BASIC PLC PROGRAMMING

The Fundamental Knowledge of PLC

program-PLC.blogspot.com

Table of Contents

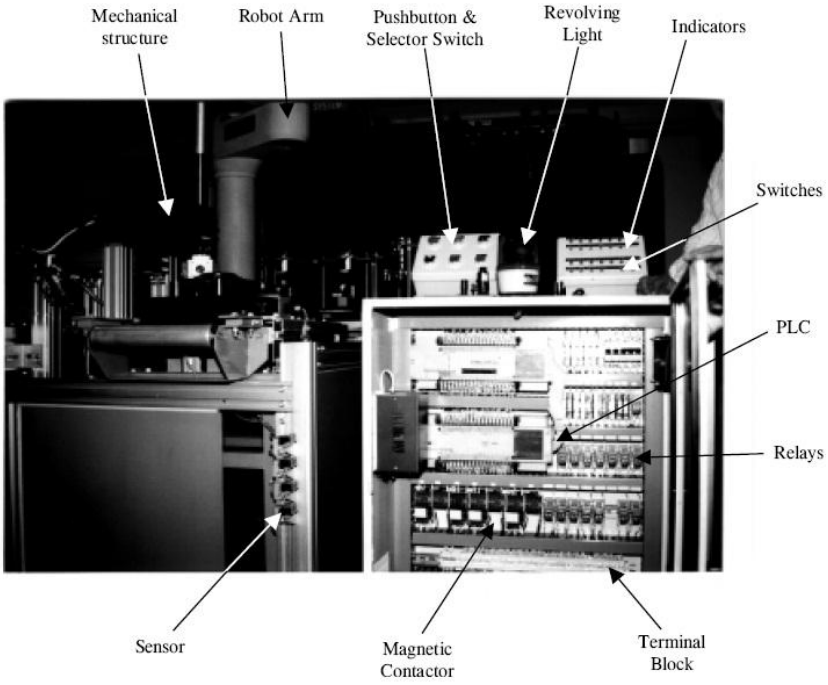
Chapter 1

PLC Introduction

1.1 Introduction

A group of engineers General Motors in 1968 was developed the first Programmable Logic Controller (PLC), when the companies were in search of an alternative to substitute complex relay control systems. The new control system had to meet the following requirements:

- Simple programming
- Program changes without system intervention and there is no internal rewiring
- Simple, low cost maintenance
- Smaller, cheaper and more reliable than corresponding relay control systems



Consequent development resulted in a system, which allowed the binary signals simple connection. The conditions as to how these signals were to be linked were identified in the control program. It became feasible for the first time to plan signals on a display and to file these in electronic memories in this new systems.

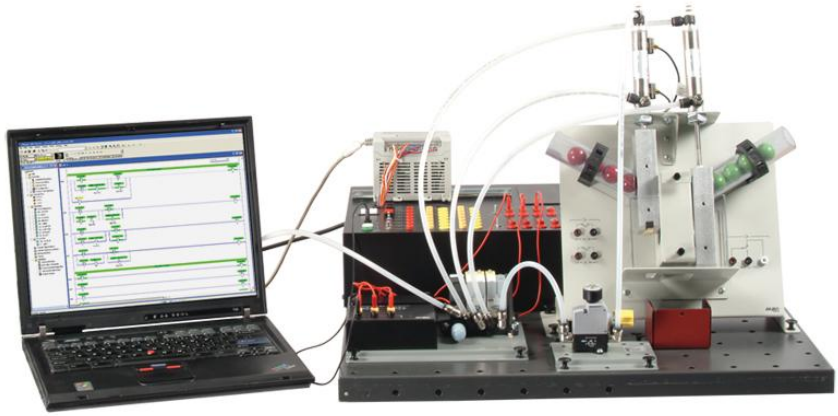
Three decades have left behind, during which the massive progress developed in the microelectronics development did not stop short of PLC. For example, even if program

optimization and hence a reduction of required capacity of memory firstly still characterized a vital key task for the programmer, currently this is hardly of any importance.

Furthermore, the functions range has grown significantly. Some years ago, analogue processing, process visualization or even the PLC utilization as a controller, were considered as Utopian. Presently, these functions support forms a fundamental part of many PLCs.

1.2 Areas of application of a PLC

Every machine or system has a controller. Dependent on the technology type used, controllers can be separated into hydraulic, pneumatic, electronic and electrical controllers. Often, a mixture of different technologies is applied. Moreover, differentiation is created between hard-wired programmable (e.g. wiring of electro-mechanical or electronic components) and PLCs. The initial is utilized principally in cases, where any reprogramming by the user is out of the query and the task size guarantees the development of a special controller.



Characteristic applications for such controllers can be found in cars, video cameras, and automatic washing machines. Nevertheless, if the task size does not guarantee the development of a special controller or if the user is to include the facility of setting timers and counters, or of making easy or independent program changes, then a universal controller use, where the program is written to a memory of electronic, is the ideal option? The PLC

stands for such a universal controller. It can be applied for different applications and, through the program installed in its memory, offers the user with an easy means of changing, expanding and optimizing control processes.

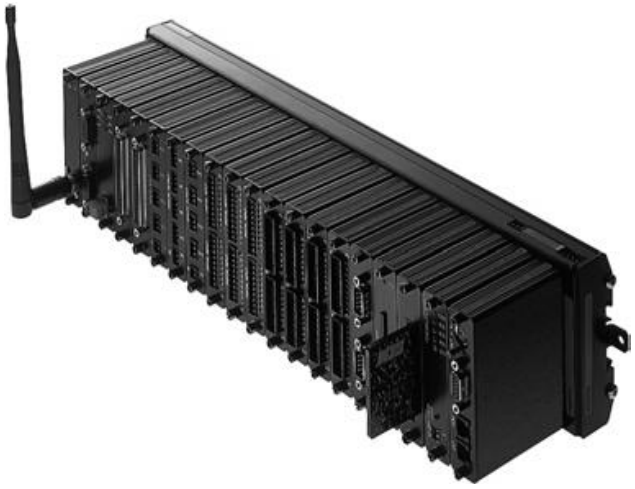
The creative task of a PLC engaged the input signals interconnection along with a specified program and, if "true", to switch the corresponding output. Boolean algebra forms the basis of mathematical for this operation, which recognizes accurately two defined statuses of one variable: "0" and "1". Consequently, an output can only think these two statuses. For example, a linked motor could thus be either switched on or off, i.e. controlled.

This function has coined the name PLC: Programmable logic controller, i.e. the behavior of input/output is related to that of a pneumatic switching valve or electromagnetic relay controller; the program is saved in a memory of electronic. However, the PLC tasks have quickly multiplied: the functions of timer and counter, setting and resetting of memory, mathematical computing operations all stand for functions, which can be implemented by practically any of PLCs nowadays.

The requirement to be met by PLC's continued to grow up in line with their speedily spreading usage and the automation technology development. Visualization is the representation statuses of machine for instance the control program being executed, through display or monitor. Also controlling, i.e. the facility to intervene in control processes or, alternatively, to make such intervention by unauthorized persons impossible. It also became required to interconnect and harmonize individual systems controlled via PLC by means of automation technology. Therefore a master computer makes easy the means to issue higher-level commands for program processing to some PLC systems.

The networking of some PLCs as well as that of a master computer and PLC is affected through special communication interfaces. To this effect, a lot of the more current PLCs are well-matched with open, standardized bus systems, for instance Profibus to EN 50170.

End of the seventies, binary inputs and outputs were finally extended with the analogue inputs and outputs addition, since many of today's technical applications need analogue processing such as speed setting, force measurement, servo-pneumatic positioning systems. At the same time, the analogue signals acquisition or output allows an actual/set point value comparison and as a result the automatic control engineering realization functions, a task, which broadly exceeds the scope suggested by the name as programmable logic controller.

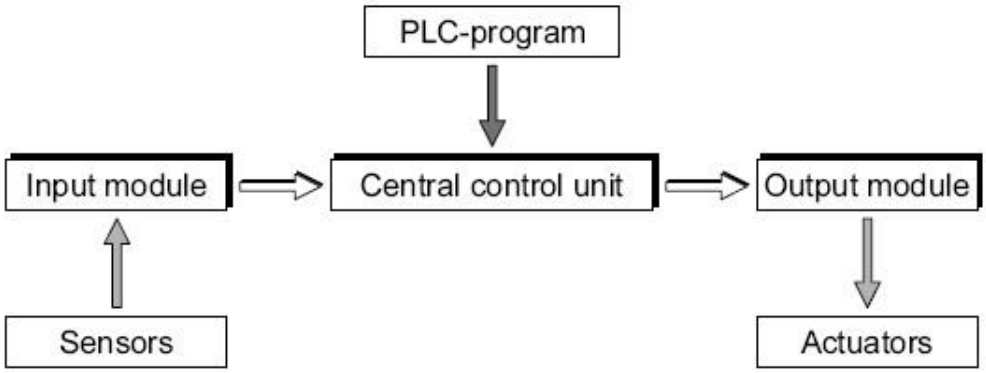


The PLCs presently on offer in the market place have been modified to customer demands to such an extent that it has become possible to buy a highly suitable PLC for virtually any application. As such, miniatures PLCs are currently available with a minimum number of inputs/outputs beginning from just a few hundred Pounds. Also available are larger PLCs with 28 or 256 inputs/outputs. A lot of PLCs can be extended by means of additional input/output, positioning, communication and analogue modules. Special PLCs are available for shipping or mining, safety technology tasks. Yet further PLCs are capable to process numerous programs concurrently or multitasking. Lastly, PLCs are coupled with other automation components, accordingly creating significantly wider areas of application.

1.3 Basic design of a PLC

The programmable logic controller (PLC) term is defined by EN 61131-1 (IEC 61131-1): "A digitally operating electronic system, designed for use in an industrial environment, which uses a programmable memory for the internal storage of user-oriented instructions for implementing specific functions such as logic, sequencing, timing, counting and arithmetic,

to control, through digital or analogue inputs and outputs, various types of machines or processes. Both the PC and its associated peripherals are designed so that they can be easily integrated into an industrial control system and easily used in all their intended functions."



A PLC is consequently nothing more than a computer, modified specifically for firm control tasks. The input module function is to convert incoming signals into signals, which can be processed by the PLC, and to pass these to the central control unit. The reverse task is executed by an output module. This converts the PLC signal into signals appropriate for the actuators. The actual signals processing is affected in the central control unit in compliance with the program saved in the memory. The PLC program can be created in a variety of methods: through assembler kind instructions in 'statement list', in higher-level, problem-oriented languages for example structured text or in the form of a flow chart such as represented by a sequential function chart. In Europe, the use of function block diagrams based on function charts with graphic symbols for logic gates is extensively used. In America, the ladder diagram is the chosen language by users.

Depending on how the central control unit is linked to the modules of input and output, differentiation can be created between compact PLCs (input module, central control unit and output module in one housing) or modular PLCs.

1.4 The new PLC standard EN 61131 (IEC 61131)

Before valid PLC standards focusing mostly on PLC programming were usually geared to current state of the technology of art in Europe at the end of the seventies. This took into account non-networked systems of PLC, which mainly perform logic operations on binary signals. Previously, no comparable, standardized language parts existed for the developments of PLC and system expansions created in the eighties, for example interconnection of intelligent modules, processing of analogue signals, networked PLC systems etc. accordingly, and PLC systems by different manufacturers required totally different programming. Since 1992, an international standard now exists for programmable logic controllers and associated peripheral devices (programming and diagnostic tools, testing equipment, man-to-machine interfaces etc.). In this context, a device built by the user and consisting of the above components is known as a PLC system.

The new EN 61131 (IEC 61131) standard consists of five parts:

- Part 1: General information
- Part 2: Equipment requirements and tests
- Part 3: Programming languages
- Part 4: User guidelines (in preparation with IEC)
- Part 5: Messaging service specification (in preparation with IEC)

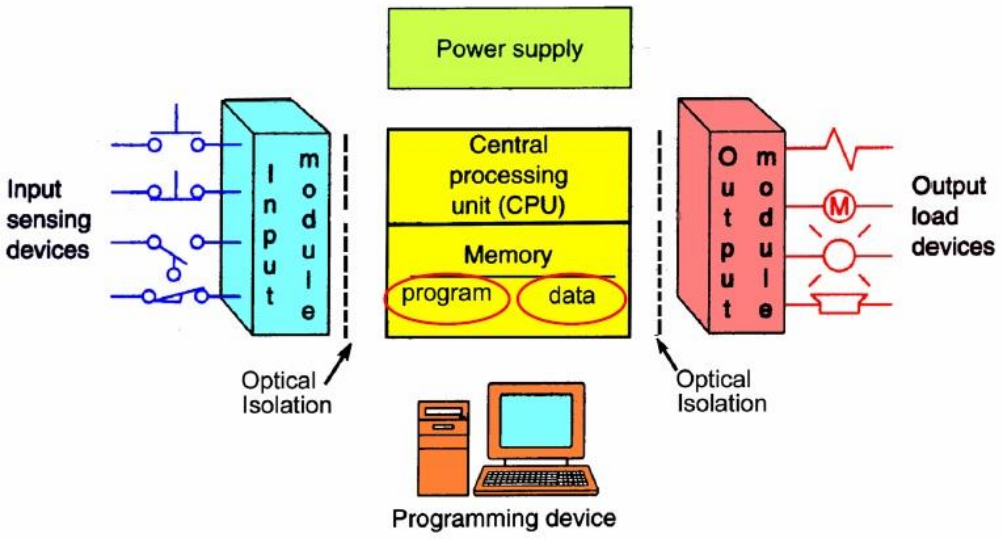
Chapter 2

Design and mode of operation of a PLC

2.1 Structure of a PLC

The PLC which is being a device of microprocessor based, has an analogous internal structure to a lot of embedded computers and controllers. They comprise the devices of CPU, Memory and I/O. These components are integral to the controller of PLC. In addition the PLC has a connection for the Programming and Monitoring Unit, Program Recorder and Printer.

This is shown in a block diagram below:



Dependent on the PLC system type i.e. small, medium or large the component parts are all housed in one compact unit (small PLC) or distributed. The distributed system has the module of CPU/memory, I/O racks and remote I/O units which may be hundreds of meters away from the main module of PLC. The larger units of PLC may also have analog units of input/output and provision for extra complex control programs that support arithmetic and other complex processes not initially present in relay logic controllers.

The key differences between PLCs and other microprocessor based devices are that PLC are rugged design units for an industrial setting and are shielded for enhanced electrical noise immunity. Additionally they are modular, allowing simple replacement and addition of units. They support signal levels and standardized I/O connections are designed for the easy programming, to let personnel unfamiliar with computer languages to program the PLCs in-plant.

The capabilities of the systems PLC are not present in previous relay logic systems are the PID control, analog I/O, and interfaces to a central PLC or a controlling computer.

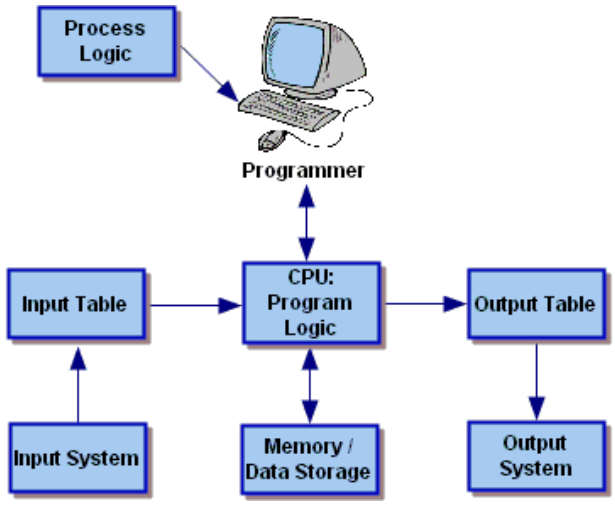
PLC Components

The CPU utilized in a system of PLC is a standard CPU present in a lot of other microprocessor controlled systems. The choice of the CPU depends on the process to be controlled. Generally 8 or 16 bit CPUs fulfill the requirements sufficiently.

A PLC system Memory is separated into the program memory which is generally saved in EPROM/ROM, and the operating memory. The RAM memory is required for the program operation and the input and output data temporary storage. Typical PLC systems memory sizes are around 1 kb for small PLCs, few kbs for medium sizes and greater than 10 to 20 kb for larger PLC depending on the requirements. A lot of PLC would support easy memory upgrades.

PLC Operation

The PLC operates internally in a way very similar to computers. The inputs are continuously monitored and copied from the I/O module into RAM memory which is divided into the input and output sections. The CPU steps through the control program in another section of the memory and fetches the input variables from the input RAM. Depending on the program and the state of inputs, the output RAM is filled with the control variables which are then copied into the I/O module where they control the processes.



PLC Programming

One of the major benefits of the PLC controller is that it is a programmable device, which builds it possible, unlike in the relay logic, to simply design and adapt the control program or process without any changes in the wiring. To create the PLC systems programming easy and efficient, industry standards defining the programming languages and the programming approach used were adopted. This reduces the requirement for personnel training by creating a set of languages standard for all platforms of PLC on the market. Knowing the PLC programming standards and programming languages is consequently one of the most important considerations for anyone involved in the area of PLC.

2.2 Function mode of a PLC

Modes of operation

A processor has basically 2 modes of operations. They are the program mode or some variation of the Run mode. Program mode may be used to

- Enter a new program
- Upload and download files
- Edit or update existing program
- Change software configurations
- Document programs.

When the PLC is switched into the program mode, all outputs from PLC are forced off regardless of their rung logic status and the ladder I/O scan sequence is halted.

Variations of the Run mode

- Run Mode – it is used to execute the user program. Input devices are monitored and output devices are energized accordingly.
- Test Mode – it used to operate, or monitor the user program without energizing any outputs.
- Remote Mode – it allows the PLC to be remotely changed between program and run mode by a personnel computer connected to PLC processor.

2.3 Application program memory

Advanced ladder logic functions such as timers and counters allow controllers to perform calculations, make decisions and do other complex tasks. They are more complex than basic input contacts and output coils and they rely upon data stored in the memory of the PLC. The memory of the PLC is organized to hold different types of programs and data. This chapter will discuss these memory types.

The memory in a PLC is divided into program and variable memory. The program memory contains the instructions to be executed and cannot be changed while the PLC is running. (Note:

some PLCs allow on-line editing to make minor program changes while a program is running.) The variable memory is changed while the PLC is running. In ControlLogix the memory is defined using variable names (also called tags and aliases).

The PLC has a list of 'Main Tasks' that contain the main program(s) run each scan of the PLC. Additional programs can be created that are called as subroutines. Valid program types include Ladder Logic, Structured Text, Sequential Function Charts, and Function Block Diagrams. Program files can also be created for 'Power-Up Handling' and 'Controller Faults'. The power up programs are used to initialize the controller on the first scan. In previous chapters this was done in the main program using the 'S:FS' bit. Fault programs are used to respond to specific failures or issues that may lead to failure of the control system. Normally these programs are used to recover from minor failures, or shut down a system safely.

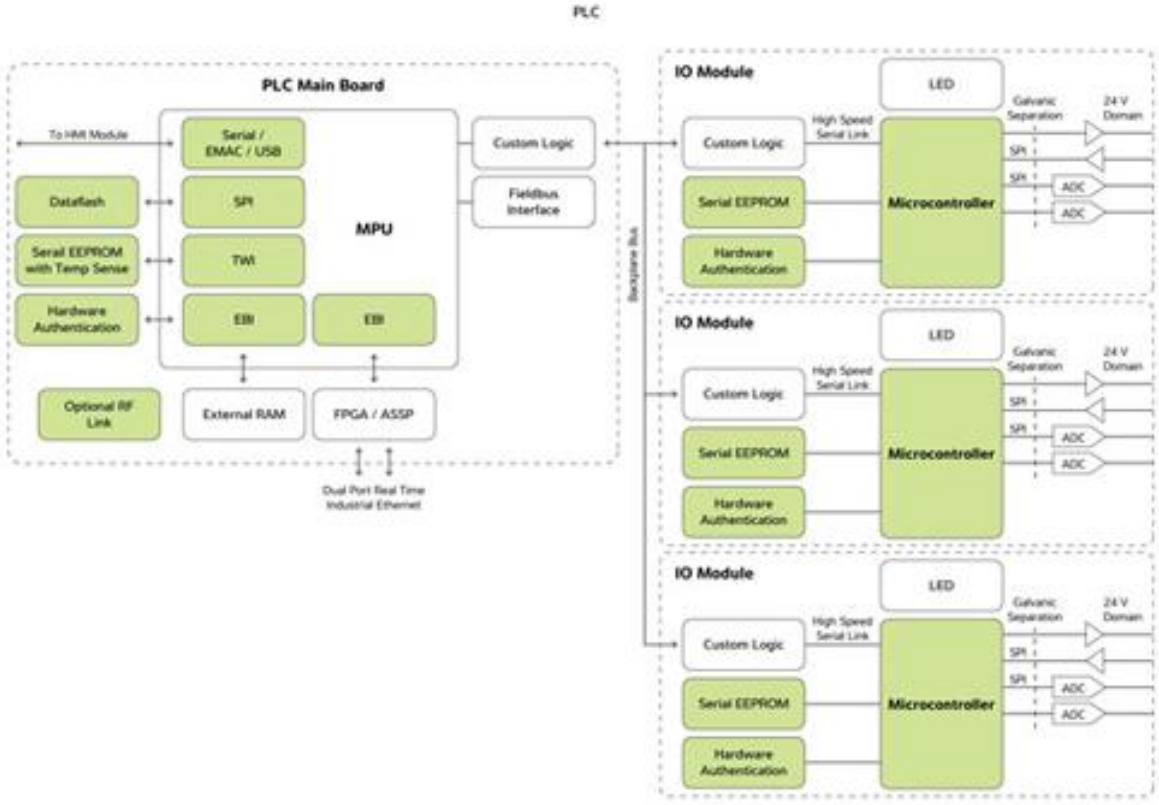
2.4 PLC and IO Devices

- Main CPU applications can utilize the Atmel SAM9 family, which enables developers to reduce the cost of the main CPU board without compromising on the system performance and functionality.
- IO modules (digital, analog or safety module) have a variety of requirements, depending on the product. The Atmel AVR and ARM-based microcontroller families offer diverse package, flash size, and peripheral sets to meet a range of needs.

For IO module solutions:

- High-speed serial peripherals for a fast communication with backplane bus interface or the connection to high resolution external ADC or DAC, with SPI data rates up to 48Mbps on the SAM3U. CAN modules are available on Atmel AVR UC3, megaAVR and AT91SAM microcontrollers.
- Numerous 16-bit timers with input capture function for time stamping.
- PWM channels support control and dim functions for LEDs.

- Atmel supports a rich set of analog functions such as 12-bit ADC and DAC, as well as analog comparator for monitoring the operation condition of the IO-module.
- Safety functions ease the implementation of Safety Integrity Levels (SIL) IEC61508 standard.
- High performance CPU up to 96MHz with integrated MAC unit supports the growing demand for signal conditioning on the analog IO-module.



Chapter 3

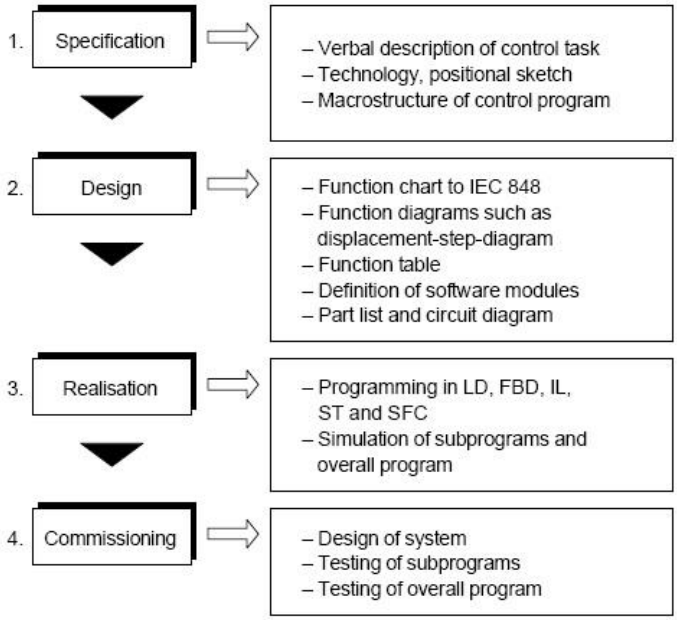
Programming of a PLC

3.1 Systematic solution finding

Control programs must be systematically designed well structured and fully documented in order to be as error-free low-maintenance cost effective as possible

Phase model of PLC software generation

The procedure for the development of a software program illustrated in figure below has been tried and tested. The division into defined sections leads to targeted, systematic operation and provides clearly set out results, which can be checked against the task. The phase model consisting of the following sections:



The phase model can be applied to control programs of varying complexity; for complex control tasks the use of such a model is absolutely essential. The individual phases of the model are described below.

Phase 1: Specification (Problem formulation)

In this phase, a precise and detailed description of the control task is formulated. The specific description of the control system function, formalized as much as possible, reveals any conflicting requirements, misleading or incomplete specifications. The following are available at the end of this phase: Verbal description of the control task Structure/layout Macro structuring of the system or process and thus rough structuring of the solution

Phase 2: Design (Concrete form of solution concept)

A solution concept is developed on the basis of the definitions established in phase 1. The method used to describe the solution must provide both a graphic and process oriented description of the function and behavior of the control system and be independent of the technical realization.

These requirements are fulfilled by the function chart (FCH) as defined in DIN 40 719, Part 6 or IEC 848. Starting with a representation of the overall view of the controller (rough structure of the solution), the solution can be refined step by step until a level of description is obtained, which contains all the details of the solution (refinement of rough structure). In the case of complex control tasks, the solution is structured into individual software modules in parallel with this. These software modules implement the job steps of the control system. These can be special Functions such as the realization of an interface for visualization or communications systems, or equally permanently recurring job steps. The displacement-step diagram represents another standard form for the description of control systems apart from the function chart to DIN 40 719.

Phase 3: Realization (Programming of solution concept)

The translation of the solution concept into a control program is effected via the programming languages defined in IEC 1131-3. These are: sequential function chart, function block diagram, ladder diagram, statement list and structured text. Control systems operating in a time/logic process and available in FCH to DIN 40 719, P.6, can be clearly and easily programmed in a sequential function chart. A sequential function chart, in as far as possible, uses the same

components for programming as those used for the description in the function chart to DIN 40 719, T.6.

Ladder diagram, function block diagram and statement list are the programming languages suitable for the formulation of basic operations and for control systems which can be described by simple operations logic operations or Boolean signals. The high-level language structured text is mainly used to create software modules of mathematical content, such as modules for the description of control algorithms. In so far as PLC programming systems support this, the control programs or parts of a program created should be simulated prior to commissioning. This permits the detection and elimination of errors right at the initial stage.

Phase 4: Commissioning (Construction and testing of the control task)

This phase tests the interaction of the automation system and the connected plant. In the case of complex tasks, it is advisable to commission the system systematically, step by step. Faults, both in the system and in the control program, can be easily found and eliminated using this method.

3.2 EN 61131-3 (IEC 61131-3) structuring resources

IEC 61131-3 is the first vendor independent standardized programming language for industrial automation. Established by the International Electro technical Commission (IEC) a worldwide standard organization founded in 1906 and recognized worldwide for standards in the controls industry by over 50 countries. The standard is already well established in Europe and is rapidly gaining popularity in North America and Asia as the programming standard for industrial and process control.

The adoption of IEC 61131-3 by the industry is driven by the increasing software complexity of control and automation requirements. The time to create, labor cost, and maintainability of control software has a major impact on control projects which can be improved using the IEC 61131-3 vendor independent programming language standard. Applying a standard programming language has a positive impact on the software life-cycle that includes requirements analysis, design, construction, testing (validation), installation, operation, and maintenance. The impact on

maintenance is important since control software maintenance, including upgrades, is generally 2-4 times the labor of initial programming.

The IEC 61131-3 standard combined with new powerful free scale chip architectures enables an entire controller to be delivered in an embedded device. Control programs can run distributed and independently rather than concentrated in large controllers. No longer are thousands of lines of control programs required running in one controller for complex automation applications. This increases performance, improves reliability, and simplifies programs.

IEC 61131-3 provides multiple language support within a control program. The control program developer can select the language that is best suited to a particular task, greatly increasing their productivity. Plus with a standardized programming interface that is completely independent of the hardware platform, users can greatly reduce the cost of program maintenance and training across company wide automation applications.

IEC 61131-3 is hardware independent. The ability to transport automation solutions to other platforms is vastly improved over PLC applications offering users and System Integrators a level of reusability never before available. IEC 61131 increases the efficiency and speed of implementing new automation solutions by using readily available control components developed on other projects and by outside developers.

Companies that have chosen to implement IEC 61131-3 find that they reduce human resource costs in training, debugging and maintenance, and improve productivity from the higher reusability.

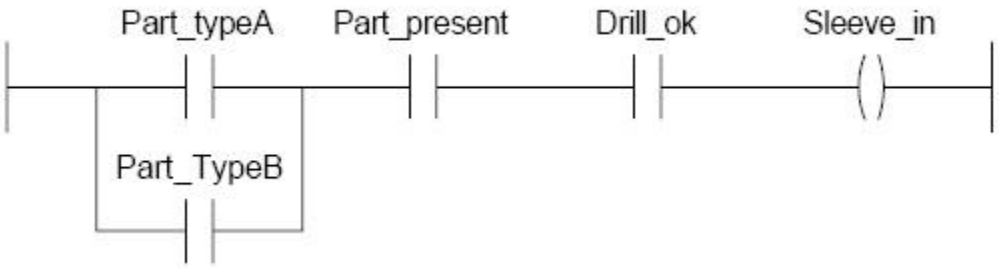
3.3 Programming Languages

The languages can be mixed in any way within a PLC project. The unification and standardization of these five languages represent a compromise of historical, regional and branch-specific requirements. Provision has been made for future expansion, (such as the function block principle or the language Structured Text) plus necessary information technology details (data type etc.) have been incorporated.

The language elements are explained with the help of a machining process involved in valve production. Two sensors are used to establish whether a work piece with correctly drilled holes is available at the machining position. If the valve to be machined is of type A or type B – this is set via two selector switches – the cylinder advances and presses the sleeve into the drilled hole.

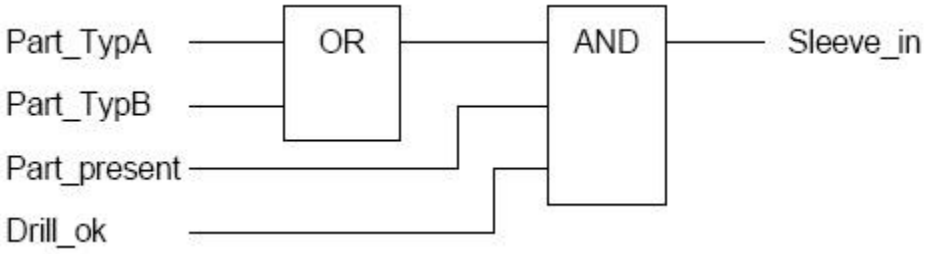
Ladder Diagram (LD)

Ladder diagram is a graphic programming language derived from the circuit diagram of directly wired relay controls. The ladder diagram contains contact rails to the left and the right of the diagram; these contact rails are connected to switching elements (normally open/normally closed contacts) via current paths and coil elements.



Function Block Diagram (FBD)

In the function block diagram, the functions and function blocks are represented graphically and interconnected into networks. The function block diagram originates from the logic diagram for the design of electronic circuits.



Instruction List (IL)

Statement list is a textual assembler-type language characterized by a simple machine model (processor with only one register). Instruction list is formulated from control instructions consisting of an operator and an operand.

```
LD    Part_TypeA
OR    Part_TypeB
AND   Part_present
AND   Drill_ok
ST    Sleeve_in
```

With regard to language philosophy, the ladder diagram, the function block diagram and instruction list have been defined in the way they are used in today's PLC technology. They are however limited to basic functions as far as their elements are concerned. This separates them essentially from the company dialects used today. The competitiveness of these languages is maintained due to the use of functions and function blocks.

Structured Text (ST)

Structured text is high-level language based on Pascal, which consists of expressions and instructions. Instructions can be defined in the main as: Selection instructions such as IF...THEN...ELSE etc., repetition instructions such as FOR, WHILE etc. and function block invocations.

```
Sleeve_in := (Part_TypeA OR Part_TypeB) AND Part_present AND Drill_ok;
```

Structured text enables the formulation of numerous applications, beyond pure function technology, such as algorithmic problems (high order control algorithms etc.) and data handling (data analysis, processing of complex data structures etc.).

Sequential Function Chart (SFC)

The sequential function chart is a language resource for the structuring of sequence-oriented control programs. The elements of the sequential function chart are steps, transitions, alternative and parallel branching. Each step represents a processing status of a control program, which is

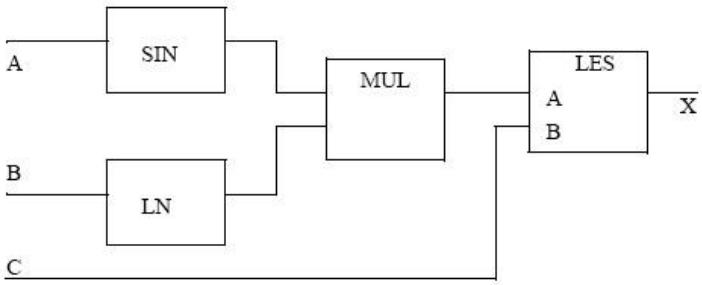
active or inactive. A step consists of actions which, identical to the transitions, are formulated in the IEC 1131-3 languages. Actions themselves can again contain sequence structures. This feature permits the hierarchical structure of a control program. The sequential function chart is therefore an excellent tool for the design and structuring of control programs.

Chapter 4

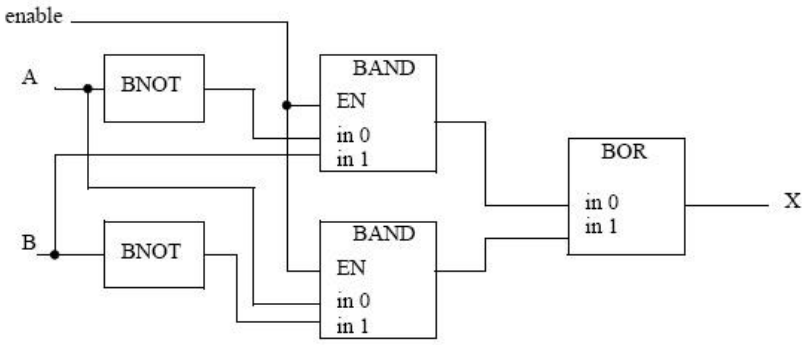
PLC Programming Language

4.1 Function Block Diagram

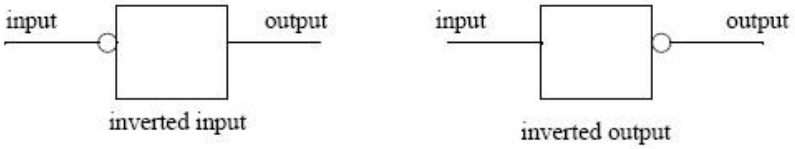
Function Block Diagrams (FBDs) are another part of the IEC 61131-3 standard. The primary concept behind a FBD is data flow. In these types of programs the values flow from the inputs to the outputs, through function blocks. A sample FBD is shown in figure below. In this program the inputs *A* and *B* are used to calculate a value *sin(A) * ln(B)*. The result of this calculation is compared to *C*. If the calculated value is less than *C* then the output *X* is turned on, otherwise it is turned off. Many readers will note the similarity of the program to block diagrams for control systems.



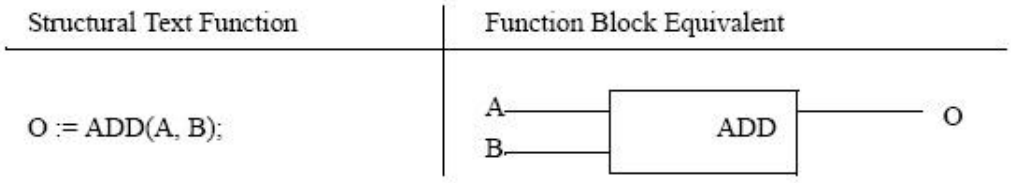
It is possible to disable part of the FBDs using enables. These are available for each function block but may not be displayed. Figure 300 shows an XOR calculation. Both of the Boolean AND functions have the enable inputs connected to ’enable’. If ’enable’ is true, then the system works as expected and the output ’X’ is the exclusive OR of ’A’ and ’B’. However if ’enable’ is off then the BAND functions will not operate. In this case the ’enable’ input is not connected to the BOR function, but because it relies on the outputs from the BAND blocks, it will not function, and the output ’X’ will not be changed.



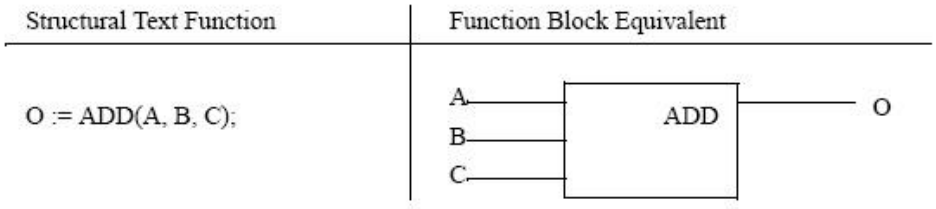
A FBD program is constructed using function blocks that are connected together to define the data exchange. The connecting lines will have a data type that must be compatible on both ends. The inputs and outputs of function blocks can be inverted. This is normally shown with a small circle at the point where the line touches the function block, as shown in figure below.



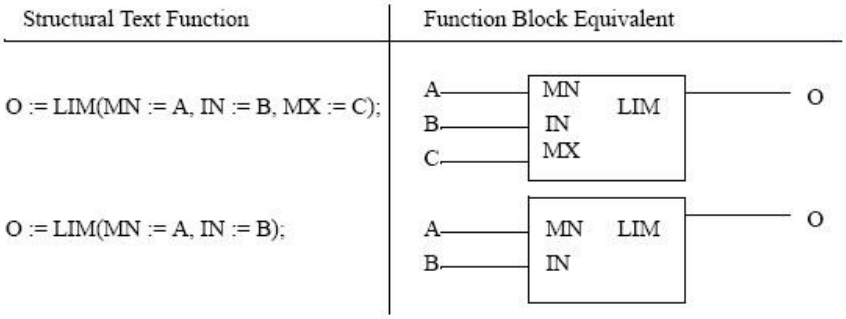
The basic functions used in FBD programs are equivalent to the basic set used in Structured Text (ST) programs. Consider the basic addition function shown in figure below. The ST function on the left adds *A* and *B*, and stores the result in *O*. The function block on the right is equivalent. By convention the inputs are on the left of the function blocks, and the outputs on the right.



Some functions allow a variable number of arguments. In Figure below there is a third value input to the *ADD* block. This is known as overloading.

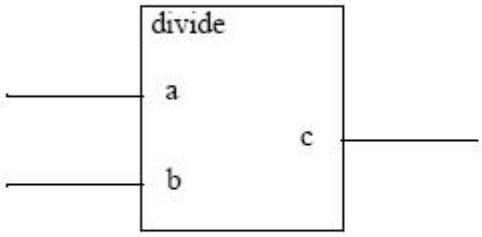


The ADD function in the previous example will add all of the arguments in any order and get the same result, but other functions are more particular. In the first ST function the maximum *MX*, minimum *MN* and test *IN* values are all used. In the second function the *MX* value is not defined and will default to *0*. Both of the ST functions relate directly to the function blocks on the right side of the figure.



Creating Function Block

When developing a complex system it is desirable to create additional function blocks. This can be done with other FBDs, or using other IEC 61131-3 program types. Figure below shows a divide function block created using ST. In this example the first statement declares it as a *FUNCTION_BLOCK* called *divide*. The input variables *a* and *b*, and the output variable *c* are declared. In the function the denominator is checked to make sure it is not *0*. If not, the division will be performed; otherwise the output will be zero.




```
FUNCTION_BLOCK divide
VAR_INPUT
        a: INT;
        b: INT;
END_VAR
VAR_OUTPUT
        c: INT;
END_VAR
        IF b <> 0 THEN
        c := a / b;
        ELSE
        c := 0;
        END_IF;
END_FUNCTION_BLOCK
```

4.2 Ladder Diagram

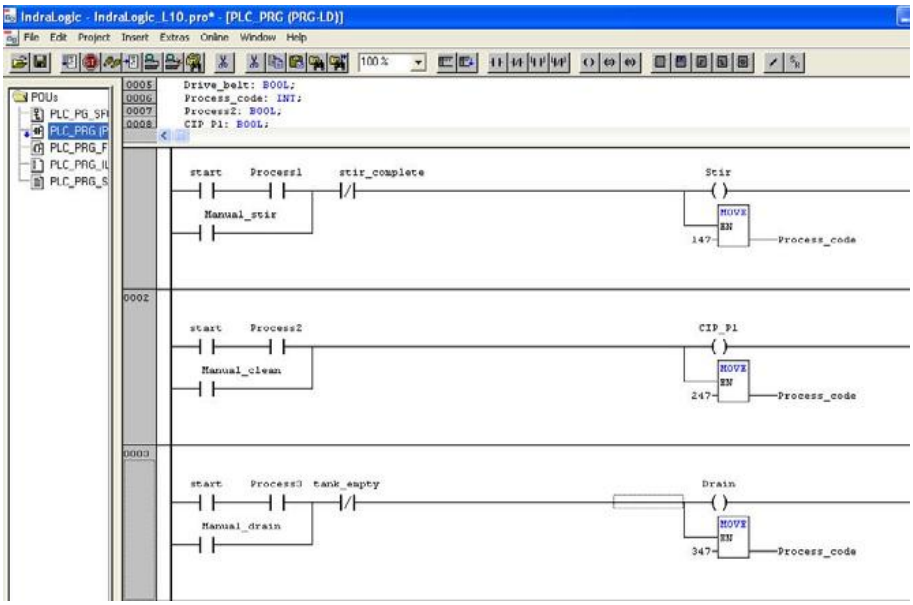
Ladder Diagram (LD) This programming language, invented in the U.S. decades ago, is probably the most widely used. Invented to replace hardwired relay control systems, Ladder Diagram programming is a mainstay in the U.S. today, used in probably 95 percent of all applications. Visually, this language resembles a series of control circuits, with a series of inputs needing to be "made" or "true" in order to activate one or more outputs.

Ladder Diagram language has experienced such widespread adoption that almost every programmer in any country or industry can read and write this language. Because it resembles the familiar electric circuit format, even a non-programmer with an electrical background can follow the program for purposes of troubleshooting a problem. It's also easy to start writing a program in Ladder Diagram. With just a basic outline of input and output signals, one can sit down are start churning out code. Most of the other IEC languages require more preparation, such as flowcharting all the potential process flows.

Finally, most implementations of Ladder Diagram allow a program to be organized into folders or subprograms that are downloaded to the PLC, allowing for easy segmentation. Ladder Diagram programming is ideal for a simple material handling application, for example, where a sensor detects the presence of a box, other sensors check for obstructions, and then an output fires an actuator to push the box to another conveyor. Digital inputs are checking for various

conditions, and a basic program is analyzing the inputs and firing digital outputs in response. There may be timers in the program, or some basic comparisons or math, but there are no complex functions involved.

As the complexity of PLC functionality has grown, however, Ladder Diagram language has been challenged to meet these advances and still maintain the paradigm of easy visualization and understanding. Functions such as PID, trigonometry and data analysis are commonly required in many control applications, but difficult to implement. Another challenge is that as program size grows, the ladder can become very difficult to read and interpret, unless it's extensively documented.

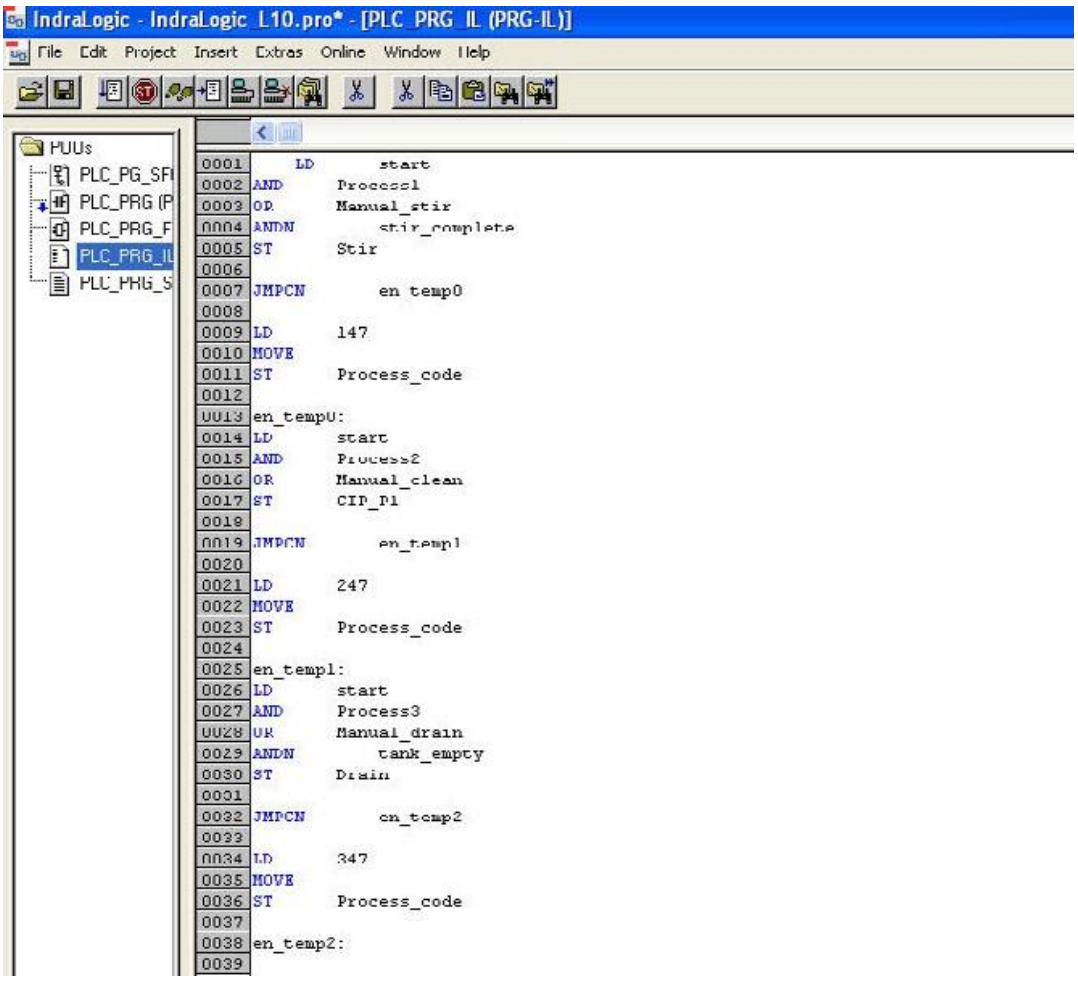


Finally, implementing full processes in Ladder Diagram can be daunting — picture a ladder rung with an output used in several phases of a process with many input conditions attempting to control exactly when that output needs to turn on.

4.3 Instruction List

Anyone who has experience programming microprocessors or experience with Assembler language programming will see similarities with Instruction List programming. This language consists of many lines of code, with each line representing exactly one operation. Thus, it is very step-by-step in layout and format, which makes the entry of a series of simple mathematical functions easy. In addition, if the programmer uses only the IEC defined instructions, a program written in this language can be moved easily between hardware platforms. These advantages make this language very popular in Europe, a fact that is surprising to many U. S. programmers who prefer the ease of maintenance in the graphical languages, and place a lower premium in the transferability of programs.

Instruction List language is a low level language and as such, will execute much faster in the PLC than a graphical language, like Ladder. This language is also much more compact and will consume less space in PLC memory. The simple one line text entry method supported by this language also allows for very fast program entry — no mouse required, no tab to click! In legacy systems, programs written in this language are easier to display and edit on a handheld programming unit, with no software or laptop required.

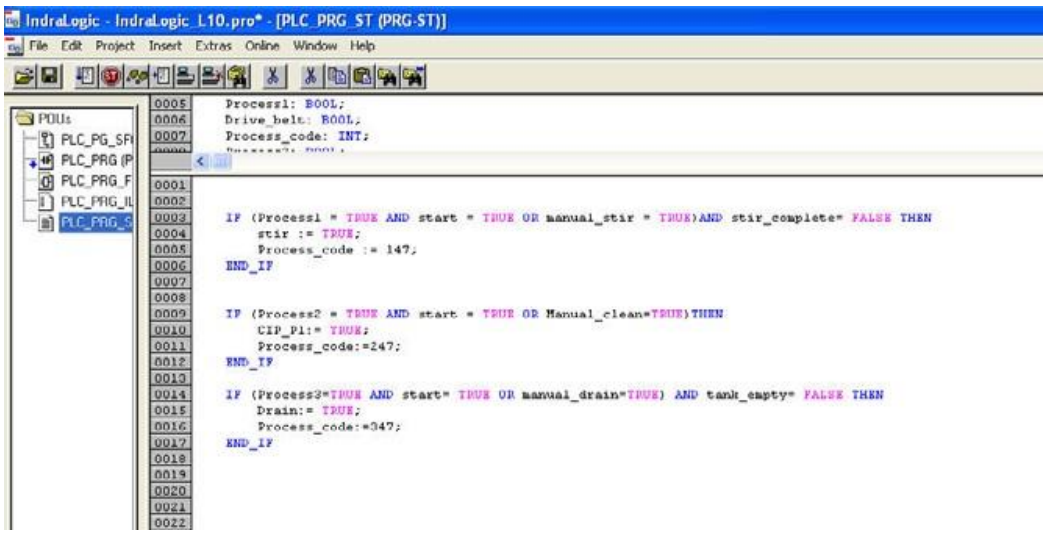


Despite the advantages this language provides to a programmer, it seems that maintenance and service engineers do not prefer Instruction List. Perhaps because it is less visual than Ladder, and Therefore more difficult to get a sense of what the program is doing and what errors it is experiencing. Similar to the issues with Ladder Diagram and increasing PLC program complexity, it can be a struggle to enter complex functions such as PID in Instruction List. This also applies to complex mathematical computations. Instruction List does not lend itself well to any form of structured programming, such as state programming or step ladder, further limiting its usefulness for implementing large programs. It is also arguable that the advantages of speed and compactness are less relevant, given the processing speeds of modern PLCs and the large amounts of memory available.

4.4 Structure Text

With its IF…THEN loops, CASE selectors, and lines ending in semicolons, Structured Text language closely resembles a highlevel computer programming language such as PASCAL or C. Therefore mentioned *Control Engineering* survey indicated that of all the IEC61131-defined programming languages, Structured Text has seen the greatest increase in adoption. This language perhaps best embraces the growing complexity of PLC programming, such as the process control functions involved in plastics or chemical manufacturing. Trigonometry, calculus, and data analysis can be implemented far easier in this language than in Ladder or Instruction List.

Decision loops and pointers (variables used to do indirect addressing) allow for a more compact program implementation than can be achieved in Ladder. The flexible Structured Text editor that is common in most programming packages makes it easy to insert comments throughout a program, and to use indents and line spacing to emphasize related sections of code. This makes the task of structuring a complex program easier. The text-based, non-graphical nature of Structured Text, similar to Instruction List, also runs much faster than Ladder. An additional benefit of Structured Text is that it comes closer than most of the other languages in achieving the transferability promise of the IEC61131 standard. Copying and pasting Structured Text from the editor of one programming package to another can often be done with just a few changes, emancipating a programmer from the hardware platform. A final benefit is that many students currently graduating from engineering studies have a better background in computer languages than in the basics of electrical wiring, and therefore can be more proficient in Structured Text than Ladder programming.



A disadvantage is that for many previously experienced programmers or maintenance and service personnel, the Structured Text language has seen the greatest increase in adoption and closely resembles a high-level computer programming language such as PASCAL or C. Text environment is somewhat unfamiliar and unsuitable for troubleshooting. In many ways, the code and structure necessary to make this code maintenance friendly can reduce some of the advantages gained from its compactness. As a result, the main tendency is to use Structured Text "behind the scenes." For example, IEC 61131 allows a programmer to build his or her own functions in one language, which can then be used in another language. Thus the programmer is most likely to encapsulate a Structured Text program inside an instruction called on in Ladder. While this may not necessarily be a disadvantage, the programmer will need to thoroughly test any code that is "hidden" and make sure it is bug free, since others will not have access to it.

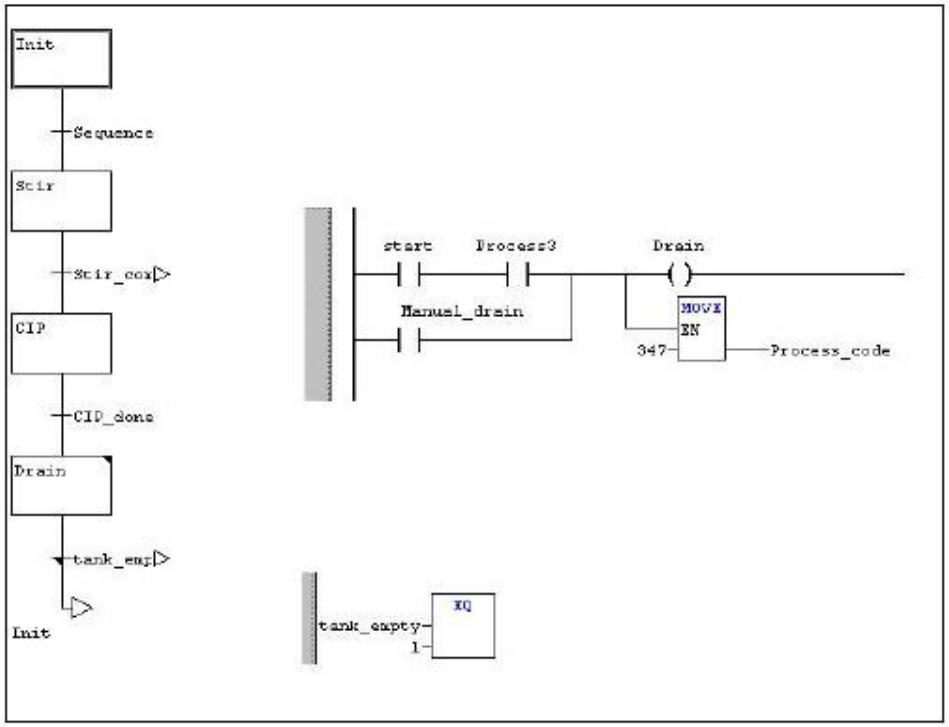
Structured Text (ST) is a high level textual language that is a Pascal like language. It is based on the IEC 61131-3 standard, which standardizes programming languages for programmable controllers (PLC). Structured Text is very flexible and intuitive for writing control algorithms.

Structured Text uses operators such as logical branching, multiple branching, and loops. People trained in computer programming languages often find it the easiest language to use for programming control logic. When symbolic addressing is used, ST programs resemble sentences, making it highly intelligible to beginner users as well.

4.5 Sequential Function Chart

Sequential Function Chart (SFC) programming resembles the computer flowcharts that many will remember drawing up in their college days. An initial step "action box" (the starting point of a flowchart) is followed by a series of transitions and additional action steps. The concept of SFC Is simple: an action box, with code inside written in any language of the programmer's choice, is active until the transition step below it activates. The current action box is for appropriate applications which have a repeatable multi-step process or series of repeatable processes, this form of programming is the easiest to implement. An example would be a pick and place application, where product is constantly picked up from one area, moved through a specific path, and placed in another area. While exceptions exist, since there is typically only one active piece of code and one transition to be concerned with, condition checking and the control of the process should be achievable without large rungs.

The language is also very friendly to maintenance engineers because the visual nature of the program plus code segmentation makes it easy to troubleshoot. For example, if the mechanism in a pick-and place application has moved to the product but not picked it, the troubleshooter could bring up the program and look at the transition condition between the "move to product" box and the "pick product" box to see what is holding up the process. On the downside, this style of programming is not suitable for every application, as the structure that is forced on a program could add unneeded complexity. A large amount of time must be spent up front preparing and planning before any programming is attempted or else the functions charts could become unwieldy and difficult to follow. The overhead required for this type of program causes it to execute slower than the other languages. A final consideration is the inability to convert to other languages. Instruction List, Function Block and Ladder programs can easily be converted into each other, allowing a piece of code to be displayed in the way most comfortable to the user.



Structured Text can also be converted into any of these three languages, but SFC stands alone. It cannot be converted. Therefore, you may want to consider this language only for end users who are comfortable with the language and are unlikely to display it in a different format or for applications where the hardware has the speed and memory necessary to store and execute an SFC program.

Chapter 5

Omron PLC Programming

5.1 Introduction to Omron PLC Programming

5.1.1 What is Control System?

In general, a control system is a collection of electronic devices and equipment hich are in place to ensure the stability, accuracy and smooth transition of a process or a manufacturing activity. It takes any form and varies in scale of implementation, from a power plant to a semiconductor machine. As a result of rapid advancement of technology, complicated control tasks accomplished with a highly automated control system, which may be in the form of PLC (Programmable Logic Controller) and possibly a host computer etc. Besides signal interfacing to the field devices such as, motors, sensors, solenoid valves, operator panel and etc, capabilities in network communication enable a big scale implementation and process co-ordination besides providing greater flexibility in realizing distributed control system.

5.1.2 The Role of the Programmable Controller

In an automated system, the PLC is commonly regarded as the heart of the control system. With a control application program in execution, the PLC constantly monitors the state of the system through the field input devices feedback signal. It will then based on the program logic to determine the course of action to be carried out at the field output devices.

The PLC may be used to control a simple and repetitive task, or a few of them may be interconnected together with other host controllers or host computers through a sort of communication network, in order to integrate the control of a complex process.

5.1.3 Input and Output Devices

Input Devices

Intelligence of an automated system is greatly depending on the ability of a PLC to read in the various types of automatic sensing and manual input field devices.

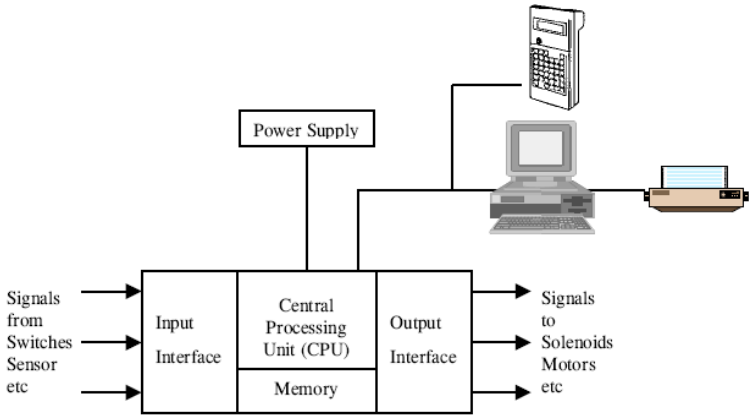
Push buttons, keypad and toggle switches, which form the basic man-machine interface, are types of manual input device. On the other hand, for detection of work piece, monitoring of moving mechanism, checking on pressure and or liquid level and many others, the PLC will have to tap the signal from the specific automatic sensing devices like proximity switch, limit switch, photoelectric sensor, level sensor and so on. Input signal types to the PLC would be ON/OFF logic or analogue. These input signals are interfaced to PLC through various types of PLC input module.

Output Devices

An automatic system is incomplete and the PLC system is virtually paralyzed without means of interface to the field output devices. Some of the most commonly controlled devices are motors, solenoids, relays indicators, buzzer and etc. through activation of the motors and solenoids the PLC can control from a simple pick and place system to a much complex servo positioning system. These type of output devices are the mechanism of an automated system and so its direct effect on the system performance.

5.1.4 What is Programmable Controller?

A PLC consists of a Central Processing Unit containing an application program and Input and output Interface modules, which is directly connected to the field I/O devices. The program controls the PLC, so that when an input signal from an input device turns ON, the appropriate response made. The response normally involves turning ON an output signal to some sort of output devices.



Central Processing Unit

The CPU is a microprocessor that coordinates the activities of the PLC system. It executes the program, processes I/O signals and communicates with external devices.

Memory

There are various types of memory unit. It is the area that hold the operating system and user memory. The operating system is acually a system software that coordinates the PLC, Ladder Program, Timer and Counter Values are stored in the user memory.

a. Read Only memory (ROM)

   ROM is a non-volatile memory that can be programmed only once. It is therefore unsuitable. It is least popular as compared with others memory type.

b. Random Access Memory (RAM)

RAM is commonly used memory type for storing the user program and data. The data in the volatile RAM would be normally be lost if the power source is removed.

c. Erasable Programmable Read only Memory (EPROM)

EPROM holds data permanently just like ROM. It doesn't require battery back up. However its content can be erased by exposing it to ultraviolet light.

d. Electrically Erasable Programmable Read Only memory (EEPROM)

EEPROM combines the access flexibility of RAM and the non-volatility of EPROM in one. Its contents can be erased and reprogrammed electrically, however, to a limit number of times.

5.1.5 PLC Panel and their Advantages

Conventional control panel and its difficulties

In the beginning of the industrial revolution, especially in the 1960 and 1970, automated machines were controlled by electromechanical relays. These relays were all hardwired together inside the control panel was so huge that it could cover the entire wall. Every connection in relay logic must be connected. Wiring is not always perfect, it takes time to troubleshoot the system. It is a very time consuming affair.

Disadvantage of conventional control panel

In this panel we can observe the below points:

- There are too many wiring work in the panel.
- Modification can be quite difficult.
- Troubleshooting can be quite troublesome as you may require a skillful person.
- Power consumption can be quite high as the coil consumes power.
- Machine downtime is usually long when problems occur, as it takes longer time to troubleshoot the control panel.
- Drawings are not updated over the years due to changes. It causes longer downtime in maintenance and modification.

Programmable controller control panel and their advantages

The control design and concept improve tremendously with the arrival of programmable controllers.

Advntages of PLC control panel

- Here are the major advantages that can be distinguishably realized.
- The wiring of the system is usually reduced by 80% compared to conventional relay control system.
- The power consumption is greatly reduced as PLC consumes much less power.
- The PLC self diagnostic functions enable easy and fast troubleshooting of the system.
- Modifications of control sequence or application can easily be done by programming through the console or computer software without changing of I/O wiring, if no additional Input or Output devices are required.
- In PLC system spare parts for relays and hardware timers are greatly reduced as compared to conventional control panel.
- The machine cycle time is improved tremendously due to the speed of PLCs operation is a matter of milliseconds.

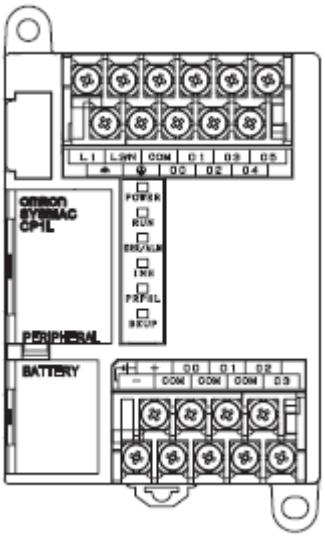
- It cost much less compared to conventional system in situation when the numbers of I/Os is very large and control functions are complex.
- The PLC reliability is higher than the mechanical timers and relays.

5.2 CP1L Overview

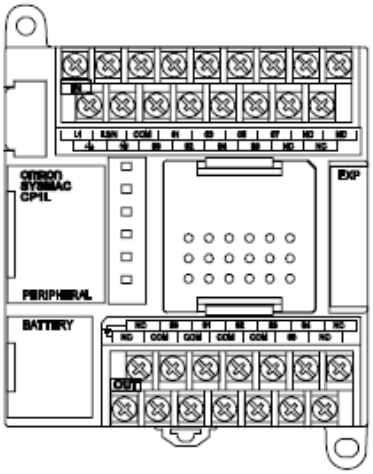
5.2.1 CP1L Models

CP1L programmable controller is a PLC package type, available with 10, 14, 20, 30, 40, or 60 I/O points.

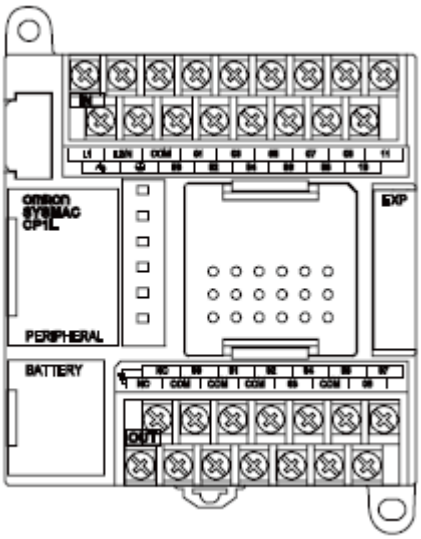
- 10-point I/O Units (CP1L-L10D□-□)
  a. The CPU Unit has 6 inputs and 4 outputs built in.
  b. The PLC cannot use CP-series Expansion I/O Units to expand the maximum total of I/O points.



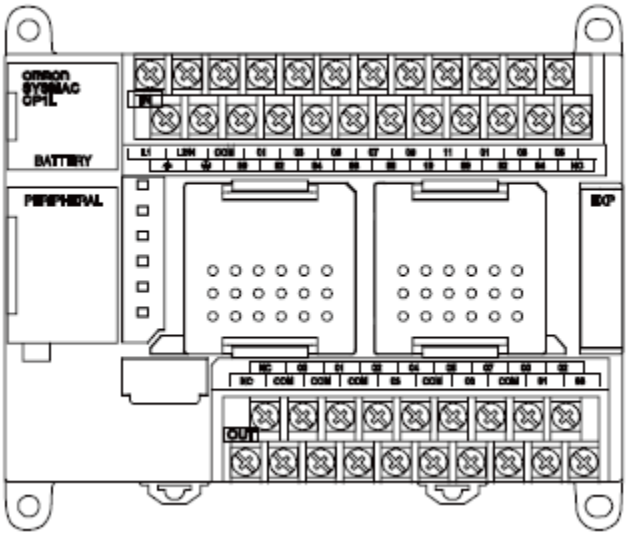
- 14-point I/O Units (CP1L-L14D□-□/CP1L-J14D□-□)
  a. CPU unit has 8 input points and 6 output points.
  b. CP-series expansion I/O units can be used to add I/O points, up to a total of 54 I/O points.



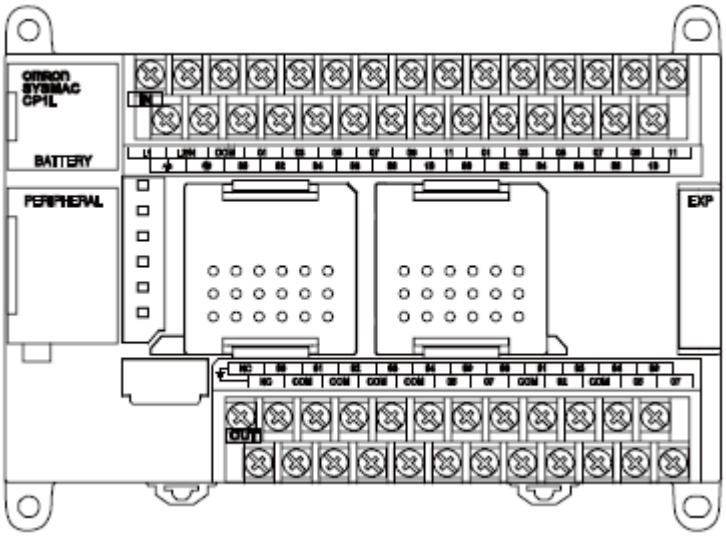
- 20-point I/O Units (CP1L-L20D□-□/CP1L-J20D□-□)
  a. CPU unit has 12 input points and 8 output points.
  b. CP-series expansion I/O units can be used to add I/O points, up to a total of 60 I/O points.



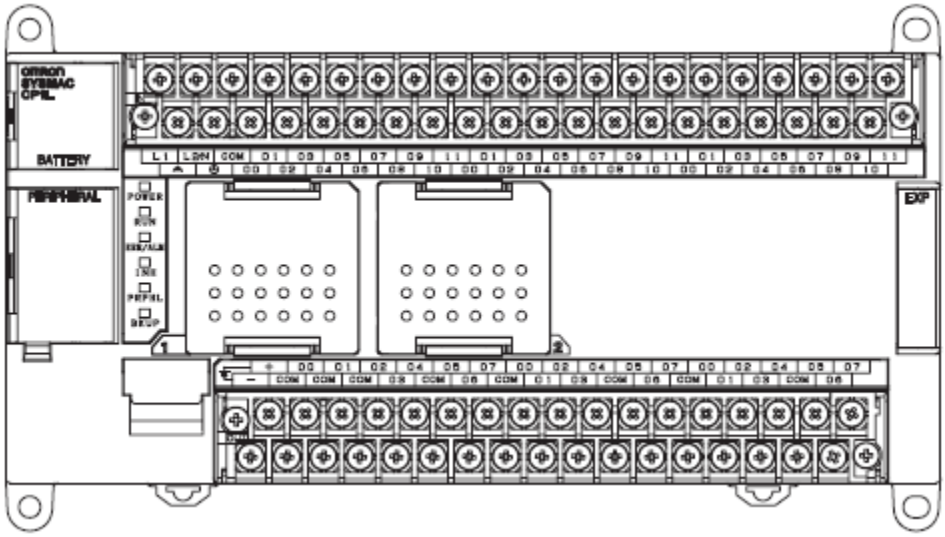
- 30-point I/O Units (CP1L-M30D□-□)
  a. CPU unit has 18 input points and 12 output points.
  b. CP-series expansion I/O units can be used to add I/O points, up to a total of 150 I/O points.



- 40-point I/O Units (CP1L-M40D□-□)
  a. CPU unit has 24 input points and 16 output points.
  b. CP-series expansion I/O units can be used to add I/O points, up to a total of 160 I/O points.



- 60-point I/O Units (CP1L-M60D□-□)
  a. CPU unit has 36 input points and 24 output points.
  b. CP-series expansion I/O units can be used to add I/O points, up to a total of 180

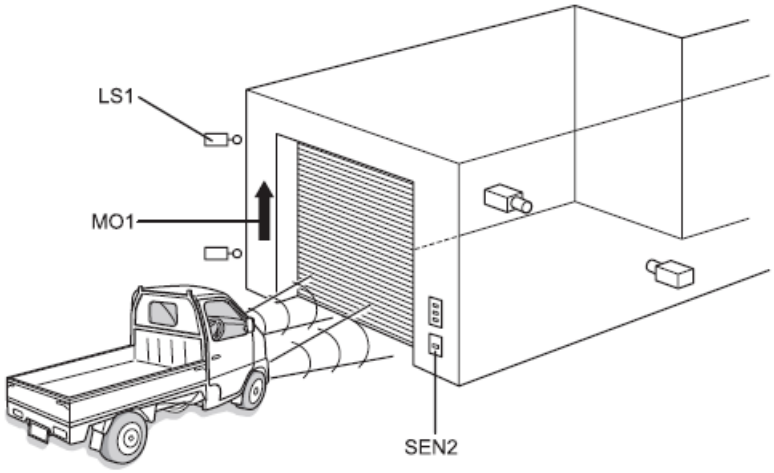


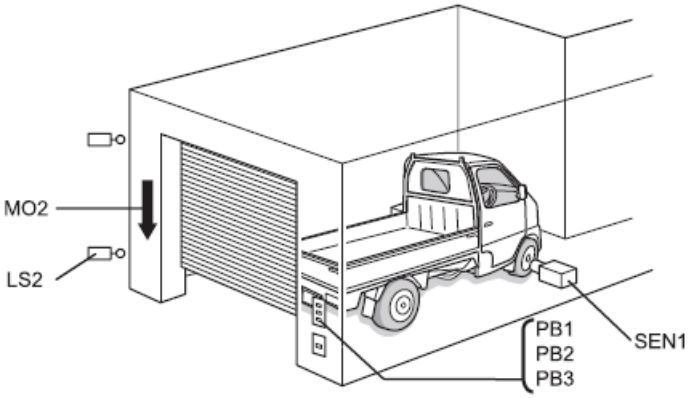
5.2.2 System Components

This section defines components to be used in the shutter control system. The following components are to be used.

1. PLC
   - CP1L (14-point I/O unit with AC power supply)
2. Equipment and Software for Programming
   - CX-Programmer
   - Computer
   - USB cable (A-B)
3. Inputs
   - Shutter OPEN button: PB1
   - Shutter STOP button: PB2
   - Shutter CLOSE button: PB3
   - Car detection sensor: SEN1
   - Headlight detection sensor: SEN2
   - Limit switch, turned ON when shutter is fully open: LS1
   - Limit switch, turned ON when shutter is fully closed: LS2
4. Outputs
   - Contact for activating the shutter escalation motor: MO1

• Contact for activating the shutter de-escalation motor: MO2





5.3 Creating Programs

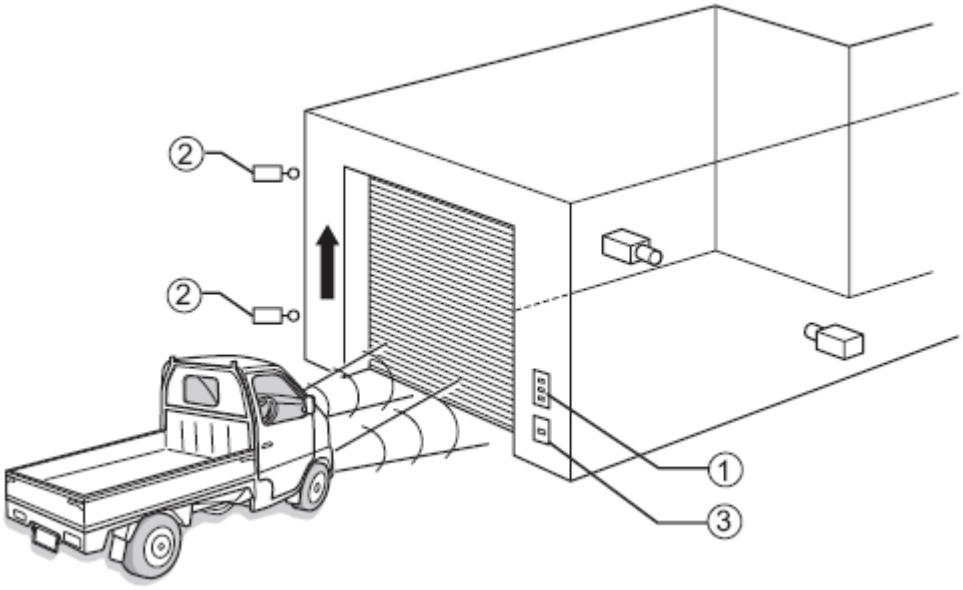
5.3.1 Creating Ladder Programs

A ladder program can now be created for the example introduced in *SECTION 2 System Design*. First, however, the functions of the ladder program will be described.

Operation

The ladder program to be created will open and close a garage shutter.

Entering the Garage



The component functions and operations will be defined in detail below.

(1) Push-buttons:

- The shutter can be opened, closed, and stopped with buttons.
- The OPEN and CLOSE buttons will continue operating the shutter even when they are not held down. A self-maintaining bit is used to achieve this.

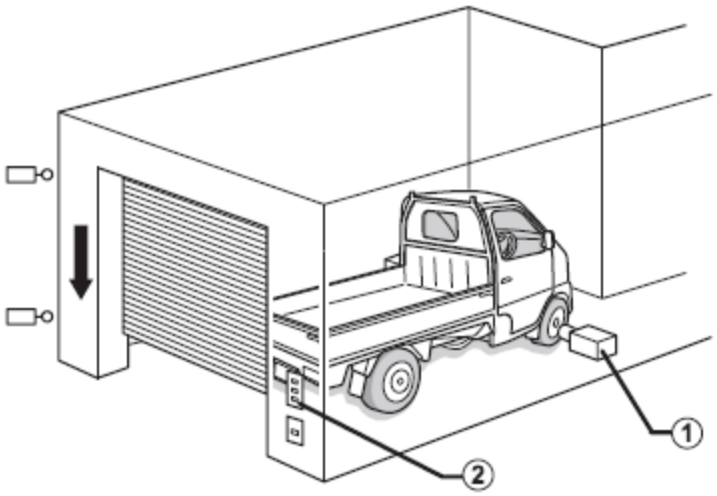
(2) Limit switches:

- When the shutter is fully opened or fully closed, it will be stopped by a limit switch.
- When the shutter is opening, the de-escalation motor will be interlocked to prevent damage.

(3) Light detection sensor:

- A light detection sensor detects light from headlights pointed at the garage. When 3 headlight flashes are detected by a counter instruction, the shutter escalation motor is activated.
- After the first headlight flash, a timer is activated by a timer instruction. After 5 seconds, a reset command is given to the counter instruction.
- The present value of the counter instruction is retained even when CP1L is powered OFF. To prevent malfunction, a reset command is given to the counter instruction when CP1L is powered ON.

After Entering the Garage / Exiting the Garage



(1) Car detection sensor:

- A car detection sensor will detect full car entrance into the garage, and activate the shutter de-escalation motor.

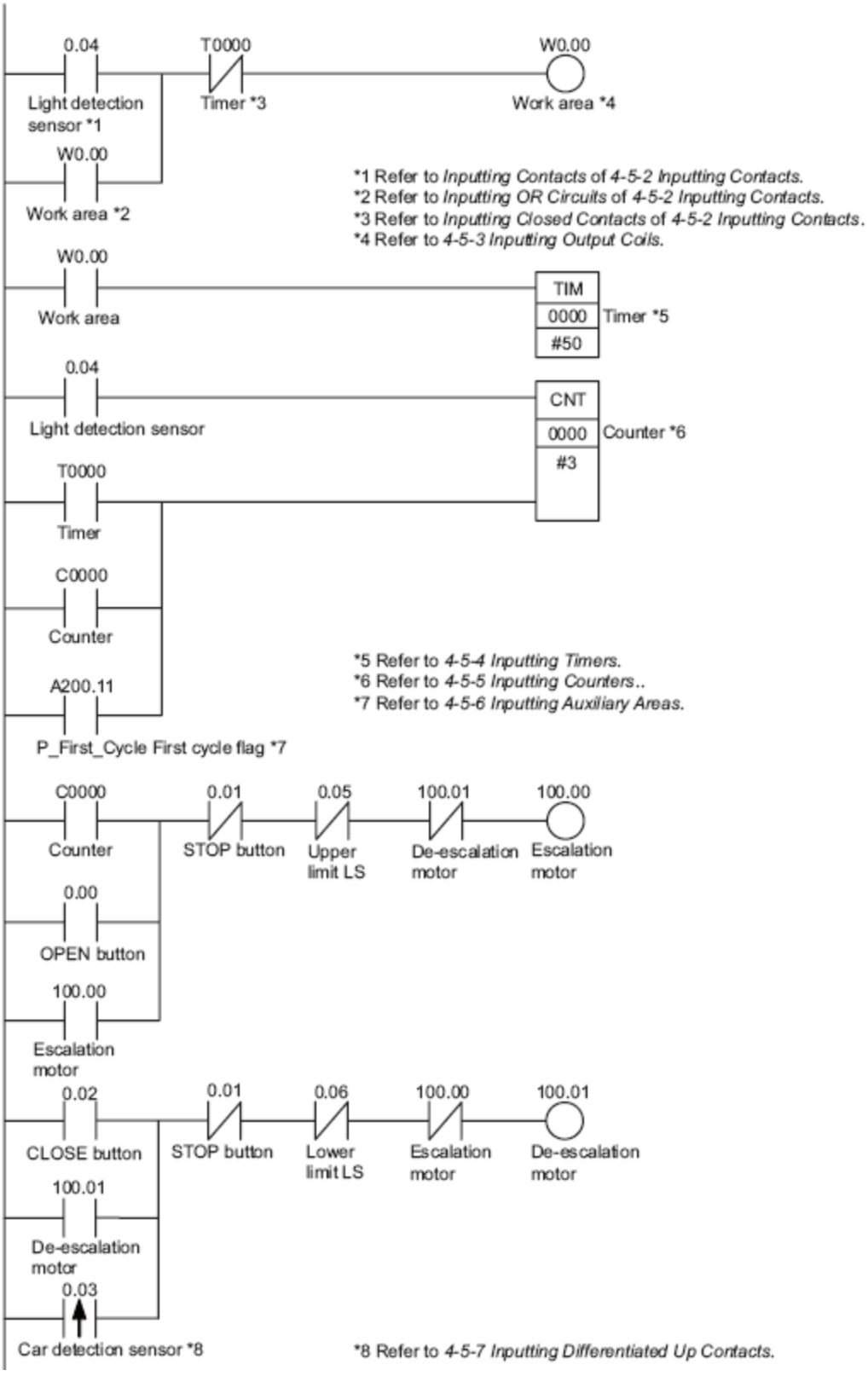
(2) Push-buttons:

- When pulling the car out of the garage, use the buttons to operate the shutter.
- When pulling the car out of the garage, a differentiated up contact should be used as the car detection sensor, so that the shutter does not close immediately upon fully opening.

A ladder program will be set forth hereafter based on the description above.

Ladder Program

The ladder program for the example application is shown below.


0.04
Light detection sensor *1
T0000
Timer *3
W0.00
Work area *4
W0.00
Work area *2
*1 Refer to Inputting Contacts of 4-5-2 Inputting Contacts.
*2 Refer to Inputting OR Circuits of 4-5-2 Inputting Contacts.
*3 Refer to Inputting Closed Contacts of 4-5-2 Inputting Contacts.
*4 Refer to 4-5-3 Inputting Output Coils.
W0.00
Work area
TIM
0000
#50
Timer *5
0.04
Light detection sensor
CNT
0000
#3
Counter *6
T0000
Timer
C0000
Counter
A200.11
P_First_Cycle First cycle flag *7
*5 Refer to 4-5-4 Inputting Timers.
*6 Refer to 4-5-5 Inputting Counters..
*7 Refer to 4-5-6 Inputting Auxiliary Areas.
C0000
Counter
0.01
STOP button
0.05
Upper limit LS
100.01
De-escalation motor
100.00
Escalation motor
0.00
OPEN button
100.00
Escalation motor
0.02
CLOSE button
0.01
STOP button
0.06
Lower limit LS
100.00
Escalation motor
100.01
De-escalation motor
100.01
De-escalation motor
0.03
Car detection sensor *8
*8 Refer to 4-5-7 Inputting Differentiated Up Contacts.

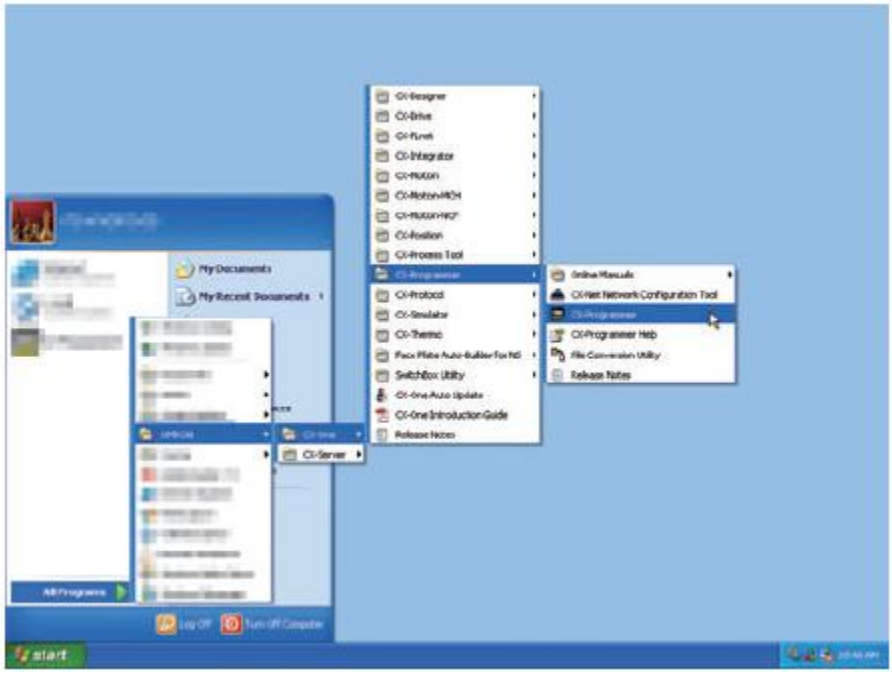
5.3.2 Using CX-Programmer

Starting CX-Programmer

1. On the desktop, select [Start] - [All Programs] - [OMRON] - [CX-One] - [CXProgrammer]

   CX-Programmer will start.

   The title screen will be displayed, followed by the main window.

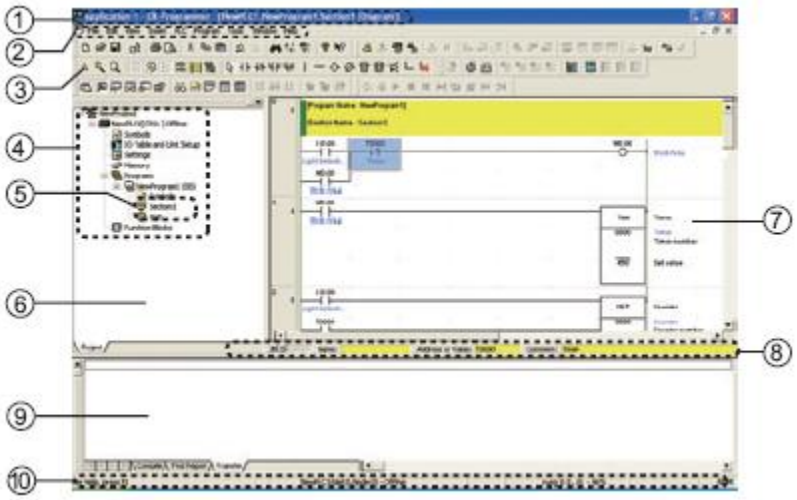


Operation Screens

This section explains the functions available on the CX-Programmer main window. For details on using CX-Programmer.

.

- Main Window



(1) Title bar

- Displays the data file name, created in CX-Programmer.

(2) Main menu

- Used to select CX-Programmer functions.

(3) Toolbars

Displays icons for frequently used functions. Place the mouse cursor over an icon to display the corresponding function name. Select View - Toolbars from the main menu to show/hide toolbars. Drag the toolbars to change their position.

(4) Project tree / (6) Project workspace

Used to manage programs and settings. Drag & drop items to copy the data. Select [View] - [Windows] - [Workspace] from the main menu to show/hide the workspace.

(5) Section

Programs can be split into and managed as multiple parts.

(6) Diagram workspace

Used to create and edit ladder programs.

(7) I/O comment bar

Displays the name, address/value, and I/O comment for the variable selected by the mouse cursor.

(8) Output window

Select [View] - [Windows] - [Output] from the main menu to show/hide the output window. Displays the following information:

- Compile:

  Displays program check results.

- Find Report:

  Displays search results for contacts, instructions, and coils.

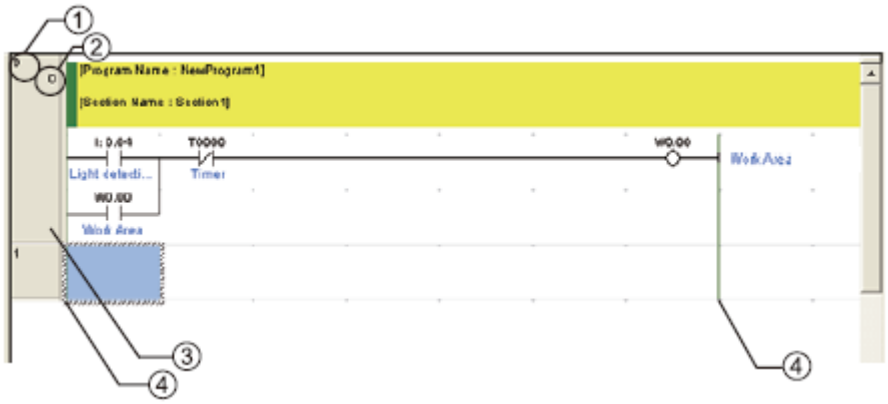
- Transfer:

  Displays errors which occurred while loading a project file.

(10) Status bar

Displays information such as PLC name, offline/online status, and active cell position. If an online connection error or other errors occur and are recorded by the error log while online, a blinking red error message will be displayed. Select [View] - [Windows] - [Status Bar] from the main menu to show/hide the status bar.

- Diagram Workspace



(1) Rung number

(2) Program address

(3) Rung header

If a rung is incomplete, a red line will be displayed to the right of its rung header.

(4) Bus bar

- Information Window

Displays basic shortcut keys used in CX-Programmer.

Select [View] - [Windows] - [Information Window] from the main menu to show/hide the information window.

Chapter 6

Mitsubishi MELSEC-F Programmable Controller

6.1 Introduction to FX Series Programmable Controller

6.1.1 Overview

This chapter gives the explanation on all aspects of operation and programming for FX, FX2C, FX0N, FX0S, FX0, FX2N, and FX2NC programmable controllers (PLCs). This is covers the functions of the highest specification PLC. For this reason, the following indicator is included to show which PLCs applies to:

$FX_{0(S)}$	FX_{0N}	FX	$FX_{(2C)}$	$FX_{2N(C)}$

Shaded boxes indicate the applicable PLC type

- "$FX_{0(S)}$" - All FX_0 and FX_{0S} PLCs
- "FX_{0N}" - All FX_{0N} PLCs
- "FX" - All FX and FX_2 PLCs (CPU ver 2.30 or earlier)
- "$FX_{(2C)}$" - All FX and FX_2 PLCs (CPU versions 3.07 or later)
- - All FX_{2C} PLCs (see page 1-4)
- "$FX_{2N(C)}$" - All FX_{2N} and FX_{2NC} PLCs

FX Family

This is a generic term which is often used to explain all programmable controllers without identifying individual types or model names.

6.1.2 FXon CPU versions

Over time Mitsubishi adds newer and better features to develop and enhance the products. Because of the nature of PLCs, that can be likened to industrial computers, changes sometime occur within the units main CPU. These changes are similar to those experienced by office and home computer users, that is going to a version up processor. The following lists identify the CPU versions that had significant upgrades or new functions and features added.

FX0N CPU versions

- CPU Ver 1.20 the folowing features were added: software control for protocol 1 and 4 communications with the FX0N-485ADP, 1:N network.
- CPU Ver 1.40 the following features were added: software control for communications using the FX0N-485ADP, peer to peer (N:N) network.

FX and FX2C CPU Versions

- CPU Ver 3.07 the following instructions were added: ASCI (FNC82), CCD (FN84), FLT (FNC49), HEX (FNC83), RS (FNC80), SE (FNC16), SORT (FNC69), SQR (FNC48)

  The following device ranges were upgraded: EI (FNC04), BMOV (FNC15), HSCS (FNC53), PLSY (FNC57), FMOV (FNC16), MEAN (FNC45), ABSD (FNC62), DSW (FNC72), SEGL(74), PR (FNC77).

  The following device ranges were added: input and output devices are independently addressable up to 256 points in software. Total combined input and output points is 256.

  Auxiliary relays increased to 1536 points (M0-M1535)

  Data registers increased to 1000 points (D0-D999)

  Optional RAM File Registers added 2000 points (D6000-D7999)

  Pointers increased to 128 points (P0-P127)
- CPU ver 3.11 The following instructions were added: PID (FNC88)
- CPU Ver 3.2 The following features were added: software control for protocol communications with the FX-485ADP, 1:N network.
- CPU Ver 3.30 The following features were added: software control for protocol communications with the FX-485ADP, 1:N network.

  The following instructions were removed: ANRD (FNC91), ANWR (FNC92), BLK (FNC97), MCDE (FNC98), MNET (FNC90).

6.1.3 Programming equipment

Programming tools operating old system software can not access the new features added to the FX CPU from version 3.07. However, programming certain standard applied instructions in conjunction with specially auxiliary coils (M coils) can achieve the same effective instructions as the new instructions. The following tables identify which version of peripheral software will

work directly with all of the new features and which peripheral software versions require use of modified instructions.

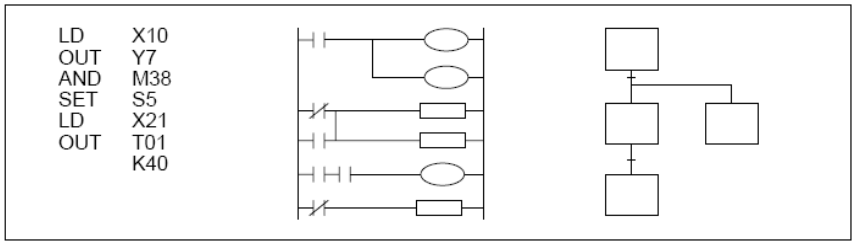
Peripherals Table			
Description	Model Number	System software version which will.....	
		....require the use of auxiliary M coils	....program all instructions directly
Hand held programmer (HHP)	FX-10P-E	V 1.10	from V 2.00
HHP cassette	FX-20P-MFXA-E	V 1.20	from V 2.00
Programming software	FX-PCS/AT-E-KIT	V 1.01	from V 2.00
	FX-A6GPP-E-KIT	V 1.00	from V 2.00
Data access units	FX-10DU-E	V 1.10	from V 2.00
	FX-20DU-E	V 1.10	from V 2.00
	Other DU units		from V 1.00

Existing Instruction And Special M Coil Combination To Mimic The Operation Of The Identified Instruction					
Existing FX instruction			used to mimic the operation of.....		
Mnemonic	FNC number	Modifying M coil	Mimicked instruction	Mnemonic	FNC Number
MOV	12	M8190	Square root	SQR	48
MOV	12	M8191	Float	FLT	49
RAMP	67	M8193	Data search	SER	61
RAMP	67	M8194	RS232 instruction	RS	80
FMOV	16	M8196	Hex to ASCII conversion	ASCI	82
FMOV	16	M8197	ASCII to Hex conversion	HEX	83
FMOV	16	M8195	Sum check	CCD	84

6.2 Basic Program Instructions

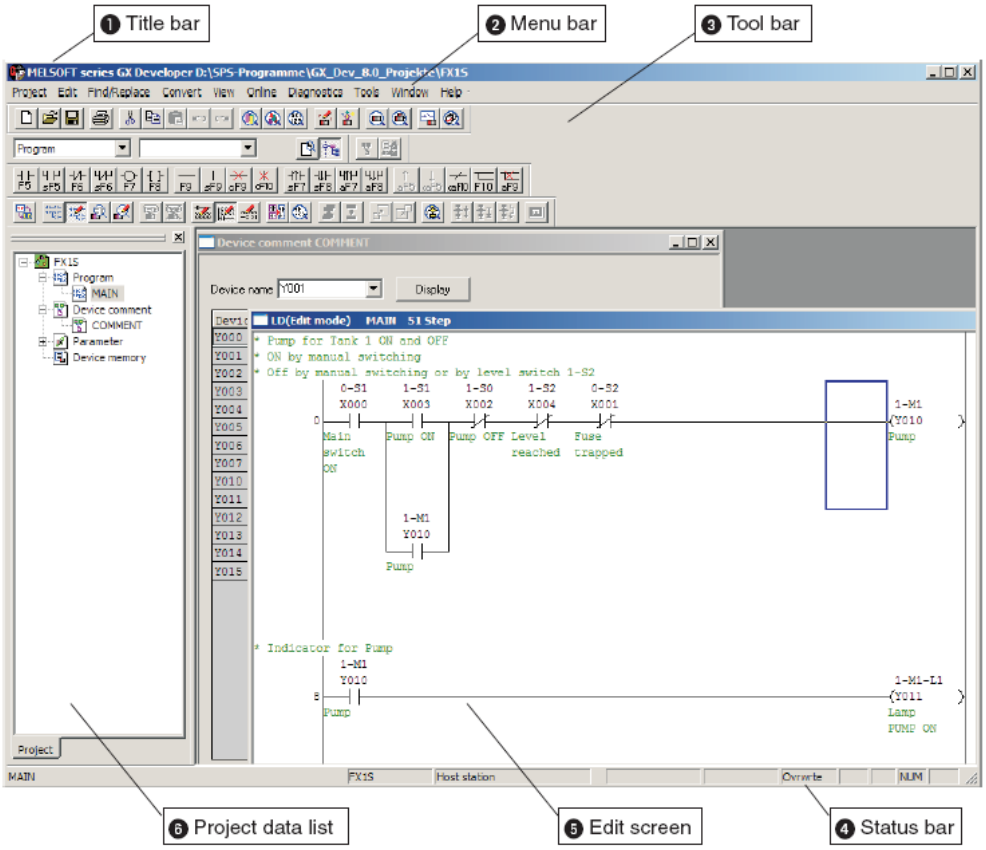
6.2.1 What is a program?

A program is a connected series of instructions written in a language that the PLC can understand. There are three forms of program format, instruction, ladder and SFC?STL. not all programming tools can work in all programming forms. Generally hand held programming panels only work with instruction format while most graphic programming tools will work with both instruction and ladder format. Specialist programming software will also allow SFC style programming.



6.2.2 Start Programming GX Developer

After installing GX developer on your PC you can start the program by selecting its entry Start>Programs>MELSEC Application>GX Developer. This Displays the main program window:



1. Title bar

   The title bar of the GX Developer FX program window shows the path and name of the current project. The usual buttons for minimizing and resizing the program window and exiting the program are located at the right hand end of the title bar.

2. Menu bar

   The menu bar contains the menus that provide access to GX Developer's functions. Clicking on a menu title displays a drop-down menu with a list of options that you can select.

3. Toolbars

   Many of the most frequently used program functions can be accessed directly with the tools (icons) in the toolbars.

4. Status bar

   The status bar displays some useful information, including the current PLC type and editing mode. You can also activate and deactivate the status bar in the View menu.

5. Editing screen

   The editing screen is where you do your programming and documentation work. You can have multiple editing and dialog windows open at the same time.

6. Project data list

   The program, its documentation and the parameters for the FX controller are stored together in a project. The project data list shows the directories in which the components of the current project are stored.

   Program

   Controllers in the MELSEC FX family can only process one program at a time. The default name assigned to this program is MAIN. You can rename MAIN if you want. To do this select the MAIN entry and then right click on it to display the context menu and select Rename.

   Device Comment

   You can assign a comment to every PLC device. These comments can be displayed in the program. You can enter and edit these comments by opening the Device comment file in the project data list.

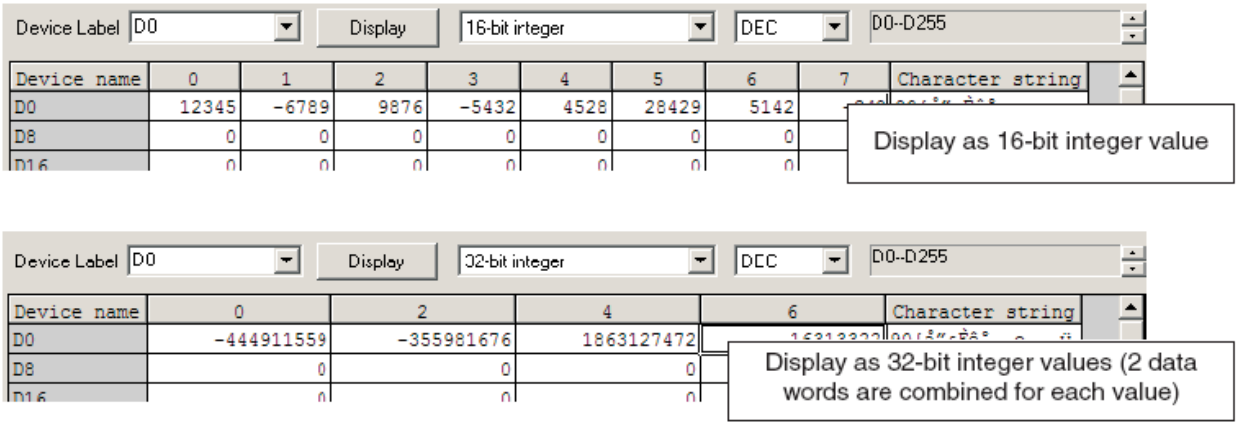
   Parameter

   Double-clicking on PLC parameter in the project data list opens a dialog in which you can enter and adjust all the settings necessary for the operation of the PLC. The PLC parameters are transferred to the CPU together with the program.

Device Memory

The file stored in the Device memory directory can be used to enter default values for each of the CPU's data registers (D) while you are programming. To create a device memory file select Device memory in the project data list and right-click to display the context menu. Then select New…and enter the name of the file you want to create.

To open the file containing the device memory values just double-click on its name in the project data list. You can select between a variety of data display formats and you can also switch between hexadecimal and decimal modes.



6.2.3 Outline of basic devices used in programming

There are six basic programming devices. Each device has its own unique use. To enable quick and easy identification each device is assigned a single reference letter;

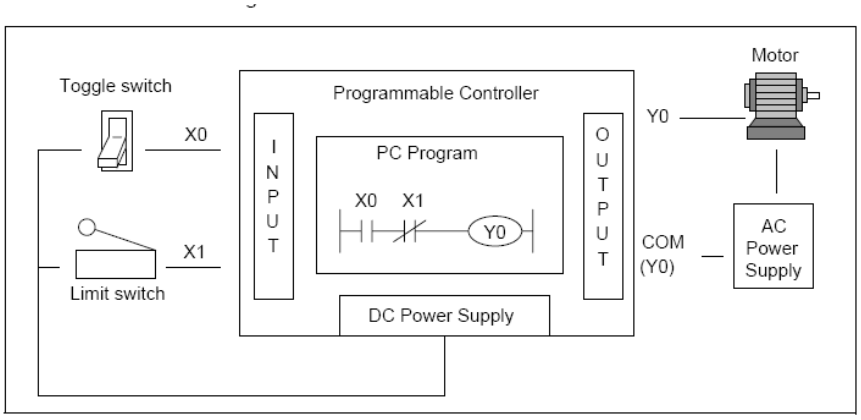
- X: this is used to identify all direct, physical inputs to the PLC.
- Y: this is used to identify all direct, physical outputs to the PLC.
- T: this is used to identify a timing device which is contained within the PLC.
- C: this is used to identify a counting device which is contained within the PLC.
- M and S: these are used as internal operation flags within the PLC.

All of devices mentioned above are known as "bit devices". This is a descriptive tittle telling the user that these devices only have two states; ON or OFF, 1 or 0.

6.2.4 How to read ladder logic

ladder logic is very closely associated to basic relay logic. There are both contacts and coils that can be loaded and driven in different configurations. However, the basic principle remains the same. A coil drives direct outputs of the PLC or drives internal timers, counters or flags. Each coil has associated contacts. These contacts are available in both 'Normally open' (NO) and 'Normally Closed' (NC) configurations.

The term ' norma' refers to the status of the contacts when the coil is not energized. Using a relay analogy, when the coils is OFF, a NO contact would have no current flow, that is, a load being supplied through a NO contact would not operate. However, a NC contact would allow current to flow, hence the connected load would be active.



6.3 STL Programming

STL/SFC Programming, although having been available for many years, is still misunderstood and misrepresented. Mitsubishi tries to correct this oversight as STL/SFC programming becoming as important as ladder style programming

6.3.1 What are STL, SFC and IEC1131?

In recent years Sequential Function Chart (SFC) style programming have become very popular throughout Europe and have prompted the creation of IEC1131 part 3.

The IEC 1131 SFC standard has been designed to become and interchangeable programming language. The idea being that a program written to IEC1131 SFC standards on one manufacturers PLC can be easily converted for use on a second manufacturers PLC.

STL programming is one of the basic programming instructions included in all FX PLC family members. The abbreviation STL actually means Step ladder programming. STL programming is a very simple concept to understand yet can provide the user with one of the most powerful programming technique possible. The key to STL lies in its ability to allow the programmer to create an operational program which flows and works in almost exactly the same manner as SFC.

One of the key differences to Mitsubishi's STL programming system is that it can be entered into a PLC in 3 formats. These are:

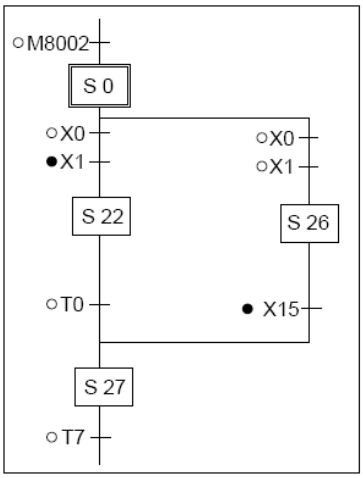
1. Instructions – a word/mnemonic entry system
2. Ladder - a graphical program construction method using a relay logic symbols
3. SFC - a flow chart style of STL program entry (similar to SFC).

6.3.2 How STL operates

STL is a system that allows user to write a program which functions in much the same way as a flow chart, this can be seen in the diagram opposite. STL derives its strength by organizing a larger program into smaller more manageable parts. Each of these parts can be transferred to as either a state or a step. To help identify the states, each is given a unique identification number.

Each step is a program

Each state is completely isolated from all other states within the whole program. A good way to envisage this is that each state is a separate program and the user puts each of those program together in the order that they require to perform their task. This means that states can be reused many times and in different orders. This saves on programming time AND cuts down on the number of programming errors encountered.



Combined SFC Ladder representation

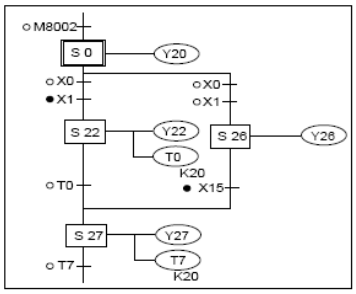
Sometimes STL programs will be written in hardcopy as a combination of both flow diagram and internal sub-program.

Please note the following convention is used:

O Normally Open contact

- Normally Closed contact

Common alternatives are ‘a’ and ‘b’ identifiers for Normally Open, normally Closed dtates or often a line drawn over the top of the Normally Closed contact name is used, e.g X000.

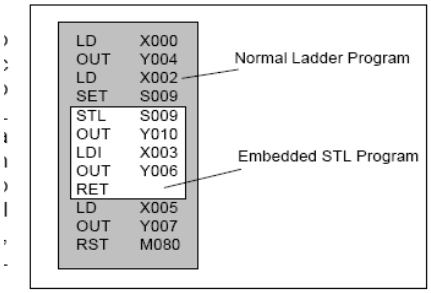


6.3.3 How to start and end an STL program

Before any complex programming can be used undertaken the basics of how to start and more importantly how to finish an STL program need to be examined.

Embedded STL programs

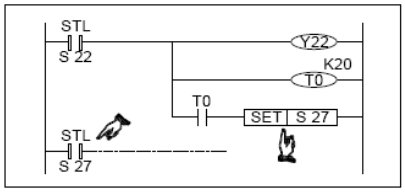
An STL style program does not have to entirely replace a standard ladder logic program. In fact it might be very difficult to do so. Instead small or even large section of STL program can be entered at any point in a program. Once the STL task has been completed the program must go back to processing standard program instructions until the next STL program block.



Activating new states

Once an STL step has been selected, how is it used and how is the program driven? This is not so difficult, if it is considered that for an STL step to be active its associated state coil must be ON. Hence, to start an STL sequence all that has to be done is to drive the relevant state ON.

There are many different methods to drive a state, for example, the initial state coils could be pulsed, SET or just included in an OUT instruction. However, within Mitsubishi's STL programming language an STL coil which is SET has a different meaning than one that is included in an OUT instruction.



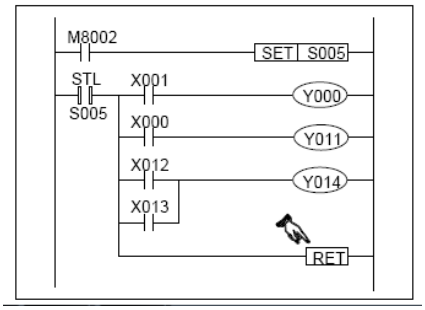
Terminating an STL Program

Once an STL program has been started the programmable controllers CPU will process all following instructions as being part of that STL program. This means that when a second program scan is started the normal instructions at the beginning of the program are considered to

be within the STL program. This is obviously incorrect and the CPU will proceed to identify a programming error and disable the programmable controllers operation.

Returning to Standard Ladder

This is achieved by placing a ERT or RETurn instruction as the last STL step of an STL program block. This instruction then returns programming control to the ladder sequence.



Chapter 7

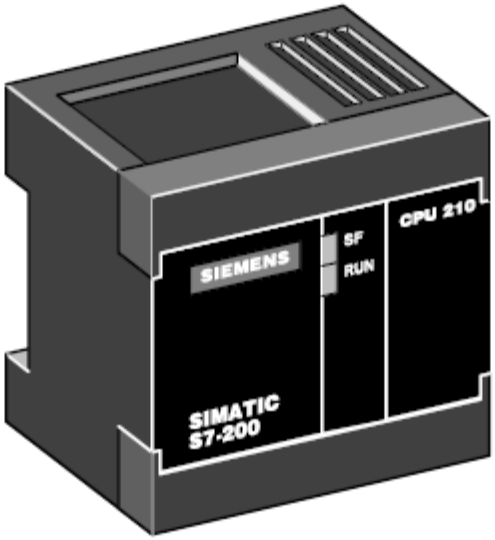
Siemens PLC Programming

7.1 Installing the S7-200 CPU 210

7.1.1 Installing a CPU 210

The S7-200 CPU 210 is one of the S7-200 series of micro-programmable logic controllers (Micro PLCs) that can control a variety of automation applications. Figure below shows an S7-200 CPU 210. The compact design and low cost of the CPU 210 make a perfect solution for controlling small applications. In addition, the variety of input and output voltages provides you with the flexibility you need to solve your automation problems with the maintenance-free operation of the CPU 210.

The CPU 210 is easy to install. You can use the mounting holes to attach the module to a panel, or you can use the built-in DIN clips to mount the module onto a DIN rail. The small size of the CPU 210 allows you to make efficient use of space.



Product Overview

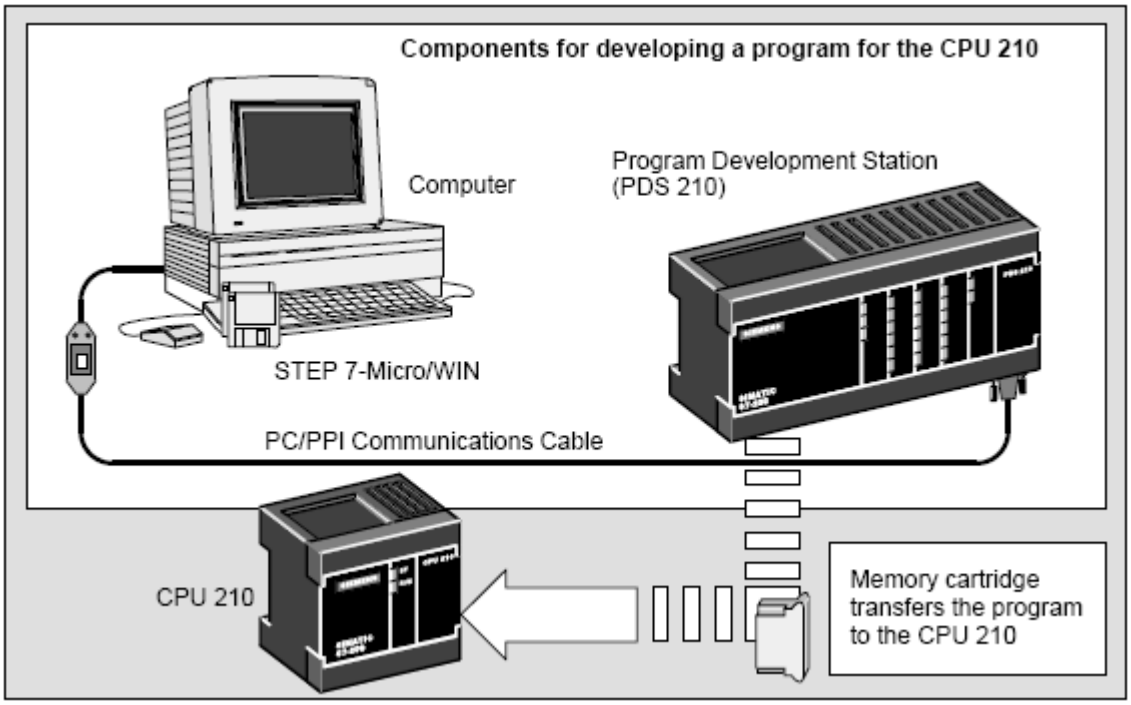
The CPU 210 combines a central processing unit (CPU), power supply, and discrete I/O points into a compact, stand-alone device.

- The CPU executes the program and stores the data for controlling the automation task or process.
- The inputs and outputs are the system control points: the inputs monitor the signals from the field devices (such as sensors and switches), and the outputs control pumps, motors, or other devices in your process.
- Status lights provide visual information about the CPU mode (RUN) or whether a system fault (SF) has been detected.

Equipment Requirements

You use the STEP 7-Micro/WIN programming software with a program development station (the PDS 210) to create and to test your program. The final program is then loaded onto a memory cartridge, which is then installed in the CPU 210. You need the following equipment to create programs for the CPU 210:

- Personal computer (PC) running the STEP 7-Micro/WIN programming software.
- Program development station (PDS 210).
- PC/PPI communications cable.
- Memory cartridge for transferring the program to the CPU 210.



7.1.2 Installing the STEP 7-Micro/WIN Version 2.0 Software

Installing and Using the STEP 7-Micro/WINVersion 2.0 Software

This manual describes Version 2.0 of STEP 7-Micro/WIN. Previous versions of the software may operate differently. STEP 7-Micro/WIN is a Windows-based software application used for programming the S7-200 Micro PLC (programmable logic controller). The STEP 7-Micro/WIN programming software package provides a set of tools required to program the S7-210 in either statement list (STL) or ladder logic (LAD) programming language. In order to use STEP 7-Micro/WIN, you must have the following equipment:

- Recommended: a personal computer (PC) with an 80486 or greater processor and 8 Mbyte of RAM or a Siemens programming device (such as a PG 740); minimum computer requirement: 80386 with 8 Mbyte of RAM
- A PC/PPI cable connected to your communications port (COM)
- A program development station (PDS 210)
- VGA monitor, or any monitor supported by Microsoft Windows

- At least 35 Mbyte of free hard disk space (recommended)
- Microsoft Windows 3.1, Windows for Workgroups 3.11, Windows 95, or Windows NT 3.51 or greater
- Optional but recommended: any mouse supported by Microsoft Windows STEP 7-Micro/WIN provides extensive online help. Use the **Help** menu command or press **F1** to obtain the most current information.

Installing the STEP 7-Micro/WIN Version 2.0 Software Pre-installation Instructions

Before running the setup procedure, do the following:

- If a previous version of STEP 7-Micro/WIN is installed, back up all application programs to diskette.
- Make sure all applications are closed, including the Microsoft Office toolbar. Installation may require that you restart your computer.

Installation Instructions for Windows 3.1

If you have Windows 3.1 (Windows for Workgroups 3.11 or Windows NT) on your machine, use the following procedure to install the STEP 7-Micro/WIN software:

- Start by inserting Disk 1 in the disk drive of your computer (usually designated drive A: or drive B:).
- From the Program Manager, select the menu command **File _ Run...**
- In the Run dialog box, type **a:\setup** and click on the "OK" button. This starts the setup procedure.
- Follow the online setup procedure to complete the installation.

Installation Instructions for Windows 95

If you have Windows 95 on your machine, you can use the following procedure to install the STEP 7-Micro/WIN software:

1. Start by inserting Disk 1 in the disk drive of your computer (usually designated drive A: or drive B:).
2. Click once on the **Start** button to open the Windows 95 menu.
3. Click on the **Run...** menu item.
4. In the Run dialog box, type **a:\setup** and click on the "OK" button. This starts the setup procedure.
5. Follow the online setup procedure to complete the installation.

Troubleshooting the Installation

The following situations can cause the installation to fail:

- Not enough memory: you need to have at least 35 Mbyte of free space on your hard disk.
- Bad diskette: verify that the diskette is bad, and then call your salesman or distributor.
- Operator error: start over and read the instructions carefully.
- Failure to close any open applications, including the Microsoft Office toolbar.

7.1.3 Creating a Program

STEP 7-Micro/WIN allows you to create the user program (OB1) with either the Ladder Editor or the Statement List Editor.

Entering Your Program in Ladder

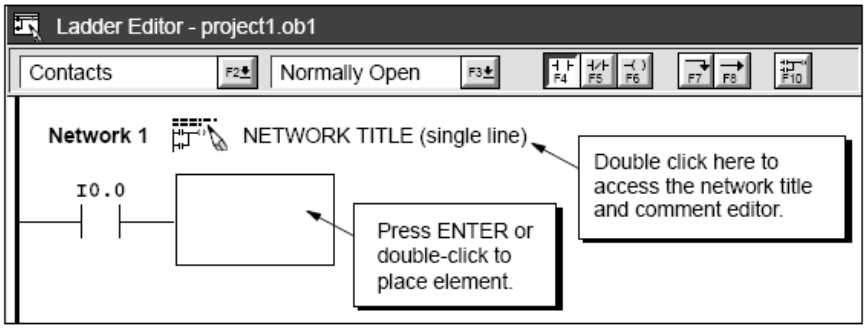
The Ladder Editor window allows you to write a program using graphical symbols. See Figure 2-5. The toolbar includes some of the more common ladder elements used to enter your program. The first (left) drop-down list box contains instruction categories. You can access these categories by clicking or pressing F2. After a category is selected, the second drop-down list contains the instructions specific to that category. To display a list of all instructions in alphabetic order, press F9 or select the All Instructions category. Each network allows two types of comments:

- Single-line network title comments are always visible in the ladder display. You can access the network editor by double-clicking anywhere in the network title region.
- Multi-line network comments are only visible through a dialog box, but can be printed (if that option has been selected through the Page Setup dialog). You can access the network comment editor by double-clicking anywhere in the network title region.

To start entering your program, follow these steps:

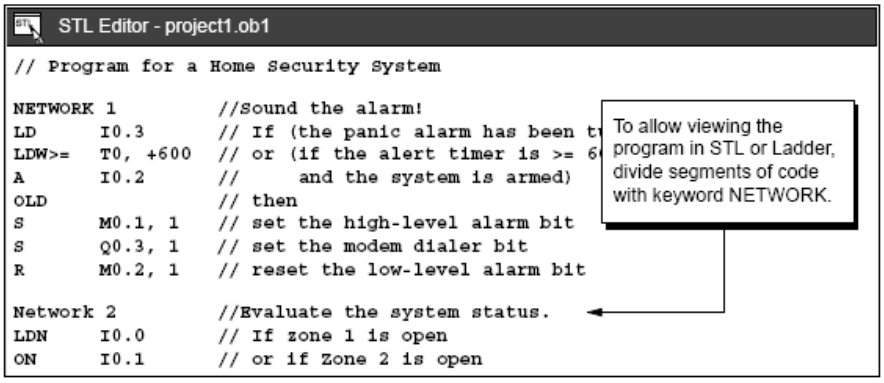
1. To enter a program title, select the menu command **Edit _ Program Title**.
2. To enter ladder elements, select the type of element you want by clicking the corresponding icon button or selecting from the instruction list.
3. Type the address or parameter in each text field and press ENTER.

To change or replace one of the elements, move the cursor to that element and select the new element. You can also cut, copy, or paste elements at the cursor location.



Entering Your Program in Statement List

The Statement List (STL) Editor is a free-form text editor which allows a certain degree of flexibility in the way you choose to enter program instructions. Figure 2-6 shows an example of a statement list program. You can cut, copy, and paste in the STL Editor. STEP 7-Micro/WIN also includes search-and-replace functions.



To enter an STL program, follow these guidelines:

- Start each comment with a double slash (//). Each additional comment line must also begin with a double slash.
- End each line with a carriage return.
- Separate each instruction from its address or parameter with a space or tab.
- Do not use a space between the operand type and the address (for example, enter **I0.0**, not **I 0.0**).
- Separate each operand within an instruction with a comma, space, or tab.
- Use quotation marks when entering symbol names. For example, if your symbol table contains the symbol name Start1 for the address **I0.0**, enter the instruction as follows:

  LD "Start1"

  To be able to view an STL program in ladder, you must divide segments of code into separate networks by entering the keyword NETWORK. (Network numbers are generated automatically after you compile or upload the program.)

Compiling the Program

After completing a network or series of networks, you can check the syntax of your code by selecting the menu command **CPU _ Compile** or by clicking the Compile button:

Installing and Using the STEP 7-Micro/WIN Version 2.0 Software

Viewing a Program in Ladder or Statement List

You can view a program in either ladder or STL by selecting the menu command **View _ STL** or **View _ Ladder**. When you change the view from STL to ladder and back again to STL, you may notice changes in the presentation of the STL program, such as:

- Instructions and addresses are changed from lower case to upper case.
- Spaces between instructions and addresses are replaced with tabs. You can accomplish the same formatting of the STL instructions by selecting the menu command **CPU _ Compile** while the STL Editor is active.

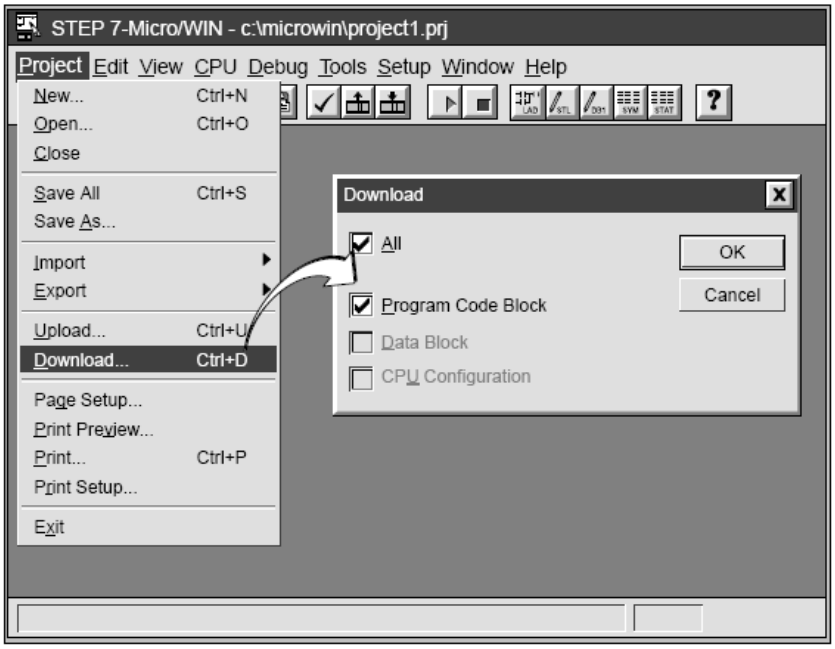
Downloading A Program

After developing and testing your program on the PDS 210, you must transfer the program to the CPU 210 using the memory cartridge. In the same manner as you could use a diskette to transfer files from one computer to another, you use a memory cartridge to transfer your program from the PDS 210 to the CPU 210.

Downloading the Program to the PDS 210

After completing your program, you can download the project to the PDS 210. To download your program, select the menu command **Project _ Download...** or click the Download button in the main window.

The Download dialog box that appears allows you to specify the project components you want to download. Select only “Program Code Block” for the PDS 210: the data block and the CPU configuration are not used by the CPU 210. Click on the “OK” button to confirm your choices and to execute the download operation.

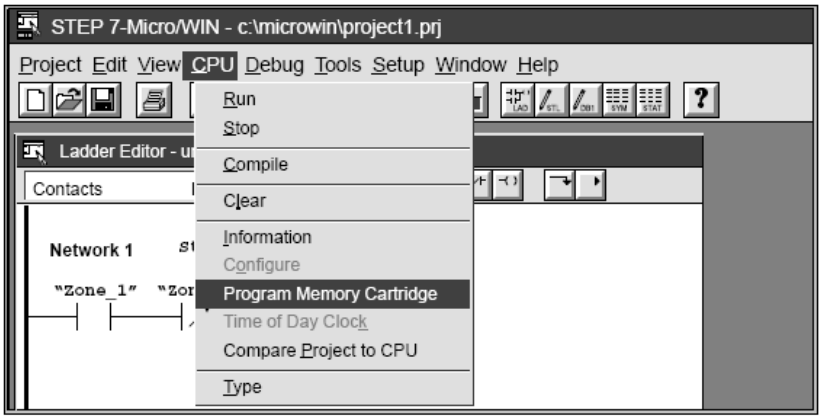


Copying Your Program to the Memory Cartridge

You can copy your program to the memory cartridge only when the PDS 210 is powered up and the memory cartridge is installed. (You can install or remove the memory cartridge while the PDS 210 is powered up.)

To install the memory cartridge, remove the protective tape from the memory cartridge receptacle and insert the memory cartridge into the receptacle located under an access cover of the PDS 210. (The memory cartridge is keyed for proper installation.) After the memory cartridge is installed, use the following procedure to copy the program:

1. If the program has not already been downloaded to the PDS 210, use the menu command **Project _ Download...** to download the program.
2. Use the menu command **CPU _ Program Memory Cartridge** to copy the program to the memory cartridge.
3. Remove the memory cartridge from the PDS 210.



Transferring the Program to the CPU 210

To transfer the program from the memory cartridge to the CPU 210, follow these steps:

1. Turn off the power to the CPU 210.
2. Insert the memory cartridge in the CPU 210. (The memory cartridge is keyed for proper installation.)
3. Turn on the power to the CPU 210.
4. After the RUN LED turns on, remove the memory cartridge from the CPU 210.

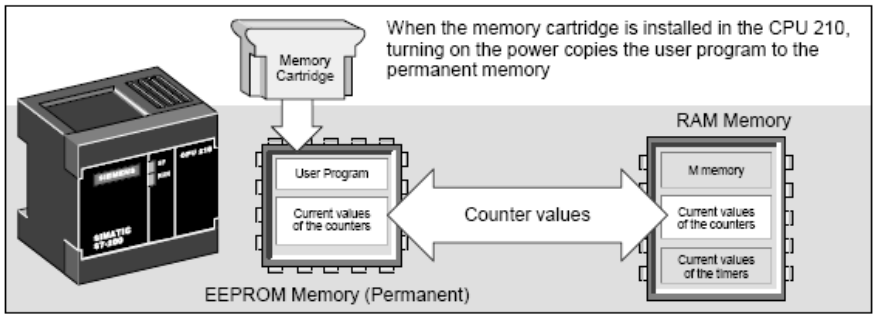
As shown in Figure 2-10, the CPU 210 performs the following tasks after you turn the power on when a memory cartridge is installed in the CPU 210:

- The M, T, and Q areas of memory are cleared.
- The current values for the counters (which are stored in the permanent memory) are cleared. (The current values for the counters are erased only when the memory cartridge is installed in the CPU 210. If a memory cartridge is not installed, the current values are retained.)
- The user program is copied from the memory cartridge to the permanent EEPROM memory.

Always remove the memory cartridge from the CPU 210 after the program has been installed. When a valid program is installed, the CPU 210 automatically goes

to RUN mode when power is applied. As your program runs, the CPU 210 updates the values stored in the RAM memory (the values stored in M memory, the current values for the four counters, and the current values for the four timers).

When you turn the power off, the CPU 210 saves the current values of the four counters to the permanent EEPROM memory. The other values stored in RAM (such as M memory, current values for the timers, and the copy of the user program) are cleared. Unless a memory cartridge is installed in the CPU 210, the current values for the counters are retentive. The current values for the counters are automatically restored to the RAM memory when you turn power on for the CPU 210 (with no memory cartridge installed).



Chapter 8

GE Fanuc Series 90 Micro PLC

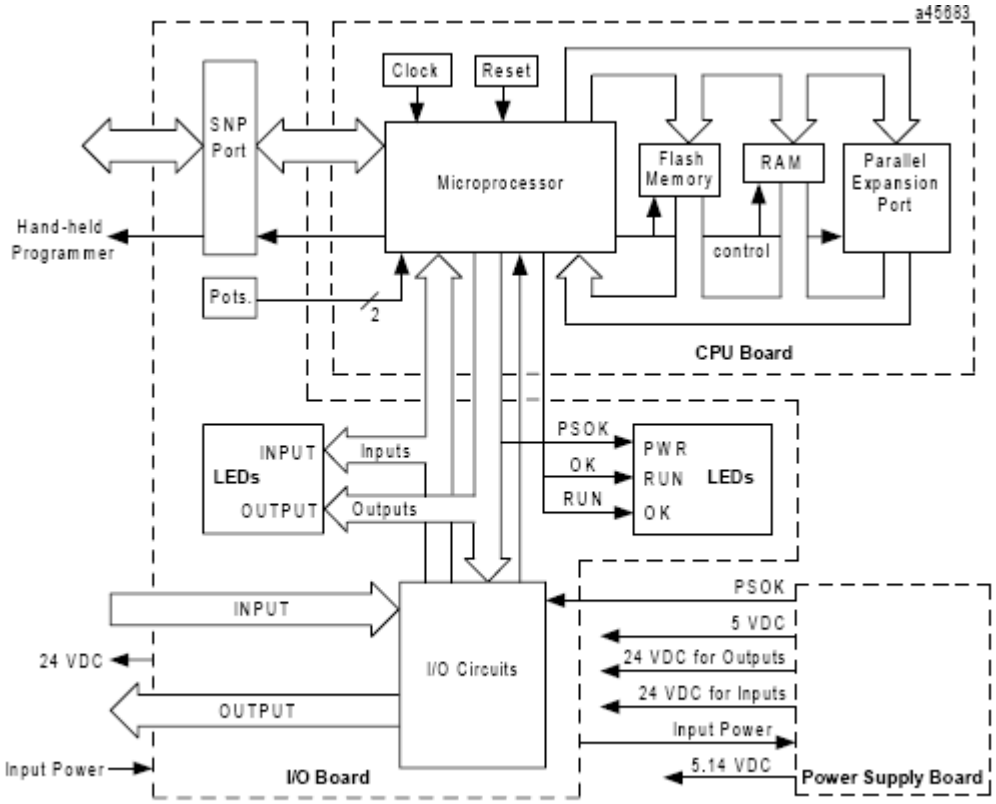
Series 90 Micro PLCs offer an array of useful features, including:

- Compatibility with Logicmaster 90-30/20/Micro programming software
- Support for the 90-30 Hand-Held Programmer (HHP)
- An alarm processor function
- Password protection to limit access to PLC contents
- A built-in High Speed Counter (HSC) function that can be configured as four type A counters or as one type B counter and one type A counter (DC in/relay out Micro PLCs only)
- Two potentiometers that provide selectable analog inputs to %AI16 and %AI17 (with configurable filtering)
- Configurable software filtering of discrete inputs
- Series 90 (SNP) and SNP Extended (SNPX), and RTU slave communication protocols
- A *pulse catch input* function, selectable on up to four inputs, that detects pulses at least 100 microseconds in width
- Pulse train and Pulse Width Modulation (PWM) outputs (Micro PLCs with DC output only)
- Compatibility with 14-point expansion unit (23 and 28-point Micro PLCs)
- Pager Enunciation function that can be configured to send a specified byte string from Serial Port 2 (23 and 28-point Micro PLCs)
- Two analog inputs and one analog output (23-point Micro PLC)

8.1 Functional Description

The Micro PLC contains a CPU circuit board, an I/O board, and a Power Supply board. Figure of Micro PLC Functional Block Diagram as below, provides an overview of Micro PLC inputs and outputs and of the functions performed by each circuit board.

Figure of Micro PLC Functional Block Diagram



CPU Board

The CPU contains and executes the user program and communicates with the programmer (HHP or computer running Logicmaster 90-30/90-20/Micro software). The primary capabilities of the Micro PLC CPU hardware are listed in Table CPU Capabilities as below:

Table CPU Capabilities

14-Point Micro PLCs	23 and 28-Point Micro PLCs
H8/3003 microprocessor running at 9.84Mhz Powerup reset circuit Interrupt for power fail warning (2.0 ms) Internal Coils - 1024 Four configurable 5Khz HSCs	
512K x 8 sectored flash memory for operating system and nonvolatile user program storage (3K words of user flash memory)	256K x 16 sectored flash memory for operating system and nonvolatile user program storage (6K words of user flash memory)
32 Kbyte RAM backed by super cap (provides data retention for 3–4 days with the power off at 25°C)	64 Kbyte RAM backed by lithium battery Real time clock backed up by lithium battery
Maximum User Program - 3K words	Maximum User Program – 6K words
Registers - 256 words	Registers – 2K words
Typical Scan Rate: 1.8 ms/K of logic (Boolean contacts)	Typical Scan Rate: 1.0 ms/K of logic (Boolean contacts)
An RS-422 serial port that supports SNP, SNPX and RTU Slave protocols	Two RS-422 serial ports: Port 1 supports SNP/SNPX slave protocols; Port 2 supports SNP/SNPX Slave and Master protocols and RTU Slave protocol. (Port 2 does not support the HHP.)
	Ability to support up to four expansion units

High Speed Counters (IC693UDR011/002/005, IC693UAL006, IC693UDR010)

The high speed counter (HSC) function consists of four built-in counters. Each counter provides direct processing of rapid pulse signals up to 5Khz for industrial control applications such as: meter proving, turbine flow meter, velocity measurement, material handling, motion control, and process control. Because it uses direct processing, the HSC can sense inputs, count, and respond with outputs without needing to communicate with the CPU.

The HSC function can be configured to operate in one of two modes:

A4 – four identical, independent, simple (type A) counters that can count up or down

B1–3, A4 – counters 1–3 configured as one type B counter; counter 4 as one type A counter.

In either mode, each counter can be enabled independently. Type A counters can be configured for up or down counting (default is up) and for positive or negative edge detection (default is positive). The HSC function is configured using the Series 90-30 and 90-20 Hand-Held

Programmer or the Logicmaster 90-30/20/Micro software configurator function. Many features can also be configured from an application program using the COMM_REQ function block.

Type A Counters

A type A counter accepts a count input that increments a 16 bit accumulator. It also accepts a preload/strobe input that can either preload the counter accumulator with a user-defined value (PRELOAD mode) or strobe the accumulator (STROBE mode) into a 16-bit register. The four type A counters provide 15 words of %AI data or 16 bits of %I data to the PLC. They receive 16 bits of %Q data from the PLC. Each counter has two discrete inputs and one discrete output.

Type B Counter

The type B counter provides an AQUADB counting function. An AQUADB input consists of two signals (designated *A* and *B*). A count occurs for each transition of *either A or B*. The counter uses the phase relationship between A and B to determine count direction.

DC Output (IC693UDR005/010, UAL006)

The high-speed DC output (%Q1) can be configured for PWM, pulse train, or HSC output. Counter channel 1 can be configured for only one of these outputs at a time. Because AQUADB counting uses channels 1–3, the PWM and pulse train outputs are not available when a type B counter is configured.

PWM Output

The frequency of the PWM output (19Hz to 2Khz) is selected by writing a value to memory location %AQ2. A PWM duty ratio (the amount of time that the signal is active compared to the signal period) within the range of 0 to 100% can be selected by writing a value to memory location %AQ3.

Pulse Output

The frequency (10hz to 2Khz) of the pulse train is selected by writing a value to memory location %AQ123. The number of pulses to be output (0 to 32767) is selected by writing a value to memory location %AQ124.

ASCII Output (IC693UDR005/010, UAL006)

This feature allows you to send a specified byte string out the serial port by including a COMM_REQ (Communications Request) instruction in a ladder diagram. The Micro PLC can automatically send a message to a remote location that has the ability to display an ASCII string, such as a pager. As an example of how pager enunciation could be used, when a specific alarm condition is detected by the PLC, the PLC would execute a COMM_REQ instruction to autodial the modem attached to the serial port. If the autodial COMM_REQ is successful, a second COMM_REQ would be executed to send an informative ASCII string to the pager where it can be viewed by the user. Finally, a third COMM_REQ would be sent to hang up the pager.

I/O Board

The I/O board provides the interface to the front panel input, output, and power supply connections for the Micro PLC.

Input Circuits

DC Input Circuits (IC693UDR001/002/005/010, UAL006)

The DC input circuits condition and filter 24 VDC input voltages so that they can be properly detected by the CPU module. The input points can be used in either positive or negative logic mode. The DC inputs can be used as regular inputs or to supply count and preload/strobe inputs for HSCs.

AC Input Circuits (IC693UAA003/007)

The AC input circuits accept 120 VAC, 50/60 Hz signals. Input characteristics are compatible with a wide range of user-supplied input devices, such as pushbuttons, limit switches, and electronic proximity switches.

Potentiometer Inputs (All Models)

Two potentiometers are provided to allow adjustment of the values in analog registers %AI16 and %AI17. The potentiometers can be turned by inserting a small screwdriver through an access hole in the Micro PLC front panel. A potential use for the potentiometers would be to set threshold values for use in logical relationships with other inputs/outputs.

Output Circuits

Relay Output Circuits (IC693UDR001/002/005/010, UEX011, UAL006)

The 2-amp, isolated, normally open output circuits allow the low-level signals from the CPU module to control relay devices. There is no fusing on relay outputs. The user should provide external fusing to protect the outputs. The outputs can be configured as regular outputs or as outputs controlled by the HSCs.

AC Output Circuits (IC693UAA003/007)

The AC output points provide 120/240 VAC, 50/60 Hz, 0.5 A signals.

DC Output (IC693UDR005/010, IC693UAL006)

The DC output circuit provides a 24 VDC output voltage. This output can be used as a normal DC output, HSC-controlled output, pulse train output, or pulse width modulation (PWM) output.

Analog I/O (IC693UAL006)

The 23-point Micro PLC features two analog input channels that map to %AI0018 and %AI0019 in the PLC. In voltage mode, the analog-to-digital (A/D) range of 0-32,000 counts corresponds to a 0-10 V input signal. In 0-20mA current mode, the A/D range of 0-32,000 counts corresponds to a 0-20mA input signal. In 4-20mA current mode, the A/D range of 0-32,000 counts corresponds to a 4-20mA input signal.

The analog output channel maps to %AQ0012. In voltage mode, the output channel digital to analog (D/A) range of 0 to 32,000 counts corresponds to a 0-10V output. In 0-20mA current

mode, a range of 0 to 32,000 counts corresponds to a 0-20mA output signal. In 4-20mA current mode, the A/D range of 0-32,000 counts corresponds to a 4-20mA output signal.

8.2 Configuration and Programming

The Micro PLC can be configured and programmed using any of the following methods.

- Logicmaster 90-30/20/Micro software on one of the following types of computers:

a. Workmaster™ II or a CIMSTAR™ I industrial computer

b. IBM® PC-AT, PS/2® (Personal System 2®) with 2 Mbyte RAM and an Intel 386 or higher processor

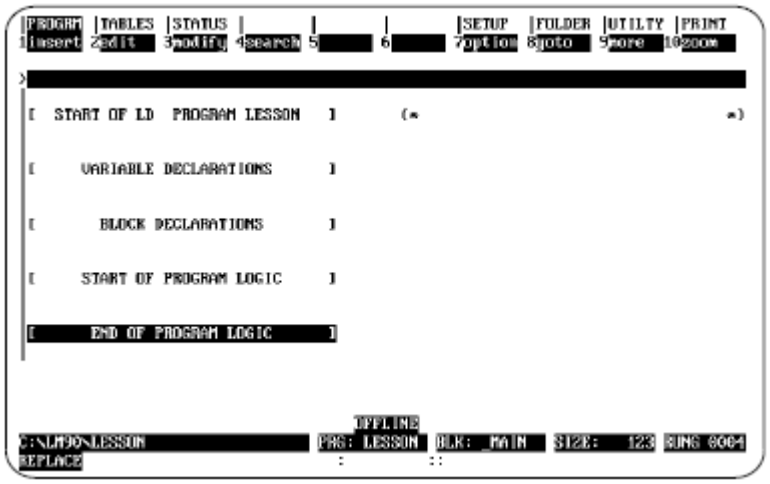
c. MS-DOS compatible Personal Computer with 2 Mbyte RAM and an Intel 386 or higher processor

  - Logicmaster 90 Micro software with any of the above computers.
  - Series 90-30/90-20 Hand-Held Programmer (IC693PRG300).

Configuration and programming can be accomplished off-line from the PLC using the Logicmaster 90 programmer. If you are using an HHP, configuration and programming can be done on-line with the HHP attached to and interfacing with the PLC. Programming and configuration communications must use Port 1. The Micro PLC provides flash memory for non-volatile user program storage and for system firmware. The user program is always executed from flash memory. However, the Micro PLC can be configured to read its configuration at power up from either RAM or flash memory (ROM).

Program Format

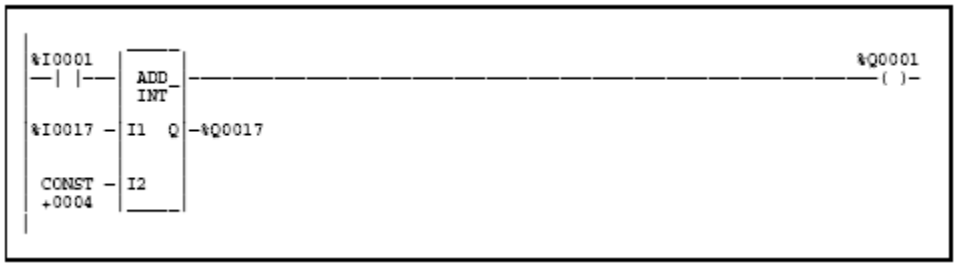
Program elements are combined to form rungs of ladder logic. A ladder diagram has a symbolic power source. Power is considered to flow from the left rail through a contact to the coil or function block connected to the right. From the main menu, select **Program Display/Edit (F1)**. The screen displays a list of markers which represent parts of a program.



Marker	Description
Variable Declarations	To access the variable declaration table, move the cursor to this marker and press **Zoom** (**F10**). Nicknames and reference descriptions can then be entered in the table.
Block Declarations	A program can include more than one block of logic. Additional blocks, known as subroutine blocks, can be called from other blocks. When that is done, blocks must be declared before they are called. The main block has a block declaration table. This table lists all blocks which are part of the complete program. Blocks do not have block declaration tables. However, blocks can be called from the main block or from any block in the program.
Start/End of Program Logic	All logic is placed between these two markers. To enter logic, place the cursor on the **[END OF PROGRAM LOGIC]** marker and press **Insert** (**F1**).

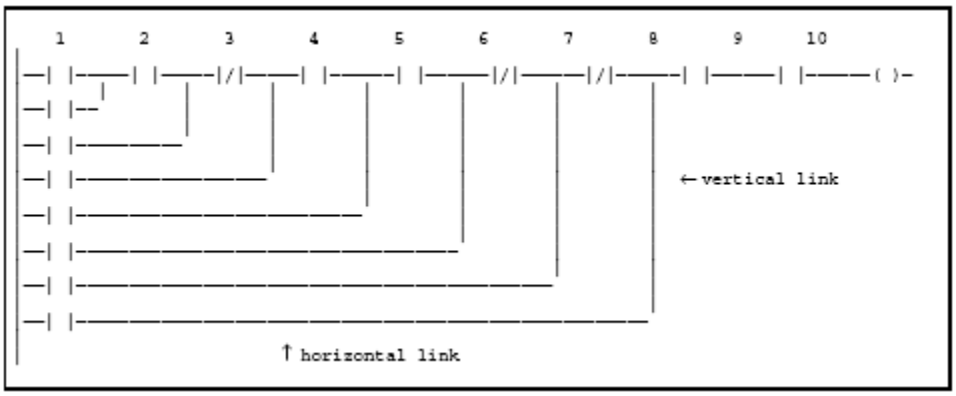
Creating or Editing Program Logic

Program logic consists of various elements such as relays, timers, math functions, and other functions, placed together to form rungs of logic.

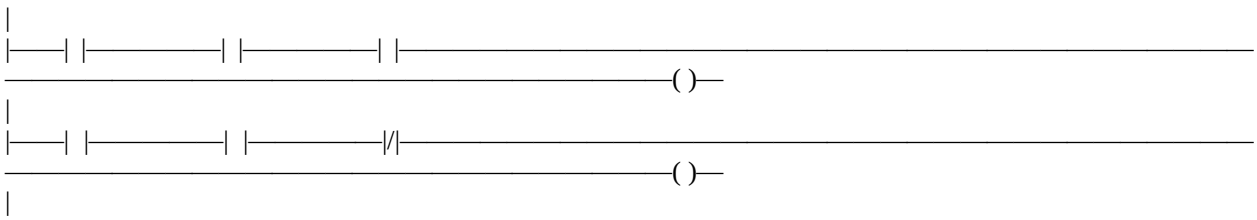


Structure of a Ladder Logic Rung

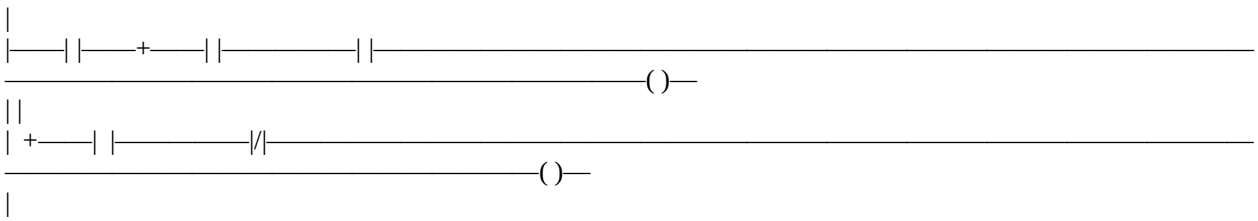
The programmer allows great flexibility in entering program elements; however, it will not allow you to enter a rung with incorrect format or syntax. Each rung may contain up to eight parallel lines; each line may have up to ten elements connected in series. Examples of an element include a normally open contact, a normally closed contact, or a coil. Horizontal and vertical links are used to carry power around an element, or to place elements in parallel or series with one another.



The following example shows two separate rungs, which must be entered and accepted separately.



In the next example, two rung lines are connected by a vertical link, forming only one rung.



The last element of a group of rung elements in series must be a coil, a jump, or a function. Nothing may be to the right of a coil or a jump. The tenth position of a rung line is reserved for

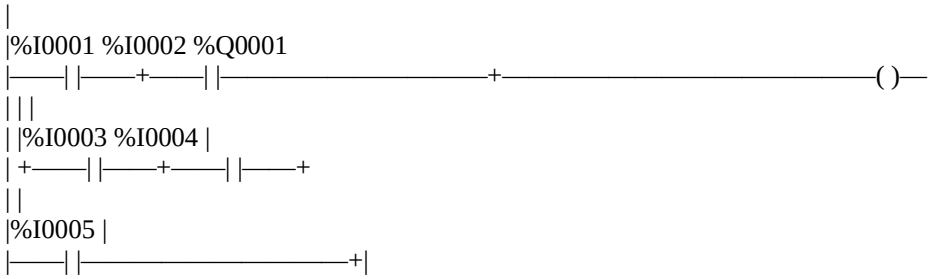
coils and jumps. A call instruction may occupy columns 9 and 10. A rung may contain up to eight coils. A rung line is not required to have elements in each column.

Ladder Logic Language Rules

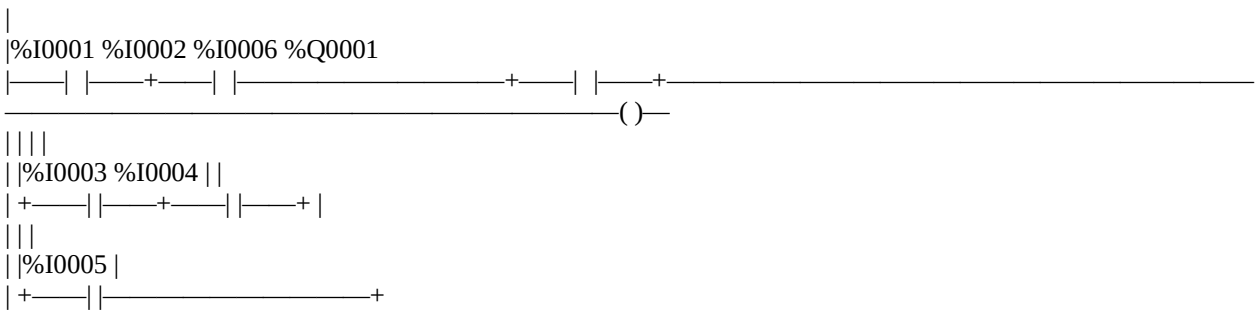
These guidelines should be followed when creating or editing ladder logic:

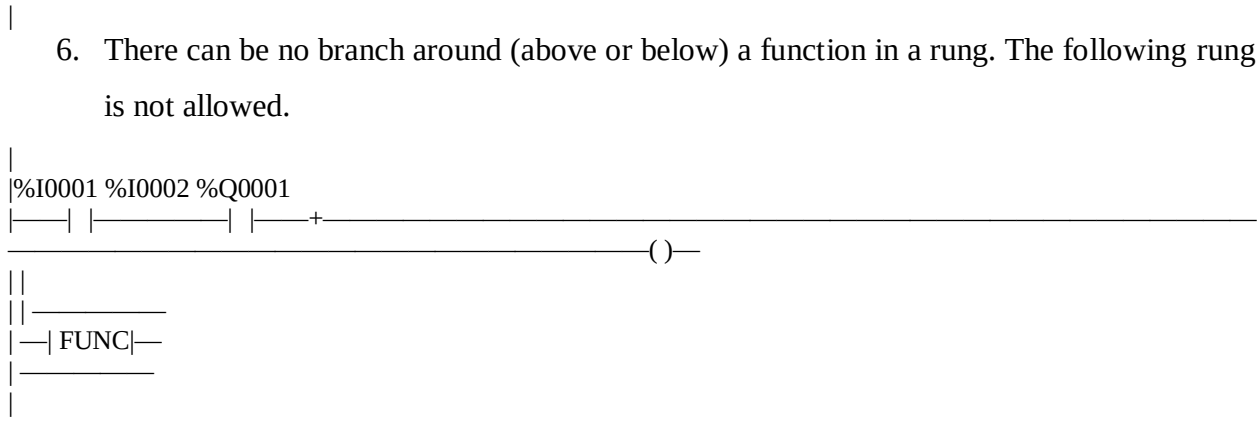
1. If a rung has a transitional coil, it must be the only coil in the rung.
2. There can be only one JUMP or MCR per rung. It must be the last instruction in the rung, and there cannot be a coil in the same rung.
3. A rung must contain at least one contact before any coil, jump, MCR, function, or vertical link. Contacts ***must*** be entered and cannot be left blank. Function blocks cannot be tied directly into the power rail.
4. Short circuits are not allowed.
5. A rung must be composed of properly nested sub-expressions. There can be no branches either into or out of another branch. The following examples contain improperly nested rungs.

A. In this example, the rung line containing the %I0005 contact branches into the middle of the sub-expression (%I0002 OR (%I0003 AND %I0004)).



B. In this example, the rung line containing the %I0005 contact branches out of the middle of the sub-expression (%I0002 OR (%I0003 AND %I0004)).

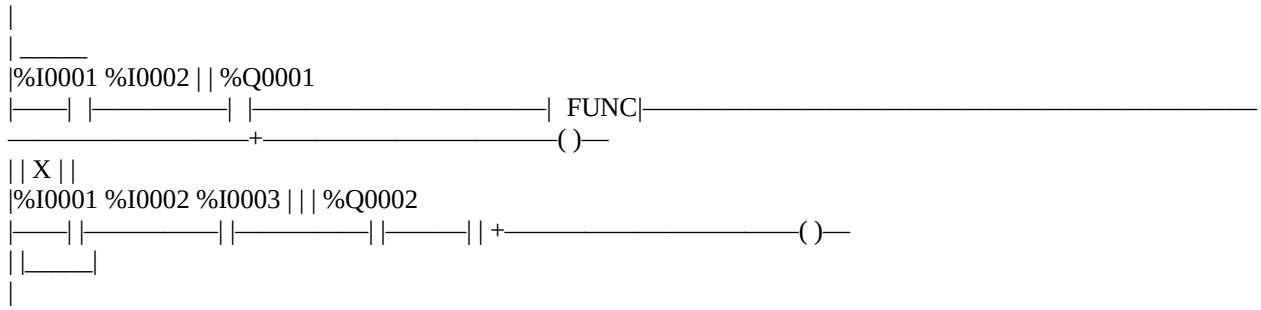




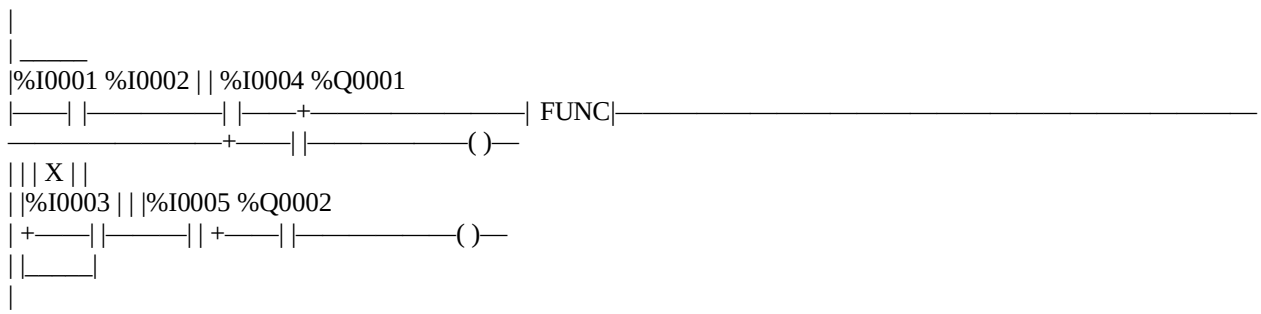
6. There can be no branch around (above or below) a function in a rung. The following rung is not allowed.

7. There can be no sub-paths starting from a vertical in a rung containing a function, except for sub-paths leading directly to coils.

   A. The following rung is allowed because the first sub-path comes directly off the power rail and the second leads directly to coils.



   B. The next rung is not allowed. It has a sub-path starting from a vertical and leading into the function. It also has a sub-path that does not lead directly to coils; it goes through contacts first.



8. There can be no contacts following a function in a rung. Note that the rung in the last example above fails this rule, too.

9. In general, execution order of rung elements is left-to-right. Within a group of parallel branches, the first (lowest rung line) parallel branch is executed first. The first of multiple sub-paths is executed first.

8.3 Fault Reporting

The Micro PLC monitors internal operations for system and user problems. These faults are reported through the %S references and through an internal fault table. Access to %S information is available through the Logicmaster 90 software or the HHP. The fault table can only be accessed by Logicmaster 90 software.

8.4 Specifications

The following tables list ordering information, physical and functional characteristics, and input power requirements for the Micro PLCs.

I/O Point Configurations

Description	Input Points (points/common)*	Output Points (points/common)*	Catalog Numbers
14 point DC in/relay out, AC power	8 DC (4 and 4)	6 relay (1, 1, and 4)	IC693UDR001
14 point DC in/relay out, DC power	8 DC (4 and 4)	6 relay (1, 1, and 4)	IC693UDR002
14 point DC in/DC out, DC power	8 DC (4 and 4)	6 DC (6)	IC693UDD004
14 point AC in/ AC out, AC power	8 AC (4 and 4)	6 AC (2 and 4)	IC693UAA003
28 point DC in/relay and DC out, AC power	16 DC (4, 4, 4, and 4)	1 DC, 11 relay (1, 4, 1, 1, 1, 1, and 3)	IC693UDR005
28 point AC in/AC out, AC power	16 AC (4, 4, 4, and 4)	12 AC (2, 4, 2, and 4)	IC693UAA007
28 point DC in/relay out, DC power	16 DC (4, 4, 4, and 4)	1 DC, 11 relay (1, 4, 1, 1, 1, 1, and 3)	IC693UDR010
23 point analog DC in/relay and DC out, AC power	13 DC, 2 analog (4, 4, 4, and 2)	1 DC, 9 relay 1 analog (1, 4, 1, 1, 1, and 1)	IC693UAL006
14-point Expansion Unit DC in/relay out, AC power	8 DC (4 and 4)	6 relay (1, 1, and 4)	IC6963UEX011

Physical and Functional Characteristic (14-point PLC)

Weight: IC693UDR001/002/UAA003/UEX011	0.86 lbs (390 g)
Module Dimensions	Height: 3.2" (82mm) Depth: 3.0" (76mm) Width: 4.5" (115mm)
Typical Scan Rate	1.8 ms/K of logic (Boolean contacts)
Maximum number of Discrete Physical I/O Points	14 (8 inputs/6 outputs)
Maximum number of slave devices per network	8 (can be increased with a repeater)
Output Power Supplies **IC693UDR001/002/UEX011**	24VDC for input circuits & user devices, 100mA max. +5VDC on pin 5 of Serial Port, 155mA max (for UDR001/002 only)
Super cap backup for RAM	Provides data retention for 3–4 days with the power off at 25°C.

Physical and Functional Characteristic (28-point PLC)

Weight IC693UDR005 IC693UAA007 IC693UDR010	1.5 lbs (680 g.) 1.54 lbs (700 g.) 1.54 lbs (700 g.)
Module Dimensions	Height: 3.2" (82mm) Width: 8.6" (218mm) Depth: 3.0" (76mm)
Typical Scan Rate	1.0 ms/K of logic (Boolean contacts)
Real Time Clock accuracy 10°C 25°C 55°C	 4.54 sec./day 5.22 sec./day 10.66 sec/day
Maximum number of Discrete Physical I/O Points	28 (16 inputs/12 outputs)
Maximum number of slave devices per network	8 (can be increased with a repeater)
+24 VDC Output Power Supply (IC693UDR005/010) (for input circuits and user devices)	200 mA maximum
+5 VDC on pin 5 of Serial Ports Serial Port 1 Serial Port 2 Serial Ports 1 & 2 combined	 155mA maximum 100mA maximum 255mA maximum (The load on either port can exceed the individual ratings listed above, if the combined load does not exceed 255mA.) *See "Caution" below.*
Lithium battery lifetime	Shelf life (powered down) Up to 7 years typical at 30 °C Up to 5 years typical at 55 °C

Physical and Functional Characteristic (23-point Micro PLC, IC693UAL006)

Weight	1.52 lbs (690g)
Module Dimensions	Height: 3.2" (82mm) Width: 8.6" (218mm) Depth: 3.0" (76mm)
Typical Scan Rate	1.0 ms/K of logic (Boolean contacts)
Real Time Clock accuracy 10°C (with internal 15 °C rise) 25°C (with internal 15 °C rise) 55°C (with internal 15 °C rise)	 4.54 sec./day 5.22 sec./day 10.66 sec/day
Maximum number of Discrete Physical I/O Points	23 (13 inputs/10 outputs)
Maximum number of slave devices per network	8 (can be increased with a repeater)
+24 VDC Output Power Supply (for input circuits and user devices)	200 mA maximum
+5 VDC on pin 5 of Serial Ports Serial Port 1 Serial Port 2 Serial Ports 1 & 2 combined	 155mA maximum 100mA maximum 255mA maximum (The load on either port can exceed the individual ratings listed above, if the combined load does not exceed 255mA.) *See "Caution" below.*
Lithium battery lifetime	Shelf life (powered down) Up to 7 years typical at 30 °C Up to 5 years typical at 55 °C
Analog inputs	Two, differential
Input ranges	0 to 10 V (10.24V maximum) 0 to 20 mA (20.5mA maximum) 4 to 20 mA (20.5mA maximum)
Resolution: 0 to 10 V range 0 to 20 mA range 4 to 20 mA range	10 bits (1 LSB = 10mV) 9 bits (1 LSB = 40µA) 8+ bits (1 LSB = 40µA)
Accuracy	1% of full scale over full operating temperature range
Linearity	±3 LSB maximum
Common mode voltage	200 V maximum
Filter response time	20.2ms to reach 1% error for step response
Analog outputs	1, single-ended, non-isolated
Output ranges	0 to 10V (10.24V maximum) 0 to 20mA (20.5mA maximum) 4 to 20mA (20.5mA maximum)
Resolution	12 bits over 0 to 10V range (1 LSB = 2.5mV) 12 bits over 0 to 20mA range (1 LSB = 5µA) 11+ bits over 4 to 20mA range (1 LSB = 5µA)
Accuracy	±1% of full scale over full operating temperature range (0°C to 55°C)

AC Power Requirements

AC Power Requirements – (IC693UDR001, IC693UAA003/007, IC693UDR005, IC693UEX011)		
Range		100 -15% to 240 +10% VAC
Frequency		50 -5% to 60 +5% Hz
Hold-up		10 ms at 85 VAC
Inrush Time		2 ms for 40 A
Inrush Current	14-point Micro PLCs and 14-point Micro Expansion Unit	18 A maximum at 120 VAC 30 A maximum at 200 VAC 40 A maximum at 265 VAC
	28-point Micro PLCs	30 A maximum at 200 VAC 40 A maximum at 265 VAC
Input Current	14-point Micro PLCs	0.12 A typical at 200 VAC 0.25 A typical at 100 VAC
	28-point, DC In/Relay Out Micro PLCs	0.26 A typical at 100 VAC 0.12 A typical at 200 VAC
	28-point, AC In/AC Out Micro PLCs	0.16 A typical at 100 VAC 0.09 A typical at 200 VAC
Input Power Supply Rating	UDR001	35 VA
	UAA003	20 VA
	UAA007	25 VA
	UDR005	40 VA
	UEX011	35 VA
AC Power Requirements – (IC693UAL006)		
Range		100 -15% to 240 +10% VAC
Frequency		50 -5% to 60 +5% Hz
Hold-up		10 ms at 85 VAC
Inrush Time		2 ms for 40 A
Inrush Currents		35 A maximum at 200 VAC 46 A maximum at 265 VAC
Input Current		0.35 A typical at 100 VAC 0.22 A typical at 200 VAC
Isolation		1500VAC rms field side to logic (both power supply input and 24 VDC power supply output)
Input Power Supply Rating		50 VA

DC Power Requirements

DC Power Requirements – (IC693UDR002/010)		
Range	14-point Micro PLC	12 -15% to 24 +25% VDC 12 -15% to 24 +10% VAC
	28-point Micro PLCs	24 -20%, +25% VDC 24 -15%, +10% VAC
Hold-up	14-point Micro PLCs	4 ms at 10 VDC 10 ms at 12 VDC
	28-point Micro PLCs	2ms at 9.5 VDC
Inrush Current	14-point Micro PLC	65 A maximum at 24 VDC 81 A maximum at 30 VDC
	28-point Micro PLC[1]	65 A maximum at 24 VDC 81 A maximum at 30 VDC
Inrush Time	14-point Micro PLC	10 ms for 81 A
	28-point Micro PLC	10 ms for 81 A
Input Current	14-point Micro PLC[2]	0.4 A typical at 24 VDC 0.8 A typical at 12 VDC
	28-point Micro PLC	1.4 A typical at 24 VDC
Input Power Supply Rating	UDR002	15 W
	UDR010	20 W

Chapter 9

Allen Bradley MicroLogix 1000 PLC Programming

9.1 Using Basic Instructions

These instructions, when used in ladder programs represent hardwired logic circuits used for the control of a machine or equipment. The basic instructions are separated into three groups: bit, timer, and counter. Before you learn about the instructions in each of these groups, we suggest that you read the overview that precedes the group:

- Bit Instructions Overview
- Timer Instructions Overview
- Counter Instructions Overview

Examine if Closed (XIC)

Use the XIC instruction in your ladder program to determine if a bit is ON. When the instruction is executed, if the bit addressed is on (1), then the instruction is evaluated as true. When the instruction is executed, if the bit addressed is off (0), then the instruction is evaluated as false.

Bit Address State XIC Instruction

0 False

1 True

Examples of devices that turn on or off include:

- a push button wired to an input (addressed as I1:0/4)
- an output wired to a pilot light (addressed as O0:0/2)
- a timer controlling a light (addressed as T4:3/DN)

Examine if Open (XIO)

Use an XIO instruction in your ladder program to determine if a bit is Off. When the instruction is executed, if the bit addressed is off (0), then the instruction is evaluated as true. When the instruction is executed, if the bit addressed is on (1), then the instruction is evaluated as false.

Bit Address State XIO Instruction

0 True

1 False

Examples of devices that turn on or off include:

- motor overload normally closed (N.C.) wired to an input (I1:0/10)
- an output wired to a pilot light (addressed as O0:0/4)
- a timer controlling a light (addressed as T4:3/DN)

Output Energize (OTE)

Use an OTE instruction in your ladder program to turn On a bit when rung conditions are evaluated as true. An example of a device that turns on or off is an output wired to a pilot light (addressed as O0:0/4). OTE instructions are reset when:

- You enter or return to the REM Run or REM Test mode or power is restored.
- The OTE is programmed within an inactive or false Master Control Reset (MCR) zone.

Output Latch (OTL) and Output Unlatch (OTU)

OTL and OTU are retentive output instructions. OTL can only turn on a bit, while OTU can only turn off a bit. These instructions are usually used in pairs, with both instructions addressing the same bit. Your program can examine a bit controlled by OTL and OTU instructions as often as necessary.

Output Latch (OTL) and Output Unlatch (OTU)

OTL and OTU are retentive output instructions. OTL can only turn on a bit, while OTU can only turn off a bit. These instructions are usually used in pairs, with both instructions addressing the same bit. Your program can examine a bit controlled by OTL and OTU instructions as often as necessary.

Using OTL

When you assign an address to the OTL instruction that corresponds to the address of a physical output, the output device wired to this screw terminal is energized when the bit is set (turned on or enabled). When rung conditions become false (after being true), the bit remains set and the corresponding output device remains energized. When enabled, the latch instruction tells the controller to turn on the addressed bit. Thereafter, the bit remains on, regardless of the rung condition, until the bit is turned off (typically by a OTU instruction in another rung).

Using OTU

When you assign an address to the OTU instruction that corresponds to the address of a physical output, the output device wired to this screw terminal is de-energized when the bit is cleared (turned off or disabled). The unlatch instruction tells the controller to turn off the addressed bit. Thereafter, the bit remains off, regardless of the rung condition, until it is turned on (typically by an OTL instruction in another rung).

One-Shot Rising (OSR)

The OSR instruction is a retentive input instruction that triggers an event to occur one time. Use the OSR instruction when an event must start based on the change of state of the rung from false to true. When the rung conditions preceding the OSR instruction go from false to true, the OSR instruction will be true for one scan. After one scan is complete, the OSR instruction becomes false, even if the rung conditions preceding it remain true. The OSR instruction will only become true again if the rung conditions preceding it transition from false to true. The controller allows you to use one OSR instruction per output in a rung.

Entering Parameters

The address assigned to the OSR instruction is *not* the one-shot address referenced by your program, nor does it indicate the state of the OSR instruction. This address allows the OSR instruction to *remember* its previous rung state. Use a bit address from either the bit or integer data file. The addressed bit is set (1) for one scan when rung conditions preceding the OSR

instruction are true (even if the OSR instruction becomes false); the bit is reset (0) when rung conditions preceding the OSR instruction are false.

Timer Instructions Overview

Each timer address is made of a 3-word element. Word 0 is the control word, word 1 stores the preset value, and word 2 stores the accumulated value.

Entering Parameters

Accumulator Value (ACC)

This is the time elapsed since the timer was last reset. When enabled, the timer updates this continually.

Preset Value (PRE)

Specifies the value which the timer must reach before the controller sets the done bit. When the accumulated value becomes equal to or greater than the preset value, the done bit is set. You can use this bit to control an output device. Preset and accumulated values for timers range from 0 to +32,767. If a timer preset or accumulated value is a negative number, a runtime error occurs.

Timebase

The timebase determines the duration of each timebase interval. The timebase is selectable as 0.01 (10 ms) second or 1.0 second.

Timer Accuracy

Timer accuracy refers to the length of time between the moment a timer instruction is enabled and the moment the timed interval is complete. Timing accuracy is –0.01 to +0 seconds, with a program scan of up to 2.5 seconds. The 1-second timer maintains accuracy with a program scan of up to 1.5 seconds. If your programs can exceed 1.5 or 2.5 seconds, repeat the timer instruction rung so that the rung is scanned within these limits.

Timer On-Delay (TON)

Use the TON instruction to delay the turning on or off of an output. The TON instruction begins to count timebase intervals when rung conditions become true. As long as rung conditions remain true, the timer increments its accumulated value (ACC) each scans until it reaches the preset value (PRE). The accumulated value is reset when rung conditions go false, regardless of whether the timer has timed out.

Using Status Bits

This Bit	Is Set When	And Remains Set Until One of the Following
Timer Done Bit DN (bit 13)	accumulated value is equal to or greater than the preset value	rung conditions go false
Timer Enable Bit EN (bit 14)	rung conditions are true	rung conditions go false
Timer Timing Bit TT (bit 15)	rung conditions are true and the accumulated value is less than the preset value	rung conditions go false or when the done bit is set

When the controller changes from the REM Run or REM Test mode to the REM Program mode or user power is lost while the instruction is timing but has not reached its preset value, the following occurs:

- Timer Enable (EN) bit remains set.
- Timer Timing (TT) bit remains set.
- Accumulated value (ACC) remains the same.

On returning to the REM Run or REM Test mode, the following can happen:

Condition	Result
If the rung is true:	EN bit remains set. TT bit remains set. ACC value is reset.
If the rung is false:	EN bit is reset. TT bit is reset. ACC value is reset.

Retentive Timer (RTO)

Use the RTO instruction to turn an output on or off after its timer has been on for a preset time interval. The RTO instruction is a retentive instruction that lets the timer stop and start without

resetting the accumulated value (ACC). The RTO instruction retains its accumulated value when any of the following occurs:

- Rung conditions become false.
- You change controller operation from the REM Run or REM Test mode to the REM Program mode.
- The controller loses power.
- A fault occurs.

Using Status Bits

This Bit	Is Set When	And Remains Set Until One of the Following
Timer Done Bit DN (bit 13)	accumulated value is equal to or greater than the preset value	the appropriate RES instruction is enabled
Timer Timing Bit TT (bit 14)	rung conditions are true and the accumulated value is less than the preset value	rung conditions go false or when the done bit is set
Timer Enable Bit EN (bit 15)	rung conditions are true	rung conditions go false

When the controller changes from the REM Run or REM Test mode to the REM Program or REM Fault mode, or user power is lost while the timer is timing but not yet at the preset value, the following occurs:

- Timer Enable (EN) bit remains set.
- Timer Timing (TT) bit remains set.
- Accumulated value (ACC) remains the same.

Counter Instructions Overview

Each Counter address is made of a 3-word data file element. Word 0 is the control word, containing the status bits of the instruction. Word 1 is the preset value. Word 2 is the accumulated value.

Entering Parameters

Accumulator Value (ACC)

This is the number of false-to-true transitions that have occurred since the counter was last reset.

Preset Value (PRE)

Specifies the value which the counter must reach before the controller sets the done bit. When the accumulator value becomes equal to or greater than the preset value, the done status bit is set. You can use this bit to control an output device. Preset and accumulated values for counters range from –32,768 to +32,767, and are stored as signed integers. Negative values are stored in two's complement form.

Count Up (CTU)

The CTU is an instruction that counts false-to-true rung transitions. Rung transitions can be caused by events occurring in the program (from internal logic or by external field devices) such as parts traveling past a detector or actuating a limit switch. When rung conditions for a CTU instruction have made a false-to-true transition, the accumulated value is incremented by one count, provided that the rung containing the CTU instruction is evaluated between these transitions. The ability of the counter to detect false-to-true transitions depends on the speed (frequency) of the incoming signal.

The accumulated value is retained when the rung conditions again become false. The accumulated count is retained until cleared by a reset (RES) instruction that has the same address as the counter reset.

Using Status Bits

This Bit	Is Set When	And Remains Set Until One of the Following
Count Up Overflow Bit OV (bit 12)	accumulated value wraps around to –32,768 (from +32,767) and continues counting up from there	a RES instruction having the same address as the CTU instruction is executed OR the count is decremented less than or equal to +32,767 with a CTD instruction
Done Bit DN (bit 13)	accumulated value is equal to or greater than the preset value	the accumulated value becomes less than the preset
Count Up Enable Bit CU (bit 15)	rung conditions are true	rung conditions go false OR a RES instruction having the same address as the CTU instruction is enabled

The accumulated value is retained after the CTU instruction goes false, or when power is removed from and then restored to the controller. Also, the on or off status of counter done, overflow, and underflow bits is retentive. The accumulated value and control bits are reset when the appropriate RES instruction is enabled. The CU bits are always set prior to entering the REM Run or REM Test modes..

Count Down (CTD)

The CTD is a retentive output instruction that counts false-to-true rung transitions. Rung transitions can be caused by events occurring in the program such as parts traveling past a detector or actuating a limit switch.

When rung conditions for a CTD instruction have made a false-to-true transition, the accumulated value is decremented by one count, provided that the rung containing the CTD instruction is evaluated between these transitions. The accumulated counts are retained when the rung conditions again become false. The accumulated count is retained until cleared by a reset (RES) instruction that has the same address as the counter reset.

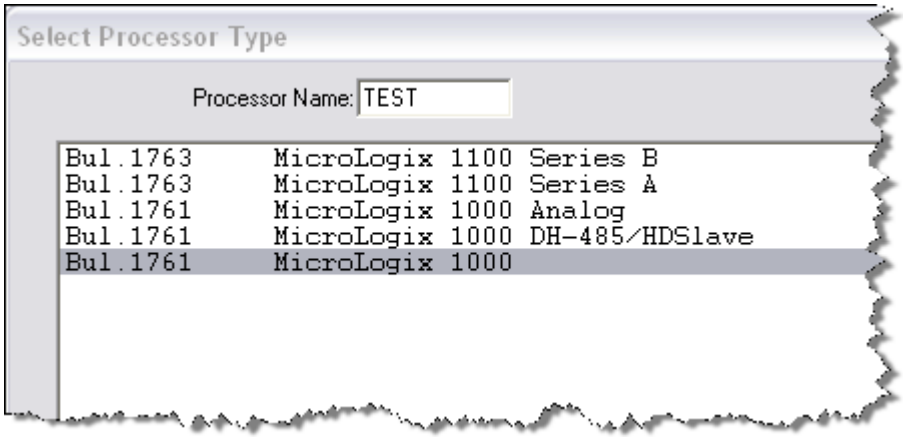
Using Status Bits

This Bit	Is Set When	And Remains Set Until One of the Following
Count Down Underflow Bit UN (bit 11)	accumulated value wraps around to +32,768 (from –32,767) and continues counting down from there	a RES instruction having the same address as the CTD instruction is enabled. OR the count is incremented greater than or equal to +32,767 with a CTU instruction
Done Bit DN (bit 13)	accumulated value is equal to or greater than the preset value	the accumulated value becomes less than the preset
Count Down Enable Bit CD (bit 14)	rung conditions are true	rung conditions go false OR a RES instruction having the same address as the CTD instruction is enabled

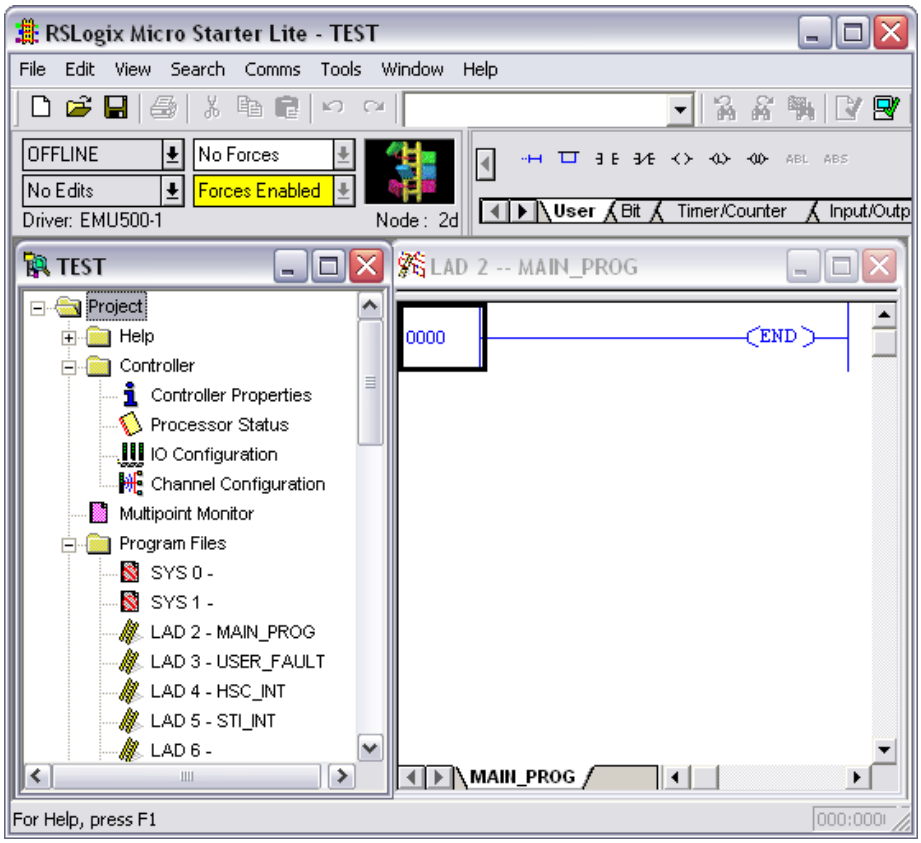
The accumulated value is retained after the CTD instruction goes false, or when power is removed from and then restored to the controller. Also, the on or off status of counter done, overflow, and underflow bits is retentive. The accumulated value and control bits are reset when the appropriate RES instruction is executed. The CD bits are always set prior to entering the REM Run or REM Test modes.

RSLogix Micro Starter Lite

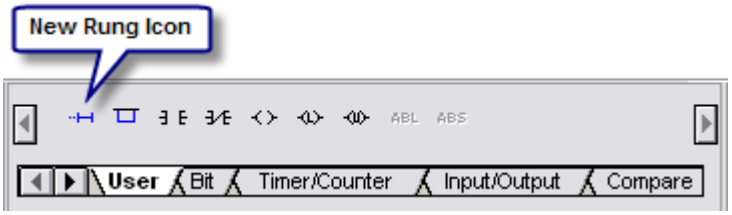
Now for the moment we are creating some ladder logic. Open the RSLogix Micro software with the *START > All Programs > Rockwell Software > RSLogix Micro English > RSLogix Micro English* shortcut. Create a brand new project by pulling down the *File* menu and selecting *New.* Every project must start with a designated processor.



It has chosen the simplest MicroLogix 1000 and then clicked the OK button. If you ever work with the purchased version of RSLogix 500 then there will be a lot more items in this hardware list. A blank project now opens up.



Let's make a simple rung to test in the emulator. Make sure the cursor is on the rung with the END on it and then click the New Rung icon in the instruction toolbar.

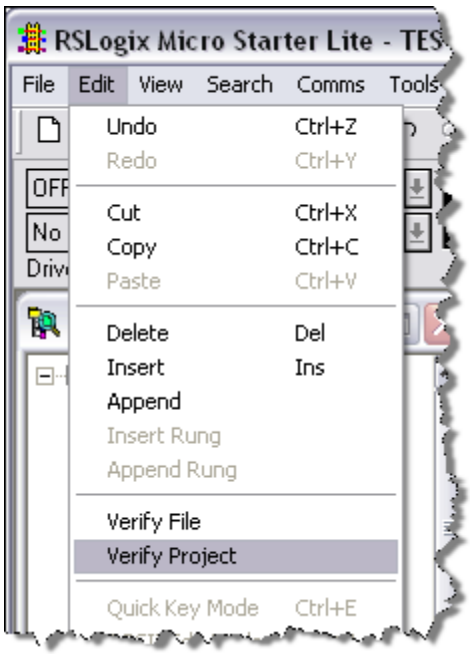


Now click on the "Examine if Closed" instruction to add it to the rung. Double click on the question mark above it and enter I:0/0 as its input address. Just leave the description pop up box empty by clicking OK.

Next, click on the Output Energize instruction to add it to the right side of the rung. Double click on the question mark above it and enter O:0/0 as its output address. Just leave the description pop up box empty by clicking OK.

You should now have something like below.

The next very important step is to verify the project with the *Edit* > *Verify Project* menu item. This will compile the project and get it ready for the emulator.



Save the project as something like Test.RSS in an easy place to find like My Documents.

9.2 Using Comparison Instructions

Comparison instructions are used to test pairs of values to condition the logical continuity of a rung. As an example, suppose a LES instruction is presented with two values. If the first value is less than the second, then the comparison instruction is true. To learn more about the compare instructions, we suggest that you read the Compare Instructions Overview that follows.

Equal (EQU)

Use the EQU instruction to test whether two values are equal. If source A and source B are equal, the instruction is logically true. If these values are not equal, the instruction is logically false. Source A must be a word address. Source B can be either a constant or word address. Negative integers are stored in two's complement form.

Not Equal (NEQ)

Use the NEQ instruction to test whether two values are not equal. If source A and source B are not equal, the instruction is logically true. If the two values are equal, the instruction is logically false. Source A must be a word address. Source B can be either a constant or word address. Negative integers are stored in two's complement form.

Less Than (LES)

Use the LES instruction to test whether one value (source A) is less than another (source B). If the value at source A is less than the value of source B the instruction is logically true. If the value at source A is greater than or equal to the value of source B, the instruction is logically false. Source A must be a word address. Source B can be either a constant or word address. Negative integers are stored in two's complement form.

Less Than or Equal (LEQ)

Use the LEQ instruction to test whether one value (source A) is less than or equal to another (source B). If the value at source A is less than or equal to the value of source B, the instruction is logically true. If the value at source A is greater than the value of source B, the instruction is logically false. Source A must be a word address. Source B can be either a constant or word address. Negative integers are stored in two's complement form.

Greater Than (GRT)

Use the GRT instruction to test whether one value (source A) is greater than another (source B). If the value at source A is greater than the value of source B, the instruction is logically true. If the value at source A is less than or equal to the value of source B, the instruction is logically false. Source A must be a word address. Source B can be either a constant or word address. Negative integers are stored in two's complement form.

Greater Than or Equal (GEQ)

Use the GEQ instruction to test whether one value (source A) is greater than or equal to another (source B). If the value at source A is greater than or equal to the value of source B, the

instruction is logically true. If the value at source A is less than the value of source B, the instruction is logically false. Source A must be a word address. Source B can be either a constant or word address. Negative integers are stored in two's complement form.

Masked Comparison for Equal (MEQ)

Use the MEQ instruction to compare data of a source address with data of a reference address. Use of this instruction allows portions of the data to be masked by a separate word.

Entering Parameters

- **Source** is the address of the value you want to compare.
- **Mask** is the address of the mask through which the instruction moves data. The mask can be a hexadecimal value (constant).
- **Compare** is an integer value or the address of the reference. If the 16 bits of data at the source address are equal to the 16 bits of data at the compare address (less masked bits), the instruction is true. The instruction becomes false as soon as it detects a mismatch. Bits in the mask word mask data when reset; they pass data when set.

Limit Test (LIM)

Use the LIM instruction to test for values within or outside a specified range, depending on how you set the limits.

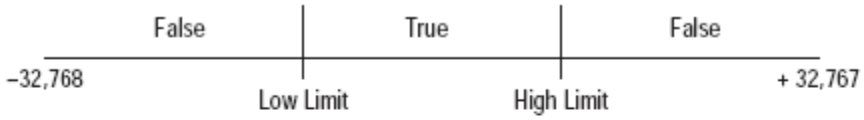
Entering Parameters

The Low Limit, Test, and High Limit values can be word addresses or constants, restricted to the following combinations:

- If the Test parameter is a constant, both the Low Limit and High Limit parameters must be word addresses.
- If the Test parameter is a word address, the Low Limit and High Limit parameters can be either a constant or a word address.

True/False Status of the Instruction

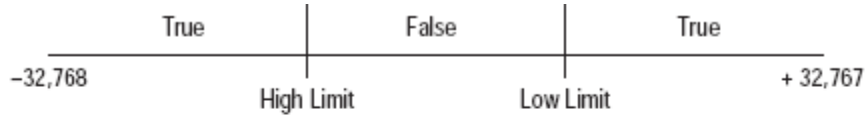
If the Low Limit has a value equal to or less than the High Limit, the instruction is true when the Test value is between the limits or is equal to either limit. If the Test value is outside the limits, the instruction is false, as shown below.



Example, low limit less than high limit:

Low Limit	High Limit	Instruction is True when Test value is	Instruction is False when Test value is
5	8	5 through 8	–32,768 through 4 and 9 through 32,767

If the Low Limit has a value greater than the High Limit, the instruction is false when the Test value is between the limits. If the Test value is equal to either limit or outside the limits, the instruction is true, as shown below.



Example, low limit greater than high limit:

Low Limit	High Limit	Instruction is True when Test value is	Instruction is False when Test value is
8	5	–32,768 through 5 and 8 through 32,767	6 and 7

References

1. GX Developer FX, Programming and Documentation System, Beginner's Manual, Mitsubishi Electric Industrial Automation.
2. GE Fanuc Automation, Programmable Control Products Logicmaster 90 Series 90-30/20/Micro, Programming Software User's Manual, GFK-0466L.
3. Programmable logic controllers, Basic level TP301, Textbook, Festo Didactic, R. Bliesener, F. Ebel, C. Löffler, B. Plagemann, H. Regber, E. v. Terzi, A. Winter.
4. CP1L CPU Unit, GETTING STARTED GUIDE, Omron.
5. Simatic, S7-200 Programmable Controller, CPU 210, System Manual.
6. GE Fanuc Automation, Programmable Control Products, Series 90™, Micro PLC User's Manual GFK-1065F.
7. Allen-Bradley MicroLogix 1000, Programmable Controllers, User Manual, (Bulletin 1761 Controllers).
8. FX Series Programmable Controllers, Programming Manual, Revision J, November 1999.
9. SINUMERIK 840D, C-PLC Programming, Description of Functions 03.96 Edition, Manufacturer Documentation.
10. Basics of PLC Programming, Industrial Control Systems, Fall 2006.

Support Me

1. Add to circles on Google+

plus.google.com/107166042091915882790

2. Become a fan on Facebook

facebook.com/plcsimulation

3. Follow us on Twitter

twitter.com/program_plc