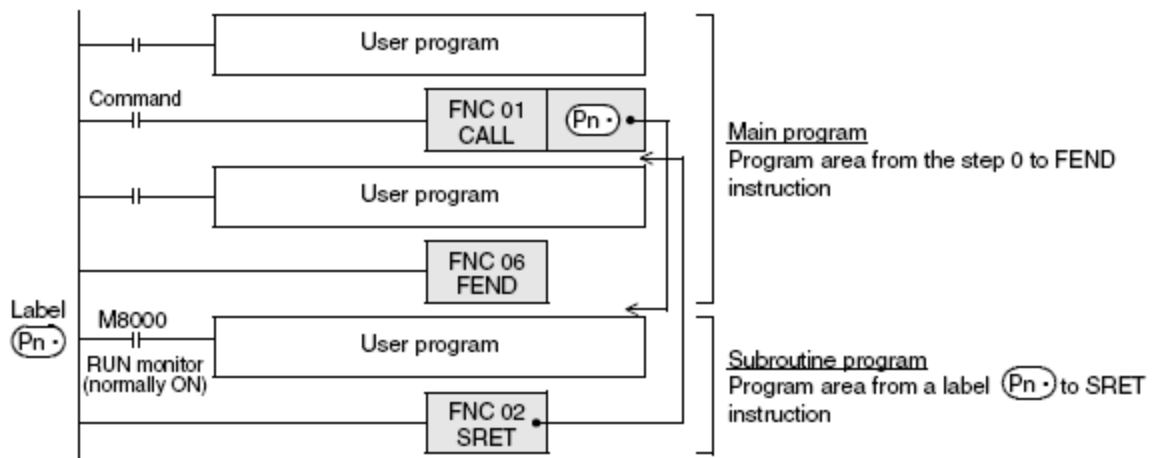
1) In the case of CJ instruction



**CALL – Call Subroutine**  
**SRET – Subroutine Return**  
**FEND – Main Routine Program End**

The call subroutine command allows the execution of a section of code only when activated. Unlike the jump command, execution of the main program resumes from the point where the subroutine was called. By removing code from the main ladder, scan times can be improved, since the subroutine code is only processed when the subroutine is active.

Subroutines are coded at the end of a PLC program. Main program execution is stopped at the **FEND** instruction, and the subroutines are created between this instruction and the END instruction. Subroutines are referenced by their pointer number for a starting point, and the next **SRET** command is considered the end of that subroutine. When the logic in front of it is active, the **CALL** instruction executes the code for that subroutine until the **SRET** subroutine return command is reached. Once the SRET is reached, program execution returns to the main ladder program and continues from the location of the CJ instruction.





# LESSON 14 – Diagnostic Devices

No programmer is perfect, and no PLC is going to last forever. Fortunately, the FX-series PLC line has a number of dedicated relays and registers that store information, including error codes about the operation of the PLC.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Identify those registers and relays that assist in troubleshooting.
- ✓ Interpret the information shown by the registers and relays.
- ✓ Write small programs that can provide diagnostic functions
- ✓ Use GX-Developer diagnostics to troubleshoot errors

**Materials:** FX-Series PLC Training Manual  
FX-series Demo Kit

## **14.1 Special M Relays**

The addresses M8000 and above are reserved for system use. In the FX1S/1N/2N/2NC, the range is M8000-M8255. In the FX3U, the range is M8000-M8511. These bits have various system functions. The complete list of special M contacts is found in the FX3U Programming Manual in Chapter 36. It can also be found in the GX-Developer software in the help menu.

These relays can be useful in your ladder program and troubleshooting. Some of the more commonly used contacts are ...

M8004	Indicates that an error has occurred
M8006	Low battery
M8064 ~ M8068	Latch on if various types of errors occur
M8064	– Parameter error
M8065	– Syntax error
M8066	– Program error
M8067, 68	– Operation error – typical example is dividing by 0.

These M806# bits have a companion D806# register which hold the error number. These will be covered in the next section.

The following relays are not diagnostic relays per se, but are very useful in writing programs:

M8000	Always on
M8001	Always off
M8002	On only for the first scan
M8003	Off only for the first scan
M8011	10ms clock pulse
M8012	100ms clock pulse
M8013	1 second clock pulse
M8014	1 minute clock pulse

## 14.2 Special D Registers

The addresses D8000 and above are reserved for system use. In the FX1S/1N/2N/2NC, the range is D8000-D8255. In the FX3U, the range is D8000-D8511. These registers have various system functions. The complete list of special D registers is found in the FX3U Programming Manual in Chapter 36. It can also be found in the GX-Developer software in the help menu.

A few of the diagnostic register available

D8004	Error relay active (i.e. 8064 indicate M8064 active)
D8006	Low battery fault alarm level
D8010	Present scan time
D8064 ~ D8068	Error codes
D8064	– Parameter error code
D8065	– Syntax error code
D8066	– Program error code
D8067, 68	– Operation error code
D8069	Error step number

The following registers aren't diagnostic registers per se, but provide useful information or functions.

D8001	PLC type and firmware revision
D8005	Battery voltage (actual)
D8013 ~ D8019	Real-time clock
D8013	Seconds
D8014	Minutes
D8015	Hours (24 hour format)
D8016	Day
D8017	Month
D8018	Year (2 digits 00-99)
D8019	Day of Week (0 Sunday – 6 Saturday)
D8020	Input Filter (range is 0-60)
	0 = 5 $\mu$ s to 200 $\mu$ s (varies by address)
	# = #ms

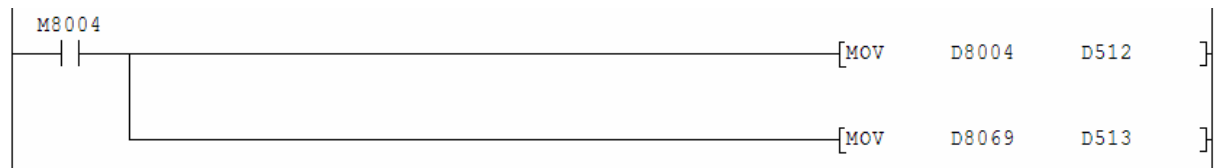
## 14.3 Handy Troubleshooting Circuits

This circuit indicates that the battery voltage is low.



The alarm bit turns on an output that can activate a light, buzzer, etc.

This circuit stores the error code and error step. If the D registers have been declared to be latched registers, or the registers are in the range of D512 to D7999, then the error information will be retained, even in the event of a power loss



## 14.4 Real Time Clock Usage

The FX PLCs all have a real time clock capability. With the FX2NC, this requires a real time clock enabled memory cassette. The data for the real time clock is stored in D8013-D8019 as previously discussed. There are a variety of functions which can be used to access the data in the real time clock. Some of them are shown below. More detail on these commands can be found in Chapter 21 of the FX3U Programming Manual.

If the registers D8013-D8019 are to be modified by commands other than the ones below, the real time clock must be stopped. To stop the clock, turn on bit M8015. When M8015 turns back off, the data from D8013 to D8019 is written to the PLC's internal real time clock. If the clock is not stopped the changes will not take effect.

If display of a 4 digit year is desired, code can be written to write the number '2000' to the year register D8018. This needs to be done on first scan every time, so it should be triggered by M8002. It will change the display to 4 digits, but it will not modify the year value already in the RTC.

### **TRD – RTC Read**

The **TRD** instruction is used to read the PLC's real time clock data from the D8013-D8019 range into another area in the PLC's memory. It copies all 7 registers to 7 consecutive data registers

### **TWR – RTC Write**

The **TWR** instruction is used to write the PLC's real time clock data in D8013-D8019 range from another area in the PLC's memory. It copies 7 consecutive data registers to the 7 registers which make up the PLC's real time clock.

### **TCMP – Time Compare**

The **TCMP** time compare instruction will compare the hours, minutes, and seconds as set in the parameters to 3 consecutive registers which hold the real time clock data.

## **TZCP – Time Zone Compare**

The **TZCP** time compare instruction will compare the hours, minutes, and seconds as set in the parameters to 2 sets of 3 consecutive registers which hold the upper and lower limits of the real time clock data to compare.

### **HTOS – Hours to Seconds (16-Bit) DHTOS – Hours to Seconds (32-Bit)**

These two commands will convert a time value in hours, minutes, and seconds into a number of seconds and back. The **HTOS** instruction uses 3 consecutive registers holding hours, minutes, and seconds and creates a 16-bit output of a number of seconds. To use 32-bit result, use the **DHTOS** command.

### **STOH – Seconds to Hours (16-Bit) DSTOH – Seconds to Hours (32-Bit)**

These two commands will convert a number of seconds into a time value in hours, minutes, and seconds. The **STOH** instruction uses a 16-bit input of a number of seconds to create 3 consecutive registers holding hours, minutes, and seconds. To use 32-bit seconds source, use the **DSTOH** command.

### **TADD – Time Add TSUB – Time Subtract**

The TADD and TSUB commands are used to add or subtract time data. The data is provided as 3 consecutive source registers for both input devices and a set of 3 registers are used for the resulting data. These registers are in hours, minutes, seconds order as with the other time commands.

## **14.5 EXERCISE Daylight Savings Time**

The PLC does not have built in option to control Daylight Savings Time. PLC code must be written to control the PLC clock for DST. The U.S. standard for Daylight Savings Time is shown below.

First Sunday in April at 2AM, set clock ahead one hour  
Last Sunday in October at 2AM, set clock back one hour.

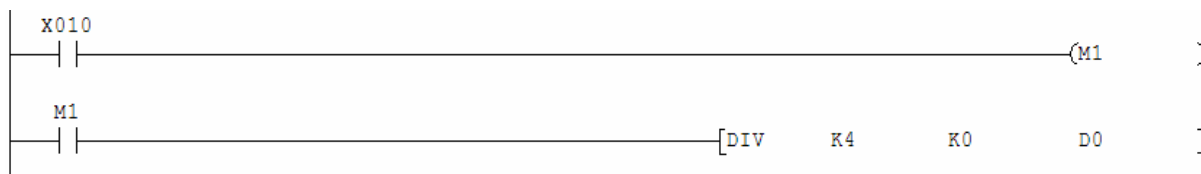
Write the PLC code to set the clock ahead in the spring and back in the fall.

Be careful in the fall, we only want to set the clock back once!

## 14.6 GX-Developer Diagnostics

Much of what has been discussed was necessary to troubleshoot programs in pre GX-Developer days. This information is still useful to display faults to an HMI, or to save fault codes for future examination, as in the ladder logic above.

Please create a new program and enter the rung shown below:

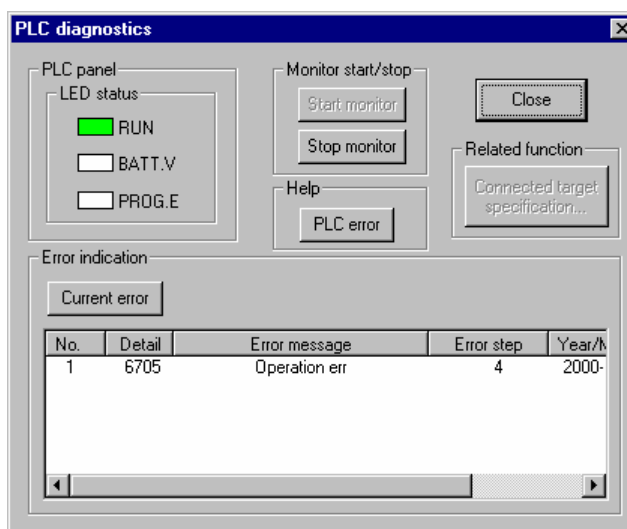


Take a moment to examine the above logic. What happens when X0 is turned on? The fixed value 4 is divided by 0 and the result is placed into D0.

This is an illegal operation, since the result is an infinite number. Trigger X0.

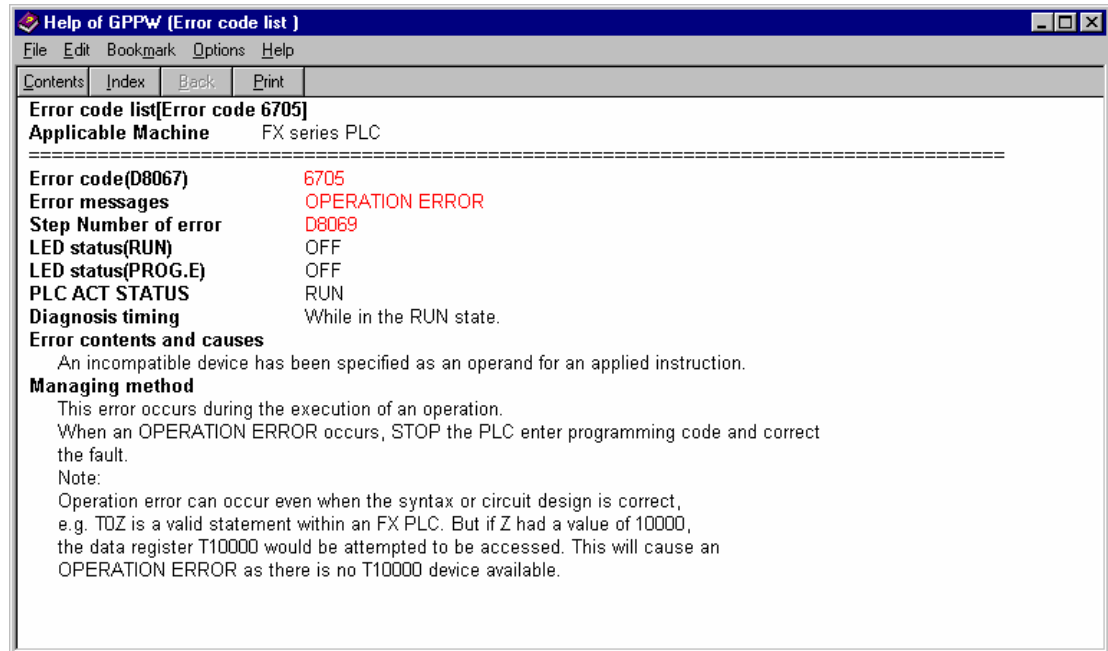
To troubleshoot do the following steps:

1. Click on the Diagnostics pull-down menu.
2. Click on PLC Diagnostics. A menu similar to the one below appears:



Error Number 6705 is the error that appears in D8067. Step 4 is the step where the error occurs and is stored in D8069. The message Operation Error refers to Error Code 8067, which is stored in D8004.

3. Highlight the error by clicking on it
4. Click on the PLC Error in the Help section.
5. The following screen should appear:

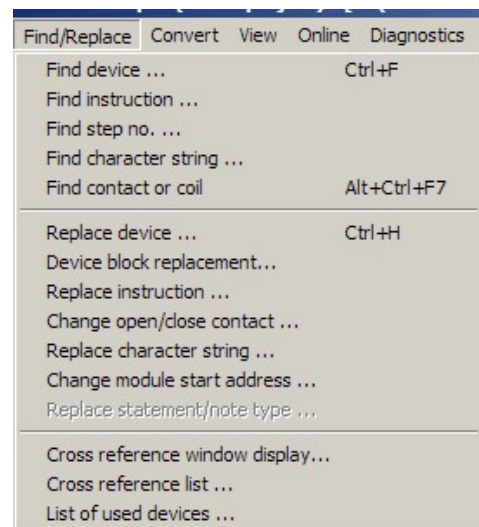


This screen explains possible causes of the error state and possible solutions. While divide by 0 isn't explicitly stated here, it can be inferred from the references to an incompatible device specification (the K0 as a divisor).

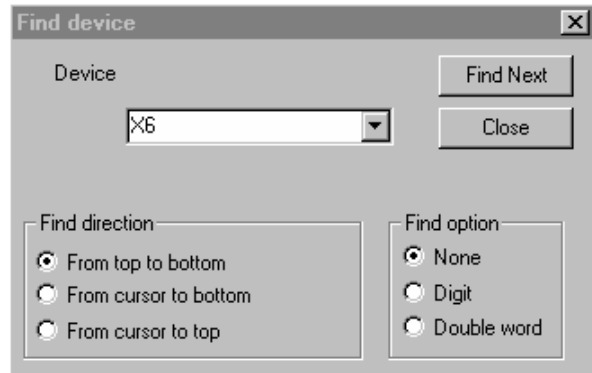
## 14.7 Find/Replace Menu

GX Developer provides several different methods for locating devices or instructions. The Find/Replace menu has a variety of tools for searching or modifying programs. The find options will search a program for an occurrence of an address, function, or string. The replace options will allow the replacing of addresses, functions, open/closed contacts, and other options. The cross reference and used devices lists will show all occurrences of an address in a program, or all used addresses in the program.

Shown to the right is the list of options on the Find/Replace menu.



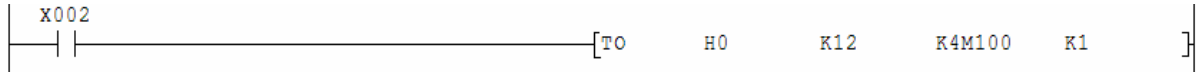
**Find Device** is used to find a device address regardless of the instruction. It will search the entire program for a particular address. There are a couple of options on this screen. In the bottom left you can limit the search to the entire program, current location back to top, or current location to end of program.



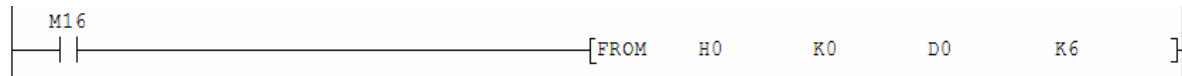
There is also a setting for a find option.

**None** causes the find function to look specifically for the address entered.

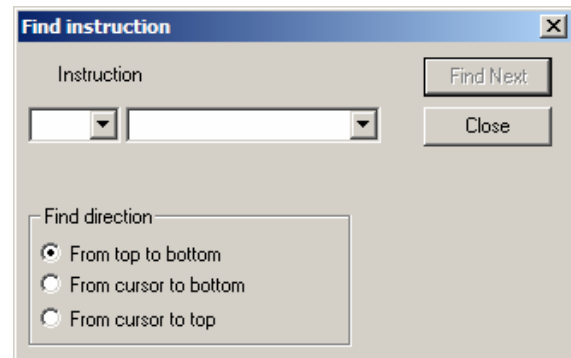
**Digit** allows the find to look for a bit address in a word of bits. A digit search of the following code will find that M110 has been used as part of this TO instruction.



**Double Word** expands the search to include a word that is used by an instruction using multiple words. A double word search of the following code will find that D5 has been used as part of this FROM instruction.



**Find Instruction** is used to locate all instances of a particular instruction. The first dropdown box selects the ladder symbol and the second dropdown box selects the command to search for. The find direction section works as previously discussed.



**Find Step Number** will find a step number in the program. This can be useful with the error information previously discussed to search out the location of a program error in the program.



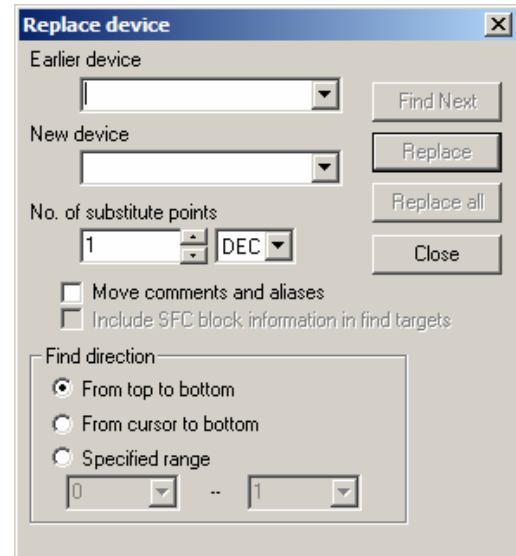
**Find Character String** locates a string of text within the program. If the currently open window is the ladder logic, it will search for text in the ladder. If the comment list is open, the comments will be searched, and selection buttons allow the search to be limited to displayed address list or all addresses, as well as limit the search to comments or labels.

**Find Contact or Coil** will search the program for an address used as either a contact or a coil. In the first window, select contact or coil. Then in the right window, enter an address. This can be handy when an address has been used as a contact numerous times in a program to find the one location it was used as a coil.



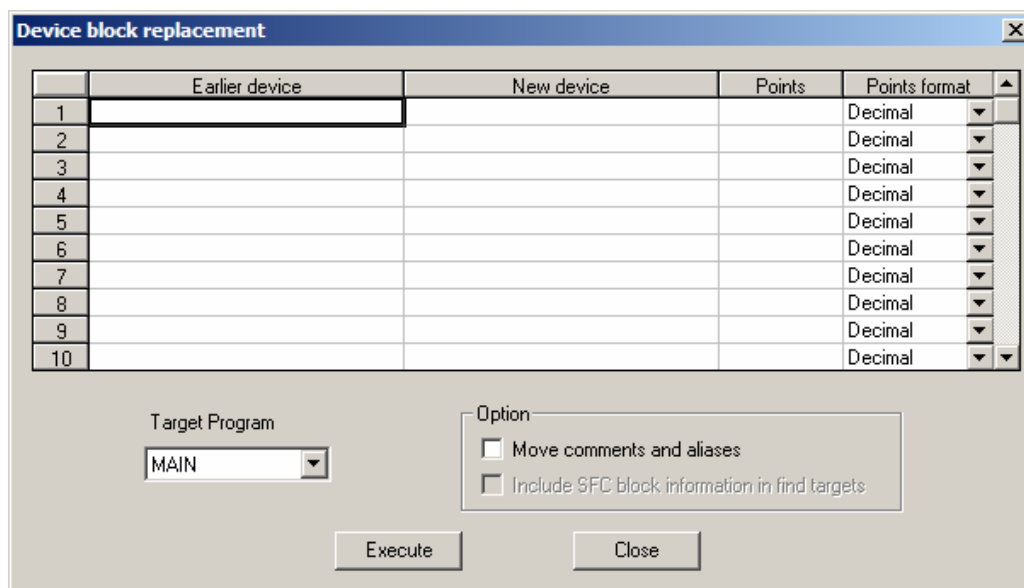
The dialog box titled "Find contact or coil" has a dropdown menu labeled "Contact" on the left and a text input field on the right. To the right of the text field are two buttons: "Find" and "Close".

**Replace Device** allows search and replace of a device or a range of devices within the program. A single address can be specified for the earlier device and for the new device. The number of substitute points sets how many consecutive addresses to move. The check box for move comments and aliases allows comments and aliases assigned to an address to be moved with the address. The find direction options allow the search and replace to be restricted to a certain portion of the program.



The dialog box titled "Replace device" contains several fields and options. It has a dropdown for "Earlier device", a text field for "New device", and a numeric field for "No. of substitute points" set to "1" with a "DEC" button. There are buttons for "Find Next", "Replace", "Replace all", and "Close". Below these are two checkboxes: "Move comments and aliases" and "Include SFC block information in find targets". At the bottom, there is a "Find direction" section with three radio buttons: "From top to bottom" (selected), "From cursor to bottom", and "Specified range". Below the radio buttons are two numeric fields with a range from "0" to "1".

**Device Block Replacement** takes the Replace Device option a step farther. It allows the programmer to replace multiple ranges of devices in a single command. Groups of timers, counters, inputs, and data registers can all be acted on at once. Each line in the chart is a different range of addresses.



The dialog box titled "Device block replacement" features a table with 10 rows. The columns are "Earlier device", "New device", "Points", and "Points format". The "Points format" column has a dropdown menu for each row, all currently set to "Decimal". Below the table is a "Target Program" dropdown menu set to "MAIN". To the right of this are two checkboxes: "Move comments and aliases" and "Include SFC block information in find targets". At the bottom are "Execute" and "Close" buttons.

	Earlier device	New device	Points	Points format
1				Decimal
2				Decimal
3				Decimal
4				Decimal
5				Decimal
6				Decimal
7				Decimal
8				Decimal
9				Decimal
10				Decimal



**Replace Instruction** will change one instruction type to another. The earlier instruction is the type of instruction to replace, and the new instruction is the instruction to substitute into the program. This can be useful for example to replace all occurrences of INC with INCP in a program.

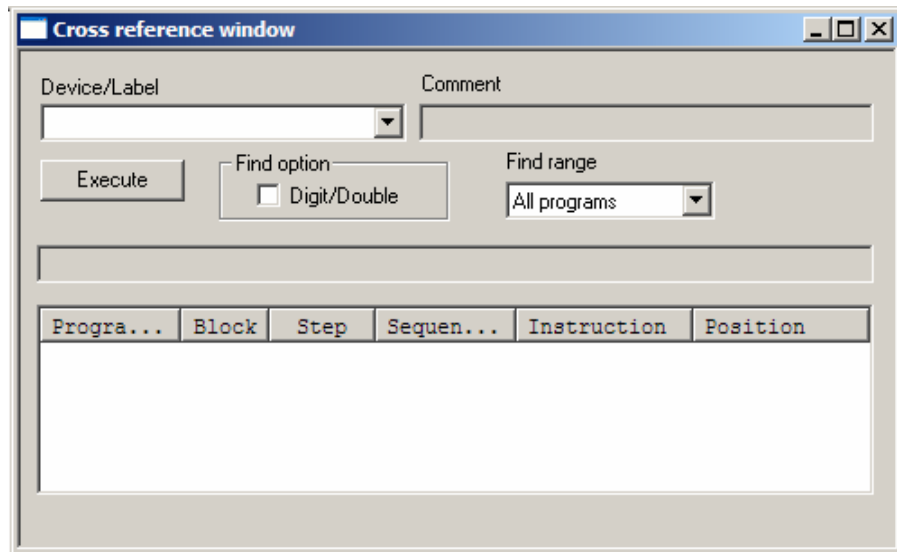
**Change Open/Close Contact** will change all occurrences of an open contact to a closed contact and change closed contacts to open. A single command will invert the state of all occurrences of the address. This is useful if the type of input provided to a machine changes or does not match the code, such as a program which is expecting a normally closed stop signal and the machine is wired with a normally open switch.

**Replace Character String** will replace a string of text. This command searches in a similar fashion to the Find Character String mentioned above. It will search for text in the ladder, comments list, or device list, and replace it with a new character string.

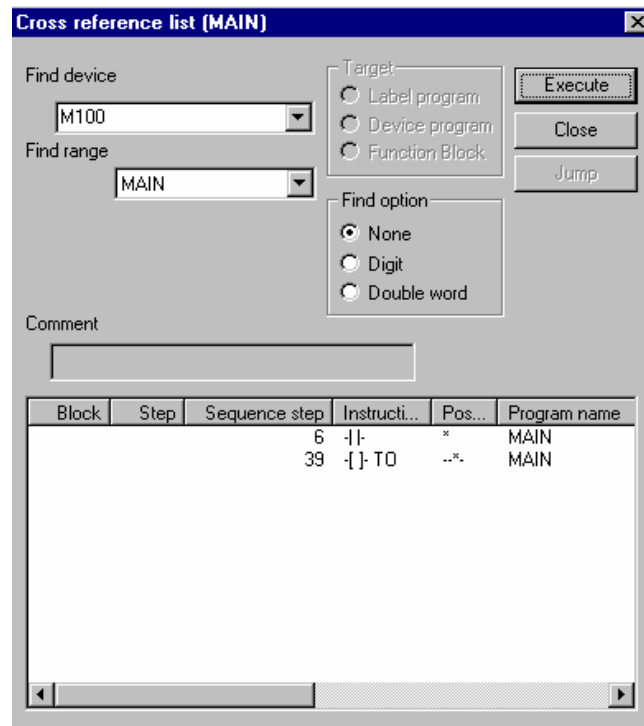
**Change Module Start Address** allows the programmer to change the head address of a special function module. This code will search a program and change all occurrences of the SFM number, such as TO or FROM instructions. By specifying a start and end as different values, a range of SFMs can be moved at once. Enter the start and end SFM numbers, and enter a single new module number. The software will move the old addresses to a range of equal size starting at the new module address.

Cross reference information can be displayed in one of two ways. One option is the Cross Reference List, which is a pop-up window display of the cross reference. This will sit on top of all other windows, and must be closed to move to another screen. The other is the Cross Reference Window. This window-based display of the cross reference can be tiled, cascaded, minimized, or moved around as other windows in the workspace. It does not need to be closed to move to another screen.

**Cross Reference Window** is a window-based display of the cross reference list.



**Cross Reference List** is a pop-up window version of the list.



## List of Used Devices

The List of Used Devices will show a list of the devices used in the program. The list can be configured to show a single program (FX only supports one) or all programs in the PLC (in the case of a Q-Series PLC). An address is entered in the find device box, and after pressing the Execute button, the list starting at that address is shown in the window below. There are columns which will show an asterisk (\*) if that address has been used as an input or output. Inputs include source data for applied instructions and outputs include destinations of applied instructions. The ERR column is marked if the address is used as either input or output but not both. Depending on the addresses and their use in the program, this is not necessarily an error.

The screenshot shows the 'List of used devices (MAIN)' window. It has a title bar with a close button. Inside, there are two radio buttons: 'Target the whole program' (unselected) and 'Specify the target program' (selected). Next to the second radio button is a dropdown menu showing 'MAIN'. To the right are 'Execute', 'Close', and 'SFC find setting' buttons. Below the radio buttons is a 'Target' section with three radio buttons: 'Label program' (selected), 'Device program' (unselected), and 'Function Block' (unselected). To the right of this is a 'Find device' dropdown menu showing 'D0'. Below that is a 'Display range' section with 'D ( 0 - 511 )' and two arrow buttons. At the bottom is a table with the following data:

Device	-I	-O	Count	Error	Comment
D0	*	*	2		
D1	*	*	1		
D2		*	2	ERR	
D3		*	1	ERR	
D4		*	1	ERR	
D5		*	1	ERR	
D6					
D7					
D8					
D9					
D10					
D11					
D12					
D13					

## 14.8 Data Trace

GX Developer provides a troubleshooting tool called a trace. This tool allows the programmer to graph the state of values in the PLC over time. This tool executes a data trace within the PLC, so it is not limited by the communication method in use to the PC.

The FX series supports a data trace of 512 samples. Samples can be based on PLC scans or on an interval adjustable from 10ms to 2000ms in 10ms increments. It can be triggered from the software or by the changing of a bit or word value in the PLC. Samples can be recorded from before the trigger takes place as well.

The data trace in the FX series can log up to 10 bit devices and 3 word devices.

GX-Developer has a built-in wizard to assist in the configuration of the data trace. Settings can also be defined manually. Once the settings are made, the trace settings are stored to a file on the PLC. That file can be read back from the PLC. The results of the trace can be uploaded once the data trace has completed. The files can also be deleted from the PLC.

Once the data has been uploaded from the PLC, it can be viewed in GX-Developer or it can be output to a .CSV file which can be view or graphed from within Microsoft Excel.

# LESSON 15 – Documentation & Printing

The programs that have been written so far have been fairly simple, and you, the programmer have been present as they were written. This makes it fairly simple to troubleshoot these programs if there is a problem. Imagine if the program is 4000 steps, written by someone who left the company 2 years ago. Without any program documentation, it would be nearly impossible to troubleshoot a problem. This is why documentation of the program is a very important step in program creation.

**Lesson Objectives:** At the conclusion of this lesson, you will be able to...

- ✓ Describe the 4 types of documentation.
- ✓ Add documentation to a program.
- ✓ Describe the various options for printing a program

**Materials:** FX-Series PLC Training Manual

GX-Developer offers 4 types of documentation: Comments, Statements, Notes, and Device Labels.

Note: It has been shown earlier that it is possible to add and modify devices by double-clicking on the rung or device. This function will not work when a form of documentation has been enabled.

## **15.1 Comments**

Comments are attached to a device to provide a name or description. Typical comments for an input are: Start Push Button, Load Recipe, etc. Typical comments for a coil or internal bit are: Fault Light, Engine 1 On, and Process Enabled. Comments can be 3 lines by 5 characters per line, 2 lines by 8 characters per line, or 4 lines by 8 characters per line. This is set in Comment format selection under the View pull-down menu.

By default, the only options listed are the 3 x 5 and the 4 x 8. To enable the 2 x 8 it is necessary to go to the Tools pull down menu, select Options, click on the Whole Data tab, and select 16 in the Common Device Comment dialog box. When the Comment format selection is reopened, the options are now 3 x 5 and 2 x 8.

Comments are added by double clicking on the device and filling out the dialog box that opens.

Comments are the only form of documentation that can be downloaded to the PLC. If another programmer uploads a program that contains comments from a PLC, those comments will be available to that programmer as well. Notes and statements reside in the program on the laptop. If another programmer desires access to these, it is necessary to have a copy of the program on that hard drive.

**Important:** The PLC will only hold the first 16 characters of comments. The rest of the comment will be truncated.

If the programmer wants to document the program as the program is written, it is possible to have comments and one other form of documentation enabled at the same time. Go to the Tools pull-down menu, select Options, and click the box next to “Continues during Command Write” under the Device Comment Input heading. As devices are entered into the program, GX-Developer will prompt for a comment.

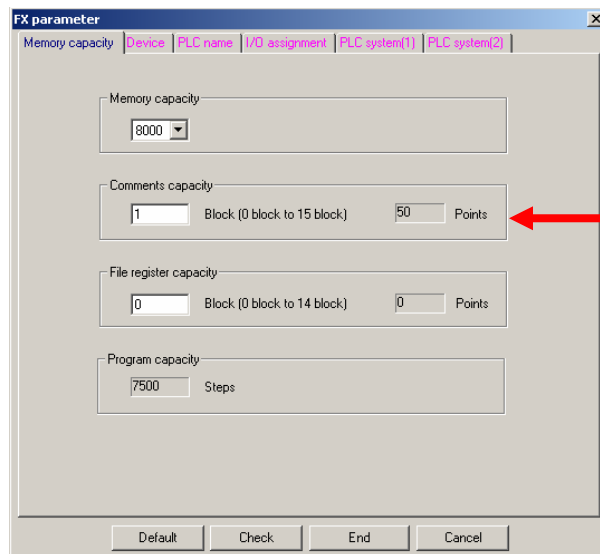
### **Downloading comments**

Downloading comments is very easy with the FX3U. Unlike with some of our other processor lines, there is no need to set comment ranges. It is only necessary to set comment capacity in parameters and download!

### **Setting comment capacity**

Comment capacity is set in PLC Parameters, under the Memory Capacity tab. The programmer allocates a number of blocks. Each block allows the programmer to comment 50 unique devices. Thus a comment for X0 is one device, even if X0 appears 15 times in the program. Unlike with some of our other processor lines, blank comments do not count towards the devices comment count.

Comment blocks consume 500 steps of programming space.

The screenshot shows the 'FX parameter' dialog box with the 'Memory capacity' tab selected. It contains four sections: 'Memory capacity' with a dropdown set to '8000'; 'Comments capacity' with a 'Block' field set to '1' (range 0 to 15) and a 'Points' field set to '50'; 'File register capacity' with a 'Block' field set to '0' (range 0 to 14) and a 'Points' field set to '0'; and 'Program capacity' with a 'Steps' field set to '7500'. At the bottom are 'Default', 'Check', 'End', and 'Cancel' buttons. A red arrow points from the 'Points' field in the 'Comments capacity' section to the right.

Set number of comment blocks – indicates how many devices can be commented

How many steps remain for programming?

## **15.2 Statements**

Statements, also known as circuit comments, provide a description of the purpose for a whole rung. A typical statement would be: “This rung waits until the counter reaches 20 and opens the gate”.

A statement can be up to 64 characters long. Multiple statements can be attached to a single rung to provide an in-depth description of the rung's purpose. Statements appear to the left and on top of a rung. To enter a statement, simply press the semicolon at anytime while the rung is in edit mode. A textbox will appear, type the message and then press OK.

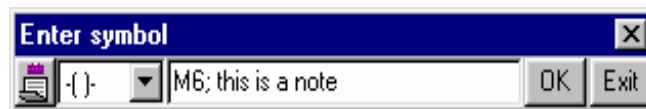
### 15.3 Notes

Notes are also referred to as coil comments. Notes appear to the right of the rung, over the output coil.

Notes can be used for any additional descriptive purpose, but they are intended to provide information about the output of the rung. Notes can be up to 32 characters long.

To enter a note, simply type a semicolon after the address of the output and then type the note.

Example:



### 15.4 Aliases

Aliases can be displayed in place of the device address (i.e. VACPUMP instead of X0). The alias can be no more than 8 characters long, with no spaces allowed. Aliases are entered by double clicking on the Comments icon in the Project List Window, and then double clicking on the icon with the name of the ladder (usually MAIN). A table will pop up, with the last column the Alias column. Find the desired address and enter the alias.

Return to the ladder, go to the pull down menu for View, and check Device Label. All the address with labels will have the addresses replaced by label. Note that it is possible to use the label as an address when entering new contacts; just enter an apostrophe before entering the alias

### 15.5 Viewing Documentation

It is necessary to enable GX-Developer to show the documentation that has been entered. This is because documentation, while very useful, tends to clutter the screen.

To view documentation, go to the View pull-down menu and check the desired documentation. It is possible to display comments, statements, and notes at the same time.

## 15.6 Printing

GX-Developer has very flexible printing capabilities. These can be accessed by clicking on the Printer Icon on the toolbar. The programmer can choose to print as much or little of the program as required. Some of the available printing options are:

1. Title – Prints a title page. Title can contain 64 characters/ line at 9 lines. Automatically includes the date of the print out
2. Ladder – Prints the Relay Ladder Logic diagram. Can choose to print out only part of the program.
3. TC setting – Prints out timer and counter information
4. Device Comments – Print out any commented devices and the associated comments
5. List of Used Devices – Prints out any device that is used in the program
6. Device Memory – Prints out the contents of the data registers
7. Parameters – Prints out the parameters of the CPU
8. Contact coil used list – Prints out all contacts and coils used in the program

While each tab can be configured and printed separately, there is an option to print multiple tabs of data in one print. This is the Multiple Printing button at the bottom of the window. This window will allow the selection of various tabs of information to print, as well as the order in which to print each section. Page numbers will be continuous through the entire printout.

The Page Setup button allows the configuration of the page layout. This includes margins, header and footer, page numbering, and paper size and orientation.



Here is the Print Dialog Box. Click on the tab for the method of printing desired.

**Print**

List of used device    Device memory    Device init    PLC parameters  
 Network parameters    Cross reference list    Project contents list    TEL  
 Title    MELSAP2.3    MELSAP-L    Ladder    Instruction list    TC setting    Device comment

Additional information

☒ Device comment    ( ☒ 4 \* 8    ☐ 3 \* 5 )  
☐ Statement/note  
☐ Machine name  
☒ Contact user  
 Print position    ☒ Right    ☐ Bottom    Setup range  
 Print range    ☒ All    ☐ Specified    Setup range  
☐ Coil user  
 Print range    ☒ All    ☐ Specified    Setup range

Program selection

MAIN    Select    Clear selection

Print conditions

☐ Print NOPLF  
☒ Renews page at each ladder block unit  
☐ Prints the blank lines with no device comments  
☐ Print in the Macro Instruction format

Print range

☒ All  
☐ Specified  
 Step -- Step

(In case of blocks under conversion, before printing make sure that conversion is completed )

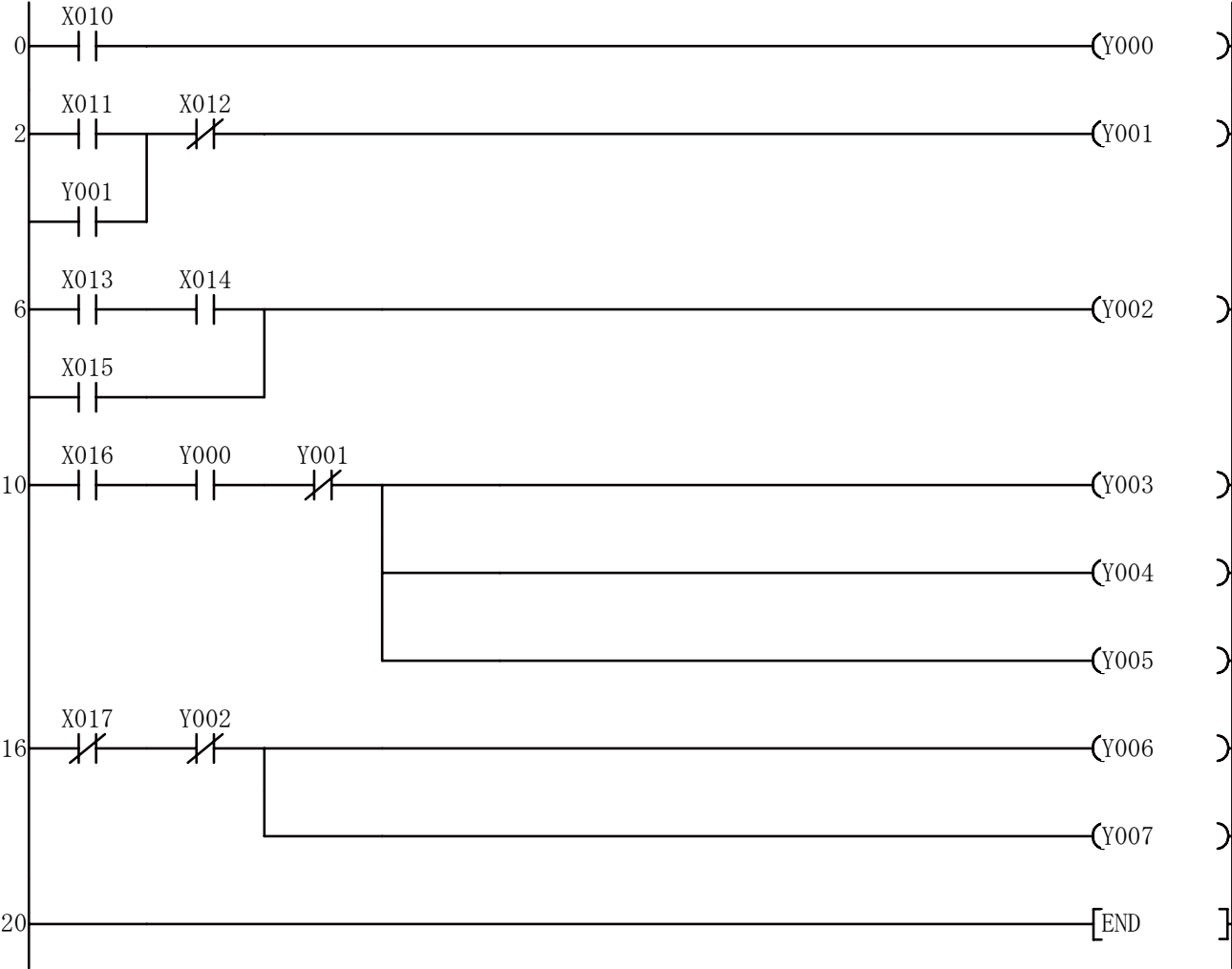
Printer setup    Page setup    Multiple printing    Print    Print preview    Close

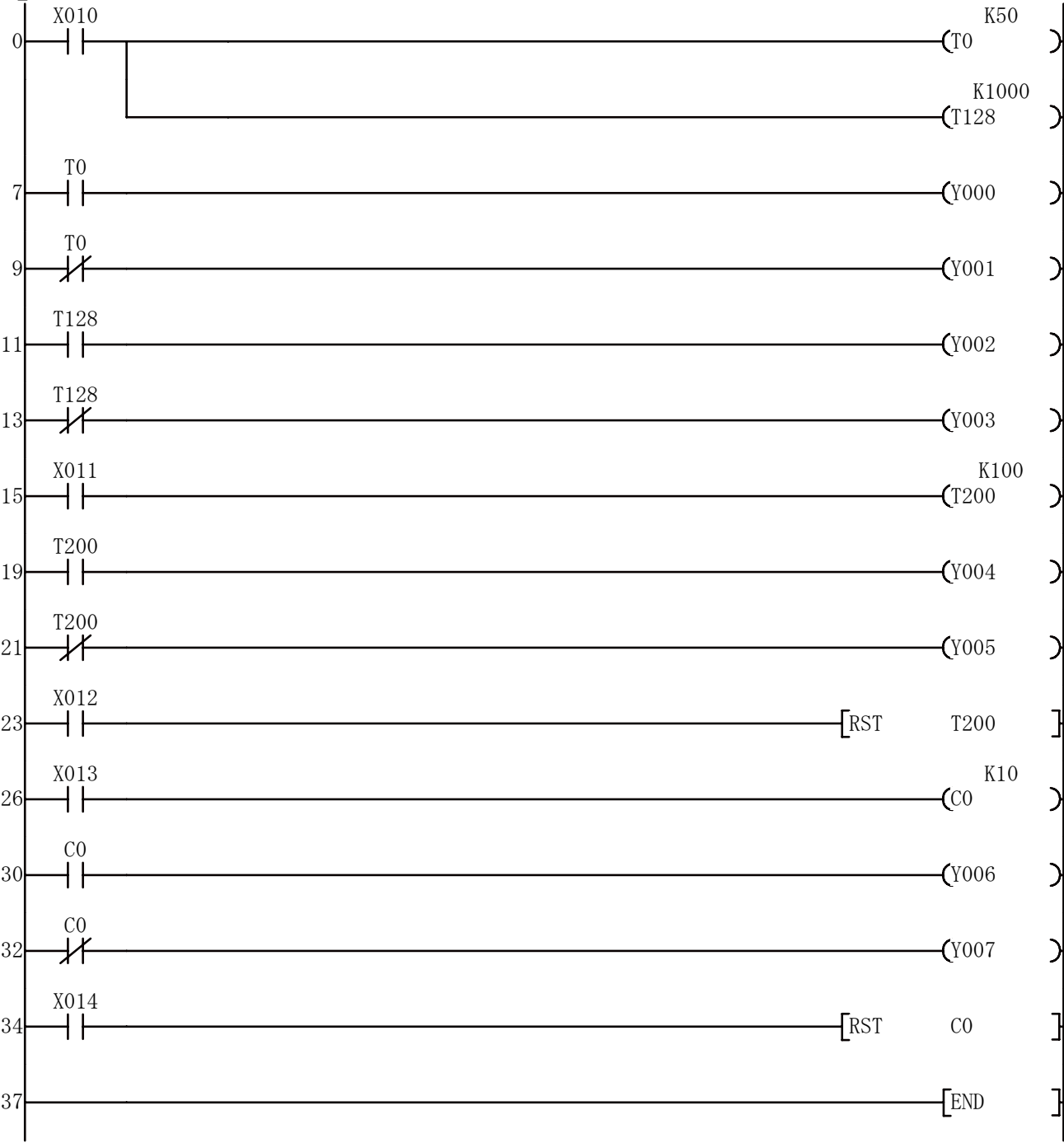


# APPENDICES



Project 1





Project 3



Project 4

